DOI: 10.14750/ME.2023.027

MISKOLCI EGYETEM

GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

MISKOLCI EGYETEM

Hatvany József Informatikai Tudományok Doktori Iskola

Doktori iskola vezetője:

Prof. Dr. Szigeti Jenő

Fuzzy szabály-interpolációs következtetési és tudásreprezentációs formák beágyazott rendszerekben

Doktori értekezés

Témavezető: Dr. habil. Vásárhelyi József

Név: Bartók Roland

Neptun: AQCCRH

2023

DOI: 10.14750/ME.2023.027

Tartalomjegyzék

DOI: 10.14750/ME.2023.027

DOI: 10.14750/ME.2023.027

Nyilatkozat

Alulírott Bartók Roland büntetőjogi felelősségem tudatában kijelentem, hogy a Hatvany József Informatikai Tudományok Doktori Iskola Doktori Iskolába beadott PhD értekezés önálló munkám eredménye, az irodalmi hivatkozások egyértelműek és teljesek.

Dátum: …….. ……..

Aláírás

Köszönet

A kutatás több éves időszakában sok segítséget kaptam kollégáktól, barátoktól.

Köszönet illeti a témavezetőmet, Dr. Vásárhelyi Józsefet a kutatómunkában nyújtott támogatásáért, segítő tanácsaiért.

Köszönöm Dr. Kovács Szilveszternek, hogy a mesterképzés idején felhívta a figyelmemet az etorobotika és fuzzy interpolációs módszerek témakörre és a kutatási időszak alatt is bármikor fordulhattam hozzá segítségért.

Köszönöm a segítségét Piller Imrének, aki szakmai és elméleti kérdéseimre készségesen válaszolt ezzel is segítve a kutatás előrehaladását.

Köszönöm a Miskolci Egyetem Automatizálási és Infokommunikációs Intézet valamennyi kollégájának, hogy segítették munkámat.

Köszönöm a szüleim és családom támogatását, hogy lehetővé tették az egyetemi tanulmányaimat.

Végül, de nem utolsó sorban köszönöm a barátoknak támogatásukat és kitartásukat.

1. Bevezetés

A technika fejlődésével a robotok megjelentek a szórakoztató iparban, vagyonvédelemben és személyvédelemben (idősgondozás, felügyelet) területén, mint megfigyelő eszközök. A beágyazott rendszerek számítási kapacitása már lehetővé teszi egyre összetettebb feladatokat ellátó mobil robotrendszerek fejlesztését. Így a robotok szaktudást nem igénylő, sokszor monoton, esetleg egészségre ártalmas munkafolyamatot is átvehetnek, amelyeket jelenleg emberek végeznek. Az emberi munkaerőt így más, változatos, kreativitást is igénylő feladatokra lehet használni.

Több kutatás is foglalkozik az ember-gép kapcsolattal, miként reagál az ember, ha egy minimális intelligenciát hordozó géppel kell valamilyen kapcsolatba lépnie vagy miként reagáljon a gép az emberek jelenlétére, hogyan kerülhetőek el például a balesetek. Már jelenleg is vannak olyan rendszerek, amelyek képesek együttműködni az emberekkel [1]. Ilyenek a ritkább exoskeleton rendszerek [2], amelyek egy külső vázként képesek az emberek erejét megnövelni. Ezek nem mondhatók intelligens gépeknek, emberi irányítás szükséges a működtetésükhöz. Népszerűbbek az együttműködő (kollaboratív) robotkarok, amelyeknek a legismertebb gyártója az Universal Robots. Ezek a robotkarok más szabványoknak felelnek meg, mint a hagyományos robotkarok. Működésük során nem szükséges őket védőkalitkába helyezni, sebességük korlátozott a hagyományos robotkarokhoz képest és nagyon gyorsan képesek felmérni az ütközés valószínűségét, ekkor képesek vészleállást végrehajtani, így nem okoznak nagyobb fizikai behatást, mintha két ember ütközne össze munka közben. Ennek megfelelő a teherbírásuk is, körülbelül 10 kg. Azonban az emberekkel együttműködve a monoton feladatokat átvéve képesek a gyártósoron növelni a termelés hatékonyságát, lásd [3] és [4].

Ezek a robotok még nem intelligens rendszerek, előre programozott mozdulatsort hajtanak végre, bizonyos feltételeket figyelembe véve.

Jelenleg is működnek már a gyakorlatban elterjedt rendszerek, amelyek például gyártósori logisztikai kiszolgáló szerepet látnak el (alkotóelemek, hulladék önálló szállítása), mezőgazdasági munkák önálló végzésére képesek [5], önvezető járművek, autonóm kórházi betegszállító robotok [6]. Az ilyen robotok már intelligens rendszereknek nevezhetőek. A szociális robotok területébe tartoznak, ahogy az egyéb

társ jellegű gépek is. Működésük során a környező világról a lehető legtöbb megfigyelést kell gyűjteniük és ezek alapján döntést hozni a saját működésükre vonatkozóan. A szociális robotika az a terület, amelynél már robot külső megjelenésén túl fontos a robot viselkedése is, lásd: [7].

Egy szociális robot a környezetből érkező megfigyelések szinte bármilyen kombinációjára kell reagáljon, nem maradhat döntésképtelen, nem állhat meg a legtöbb helyzetben. Erre ad megoldást az etorobotika tudománya és a robot viselkedésének leírása, amely fuzzy szabály interpoláció alapú változata kerül bemutatásra. Az etorobotika tudománya foglalkozik azzal, hogy a robotnak milyen viselkedésmintát kell követnie ahhoz, hogy az emberek elfogadják illetve az adott helyzetben milyen viselkedésminta a legmegfelelőbb a feladat végrehajtására. Létező viselkedések megfigyelésén alapszik, olyan viselkedésmintákat használnak fel, amelyek kiszámíthatók mégis képes a segítő vagy társrobot feladatát ellátni. Ilyen például az ember-robot interakció vizsgálatára alkalmazott kutyák viselkedésmintája az Eötvös Lóránd Tudományegyetem etorobotikai kutatásaiban [8].

Az etorobotikai viselkedésmodellek leírását ezen etológusok a Miskolci Egyetem kutatóinak közreműködésével fuzzy szabály interpolációval oldották meg. A fuzzy szabály interpoláció lehetővé teszi, hogy a kívánt viselkedés, amely túl összetett, hogy bármilyen matematikai módszerrel leírható legyen. Olyan helyzetekben is döntésképes marad a rendszer az interpolációnak köszönhetően, amelyre nincs szabály leírva. A szabály alapú leírás kiindulhat egy szakértői szabálybázisból vagy ismert viselkedésmintából generált szabályokból. [9] és [8]

A disszertáció célja a viselkedések leírására alkalmazott FIVE (**F**uzzy **I**nterpolation in **V**ague **E**nvironment – fuzzy interpoláció homályos környezetben) módszer vizsgálata többek között mobil robotok irányításában alkalmazható beágyazott rendszereken. A vizsgálat célja főként a számítási hatékonyság javítását segítő adatok gyűjtése.

Ennek érdekében elkészítettem a µFRI C nyelvű könyvtárat, amely a FIVE módszer megvalósítása főként beágyazott rendszerekre. Megvizsgáltam a µFRI könyvtár párhuzamosíthatóságát a jelenleg elérhető és népszerű többprocesszoros „rendszer a chip”-en megoldásokon (SoC – **S**ystem **o**n **C**hip), a sok szabálybázist tartalmazó leírások optimális rendszerkihasználása szempontjából. Megvizsgáltam, hogy az adott

többprocesszoros környezetben hogyan érdemes szétosztani a szabálybázisokat úgy, hogy az a lehető legnagyobb számítási sebességnövekedéssel járjon. Továbbá a dolgozat módszert mutat a gyors végrehajtást igénylő alkalmazások esetén a FIVE módszer áramköri megvalósíthatóságára FPGA-n (**F**ield **P**rogrammable **G**ate **A**rray). A kutatás tartalmazza a FIVE módszerhez tartozó számítások megvalósítását és mintát a tudásbázis tárolására szolgáló adatbázisra. Végül megoldást javasolok annak kapcsán, hogy a FIVE módszer nem csak a viselkedések leírására alkalmas, hanem szenzoradatok elemzésével a robotot körülvevő akadályok érzékelésére is felhasználható osztályozó eljárásként. Az eredmények összevetésre kerülnek az ismert feltételes valószínűséget és nagy mennyiségű tanító mintát alkalmazó Naiv-Bayes osztályozóval.

A dolgozat 9 fejezetből áll, amelyből az első a „*Bevezetés”*, ezt követi a „*Viselkedés alapú irányítás*” ismertetése, klasszikus fuzzy és fuzzy interpolációs módszerek áttekintése, fuzzy logikai irányítás és kapcsolata a viselkedés alapú irányítással. „*A FIVE módszer”* című fejezet a FIVE fuzzy interpolációs módszer alapjait foglalja össze és ismerteti a viselkedés leíró nyelv felépítését. „*A µFRI könyvtár”* című fejezet az első tézis keretében készített beágyazott rendszerekre kihegyezett FIVE módszer C nyelvű változatát írja le, amely képes futás közbeni paraméter és szabálybázis módosításra is. A µFRI könyvtár többprocesszoros működésének vizsgálata a második tézisben kerül leírásra „A FIVE módszer párhuzamos számítási sebességének gyorsítása” című fejezetben. A FIVE módszer FPGA-ra készített, dinamikus paraméter-változtatást tesz lehetővé a harmadik tézis keretében, amelyet „*A FIVE módszer hardveres implementálása”* című fejezet foglal össze. Végül „*A FIVE módszer, mint osztályozó eljárás alkalmazása”* című fejezetben összehasonlításra kerül a FIVE módszer, mint szabály alapú osztályozó a Naiv-Bayes osztályozóval egy adott gyakorlati példában. Ez a fejezet egyben a µFRI könyvtár alkalmazáspéldája is a negyedik tézisként. Végül az értekezést „*Az értekezés tézisei”* és az „*Összefoglalás”* zárják a disszertációt.

2. Viselkedés alapú irányítás

Ebben a fejezetben a viselkedés alapú irányítás alapjait írom le. Röviden ismertetem a klasszikus fuzzy logika alapjait, leírom a fuzzy szabály interpolációs módszereket és sor kerül a fuzzy logikával történő irányítás elméletének leírására.

Mi az a viselkedés alapú irányítás és hogyan kapcsolódik a fuzzy logikához?

A viselkedés alapú irányítás elve, hogy az ágens több, az adott helyzethez, célhoz illeszkedő viselkedésminták közül választhat. Az ágens lehet szimulált vagy fizikailag létező robot is. A viselkedésminták felhasználják az érzékelők adatait, mint megfigyeléseket és a viselkedés célja szerinti döntést hoznak az ágens irányítására, amely lehet akár elmozdulás, beavatkozó szervek mozgatása vagy egyéb a jelzés a környezet felé. A pillanatnyi megfigyelésekre reagálva mindig kiszámításra kerül az összes létező viselkedése az ágensnek. Ezek közül különböző módszerek segítségével kell választania az adott helyzet kezelésére. Fuzzy alapú irányítás esetén a viselkedések és azok (és/vagy akciók) közötti választás is történhet fuzzy logikával. Így az adott helyzethez alkalmazkodva, minden viselkedés, hatással lesz az ágens válasz reakciójára, csak eltérő mértékben. Ezzel elérhető a folyamatos átmenet az egyes tevékenységek között. Felmerülhet olyan probléma, hogy egymásnak ellentmondó viselkedések jelennek meg a rendszerben, ennek feloldása az akció kiválasztási algoritmus vagy viselkedés feladata.

A viselkedés alapú irányítás alkalmazható: etológiai területen, mesterséges élet szimulálásában, virtuális valóság megvalósításában, grafikai szimulációban, avatárok mozgatására, szoftveres ágensekben és robotikai területen. Ezeken az összetett viselkedéseket igénylő alkalmazásokon kívül használható egyszerűbb szabályozási feladatok ellátására, nemlineáris rendszerek szabályozóinak megvalósítására.

A viselkedés alapú irányítás a megfigyelések előfeldolgozása, viselkedésminták számítása és az akció kiválasztásához részszámítások egyértelmű párhuzamosítási lehetőséget hordoznak. A felsorolt alkalmazási irányok mindegyike igényli a viselkedések számításának gyors végrehajtását, vagy a nagyszámú viselkedésminta miatt vagy pedig a valós idejű gyors működés igénye miatt.

A viselkedések leírása és az akció kiválasztás megadására meglehetősen sok lehetőség létezik. Ezek közül a továbbiakban a fuzzy alapú viselkedés leírás

számításának vizsgálatáról, hardveres gyorsítási lehetőségéről lesz szó, robotikához azon belül is etorobotikához kapcsolódó témában, bővebben lásd [10], [8], [11] és [12].

2.1. Fuzzy logika

Az emberi fogalmakat, mint például a kis gázzal előre vagy hamarosan délután két óra lesz, a hagyományos logikával nem lehet a számítógép számára értelmezhető formában leírni. Az ilyen fogalmak leírásához használható a fuzzy halmaz, fuzzy logika, amely nem csak diszkrét 0 és 1 értéket vehet fel, hanem a fuzzy halmaz elemeihez egy 0 és 1 közé eső valós számot rendel, amely a halmaz adott tagjáról azt mondja meg, hogy milyen mértékben tagja az adott halmaznak. A fuzzy halmazokról először L.otfi A. Zadeh tett említést az 1965-ös „Fuzzy sets" című cikkében [13].

A fuzzy logika két részre osztható, egyik részét képezik a fuzzy halmazok, másikat pedig a fuzzy halmazokból alkotott szabályok. Ezen halmazok segítségével leírhatók a gép számára az emberi fogalmak, mint például, hogy egy sebesség gyors vagy a hőmérséklet alacsony. Természetes nyelvi mennyiségi fogalmak írhatók le a gép számára a fuzzy halmazok segítségével úgy, mint „alacsony hőmérséklet", „közepes sebesség", stb.

A fuzzy logika meghatározza a fuzzy halmazokon végzett műveleteket. Fuzzy implikációval összekapcsolva a halmazokat, fuzzy szabályok jönnek létre. Klasszikus fuzzy logika esetén, amikor minden lehetséges megfigyeléshez tartozik szabálypont, a szabálybázis teljesnek tekinthető. A szabálybázis tudásreprezentációs forma. Ha-akkor típusú állítások és következtetésekbe foglalt kijelentések tehát a szabályok gyűjteménye. Ez feladattól függően nagyon sok szabályt jelenthet. Ezzel együtt jelentős számítási teljesítményigény növekedés tapasztalható.

Ritka a szabálybázis, ha nem minden megfigyeléshez tartozik szabálypont. Ekkor azonban lehet olyan megfigyelés, amelynél nem megbecsülhető a rendszer működése, hibás működést eredményezhet, lásd [14] és [15]. Ritka szabálybázis akkor alakulhat ki, amikor a szabályokkal leírni kívánt rendszer működése nem teljes mértékben ismert, túlságosan összetett. Ekkor a szabályokat létrehozó szakértő vagy szakértő rendszer nem képes minden eshetőséget szabályba foglalni.

2.1.1. Néhány fuzzy alapfogalom, jelölés

Az alapfogalmak, jelölések magyarázatát Johanyák Zsolt Csaba PhD értekezése [14] és Kovács Szilveszter: Fuzzy logikai irányítás című dolgozata [15] alapján készítettem el.

Alaphalmaz: Egy nem fuzzy halmaz, amelyen a fuzzy halmazok elemei értelmezhetőek. Jelölése: X vagy U, a továbbiakban X.

Nyelvi kifejezés (érték): Egy emberi nyelven megfogalmazott szó, szimbólum, amely egy fuzzy halmaz neve, címkéje.

Nyelvi változó: A nyelvi kifejezéseket tároló speciális változó.

Fuzzy halmaz: Nyelvi kifejezések bizonytalanságának leírására szolgál. A crisp, éles vagyis hagyományos halmazok kiterjesztése, ahol egy alaphalmaz-elem halmazhoz tartozásának mértékét is tartalmazza, amely egy 0 és 1 közé eső valós szám. A fuzzy halmaz jelölése: az angol ABC egy nagybetűje.

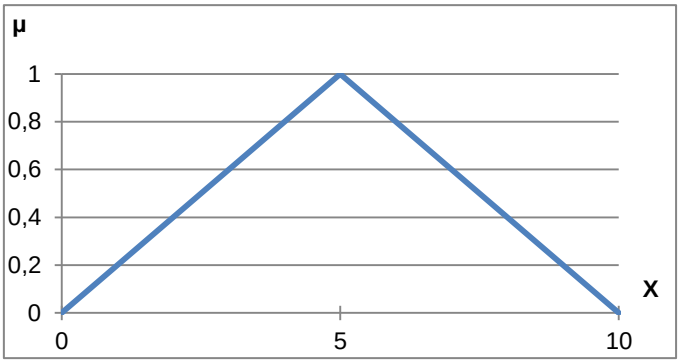
Tagsági függvény (lásd (1) egyenlet): Megadja, hogy az *X* alaphalmaz elemei milyen mértékben tartoznak az adott *A* fuzzy halmazhoz egy 0 és 1 közötti valós szám hozzárendelésével. Ez esetben a 0 azt jelenti, hogy az adott elem nem része a halmaznak, az 1 pedig azt, hogy az elem része a halmaznak. Jelölése: μ_A.

$$\mu_A : X \rightarrow [0,1] \tag{1}$$

Egy fuzzy halmaz megadása (1. ábra) az elemeinek és a hozzájuk tartozó tagsági függvények megadásával történik. Diszkrét esetben:

$$\mu_A : X \rightarrow A = \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_n/x_n\,, \qquad x_i \epsilon X\ [0,1] \tag{2}$$

ahol az x_i egy halmazelem, a μ_i pedig a halmazelem tagsági függvénye. A „ / ” operátor ebben az esetben nem osztást jelent, hanem a tagsági függvény és a halmazelem összetartozását, a „ + ” operátor pedig a felsorolásban segít.



1. ábra Példa fuzzy halmaz ábrázolására

Folytonos elemű halmazok esetén:

$$A = \int_X \mu_{(x)}/x, \qquad x \epsilon X \tag{3}$$

ahol a „ / ”, „ ∫ ” szimbólumok csak a jelölést segítik.

α-vágat: Egy fuzzy halmaz α-vágata egy éles halmaz, amely az alábbi módon adható meg (az α-vágat nem fuzzy halmaz):

$$[A]_\alpha = \{x \epsilon X | \mu_A(X) \geq \alpha; \alpha \epsilon (0,1]\} \tag{4}$$

Fuzzy halmaz magja: Az a fuzzy halmaz, amely tartalmazza az A fuzzy halmaz minden olyan elemét, amely tagsági függvényének értéke 1:

$$[A]_1 = \text{kernel}A = \{x \epsilon X | \mu_A(x) = 1\} \tag{5}$$

Fuzzy halmaz hordozója: Az a fuzzy halmaz, amely tartalmazza az A fuzzy halmaz minden olyan elemét, amely tagsági függvényének értéke nagyobb, mint nulla:

$$[A]_{0+} = \text{supp}(A) = \{x \epsilon X | \mu_A(x) > 0\} \tag{6}$$

Fuzzy halmaz magassága: Egy A halmazban lévő tagsági függvény legnagyobb értéke, [0,1] intervallumba eső valós szám:

$$\text{height}(A) = \max_X(\mu_A(x)) \tag{7}$$

Normalizált fuzzy halmaz: Az A normalizált fuzzy halmaz magassága egy:

$$\text{height}(A) = 1 \tag{8}$$

Konvex fuzzy halmaz: Konvex egy *X* univerzumon értelmezett *A* fuzzy halmaz, ha α-vágatai mind konvex halmazok.

$$\mu_A(\lambda x + (1-\lambda y)) \geq \min(\mu_A(x), \mu_A(y), \forall x, y \in X \text{ és } \forall \lambda \epsilon [0,1] \tag{9}$$

Fuzzy szám: Olyan fuzzy halmaz, amely a valós számok halmazán értelmezett, konvex, normalizált, tagsági függvénye folytonos.

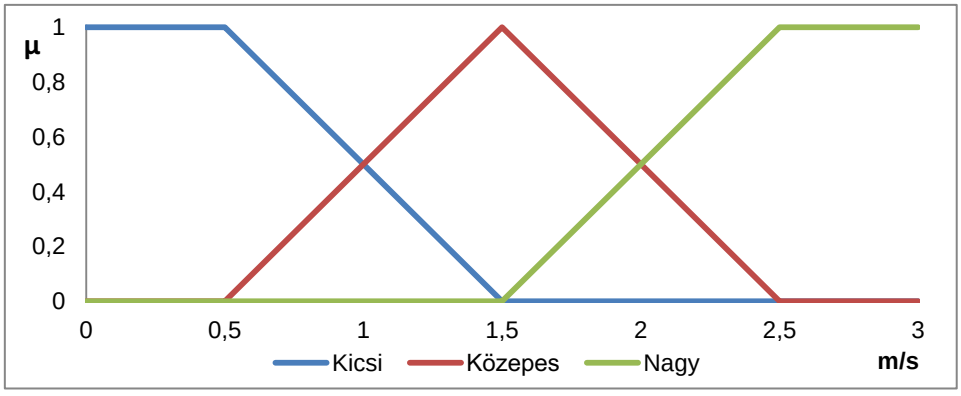
Egyelemű (singleton) fuzzy halmaz: Grafikus képe egy függőleges egyenes. Az *X* alaphalmaz elemei közül csak egyetlen elemet tartalmaz, amelynek tagsági értéke 1.

Az alaphalmaz lefedettség mértékét (ε) az *X* alaphalmazon véges számú fuzzy halmazok adják, amelyekre igaz, hogy:

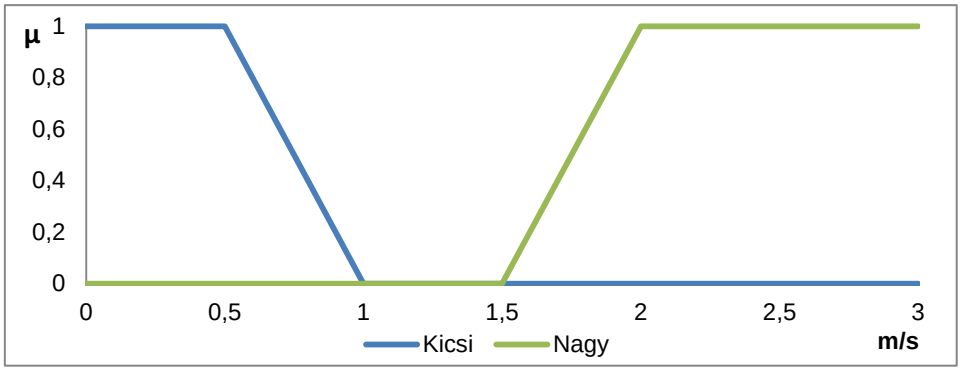
$$\arg\max_{\varepsilon} \left(\forall x \in X, \exists j \in \{1, \dots, n\}: \mu_{A_j}(x) \geq \varepsilon \right), \qquad \varepsilon \epsilon [0,1] \tag{10}$$

Fuzzy partíció: Az *X* alaphalmaz partíciója az a halmazcsalád, amely halmazai biztosítják a teljes fedettséget (2. ábra). Ritka partíció (3. ábra) esetén ez a feltétel nem teljesül.

Ritka partíció esetén csak interpolációs módszerek alkalmazásával lesz minden esetben döntésképes a fuzzy szabályrendszer. A ritka partíció és a ritka szabálybázis akkor fordulhat elő, amikor szakértői tudásbázis alapján épül fel a szabályrendszer vagy automatikusan létrehozott. A legfontosabb szabályok és halmazelemek kerülnek rögzítésre, ami nem biztos, hogy elegendő a teljes fedettség eléréshez.



2. ábra Teljes, fedő partíció példa



3. ábra Ritka partíció példa

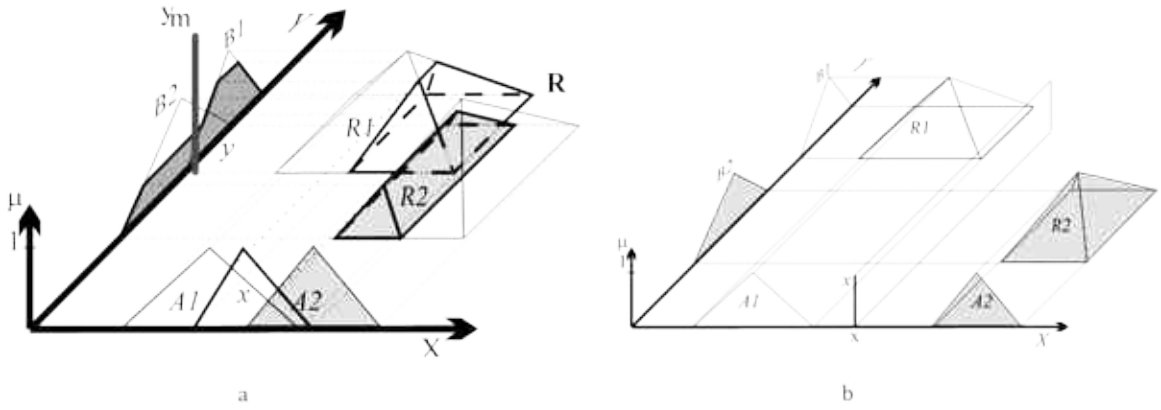
Fuzzifikáció: Az a művelet, amely során a megfigyelésből származó mért értéket (egy valós szám) fuzzy számmá alakít át.

Defuzzifikáció: A fuzzifikáció ellentétes művelete, a fuzzy számot irányításra, felhasználásra alkalmas valós számmá alakítja.

2.2. Fuzzy szabály interpolációs módszerek

A klasszikus fuzzy esetén a legnagyobb problémát az információhiány vagy éppen a túl bonyolult leírás okozza. Utóbbi akkor fordul elő, ha túl sok szabállyal írható le egy rendszer a bonyolultsága miatt vagy azért, mert nem optimalizált. A sok szabály sok számítási időt vagy nagyobb számítási teljesítményt igényel, de döntésképtelen marad a fuzzy logikai vezérlés. Ahhoz, hogy ez ne forduljon elő, teljes szabálybázist kell alkalmazni, amely az egész állapotteret lefedi szabályokkal.

Az információhiányos állapot akkor fordul elő, amikor a rendszer bonyolultsága miatt nem ismert teljesen, mint például egy állat viselkedése. Emiatt csak a legmeghatározóbb szabályok megadása lehetséges. A túl sok szabályt tartalmazó rendszer optimalizálása esetén is előfordulhat, hogy csak bizonyos szabályok elhagyásával lehet csökkenteni a szabályszámot. Ez bizonyos esetben ritka (sparse) szabálybázist eredményez, ilyenkor előfordulhat olyan megfigyelés, amelyhez nem tartozik szabály (4. ábra). Ekkor a rendszer olyan döntést hoz, amely hibás, nem meghatározható állapotot idéz elő. Erre a problémára adnak megoldást a különféle fuzzy interpolációs módszerek (fuzzy szabály interpoláció – **F**uzzy **R**ule **I**nterpolation, FRI) – lásd [16].



4. ábra Teljes fedő (a) és ritka szabálybázis (b), ahol az x megfigyeléshez nem tartozik szabály [67]

Az FRI módszerek közös jellemzője, hogy a meglévő szabályok alapján számítják ki a hiányzó szabályhoz tartozó döntést. Az eljárást két nagy csoportra lehet bontani, az egylépéses és a kétlépéses interpolációk csoportjára.

Az ismertebb FRI módszerek egyike a **KH**-módszer (**K**óczy-**H**irota módszer), mely helyes működéséhez az antecedensek és konzekvensek is konvex és normál fuzzy halmazoknak kell lenniük, amelyekre fennáll a részben rendezés és a tartójuk a halmazon belül marad. A KH módszer a megadott alsó és felső határpontok közötti Euklidészi távolságon és a fuzzy halmazok alfa vágatán alapszik. Működését tekintve lineárisan interpolál a megadott szomszédos pontok között ezért szükséges a halmazok részben rendezése, ellenkező esetben hibás értékeket ad következtetésként. A módszer előnye az egyszerűség és gyors számíthatóság. A stabilizált KH-módszer csökkenti az eredeti KH-módszer hibáját, az antecedensek és konzekvensek inverz távolságát használja a számításoknál.

A KH módszer egy másik változata a módosított α-vágat alapú interpoláció — **M**odified **α**-**C**ut based **I**nterpolation (MACI), amely kétlépéses interpoláció. Enyhíti a KH-módszer számítási problémáját. A kimeneti és bemeneti univerzumokat olyan térbe transzformálja, ahol a számítási probléma nem fordul elő, majd az eredmény létrejöttekor visszatranszformálja az eredeti térbe. A konvex, normál fuzzy halmazok megjelenítésére vektorokat használunk. A MACI interpoláció egy a gyakorlatban is használható, mivel nem fordul elő a többi interpolációra jellemző számítási hiba, lásd: [17], [18] és [19].

A VKK (**V**as-**K**almar-**K**óczy) módszer a konklúziót a középpont és a szélesség aránya alapján számítja ki. Ennél a módszernél is előfordul bizonyos esetekben a hibás döntési érték. Előnye az egyszerűség és alacsony számítási igény, lásd [20].

A GM (**G**eneral **M**ethodology) tetszőleges formájú fuzzy halmazokra is alkalmazható kétlépéses módszer. A megfigyelések és az antecedensek pontjai alapján számítja ki a döntés referenciapontját. A végső döntés létrehozásához megvizsgálja a távolságot a fuzzy megfigyelés és az interpolált megfigyelés között. [21]

A GLFRI (**G**eometry **B**ased **L**inear **F**uzzy **R**ule **B**ase **I**nterpolation) geometria alapú interpolációs módszer. A szomszédos szabályok konvex kombinációjával egy köztes szabályt származtat, amelyből előállítja a megfigyeléshez tartozó döntést. A geometriai és matematikai módszerek kombinációjával képes arra a módszer, hogy a szabályrendszer jellemzőinek megfelelően állítsa elő a döntést. Ellentétben az α-vágat alapú vagy lineáris közelítést használó módszerekkel, ahol elvész az interpoláció során a szabályrendszer eredeti karakterisztikája. [22]

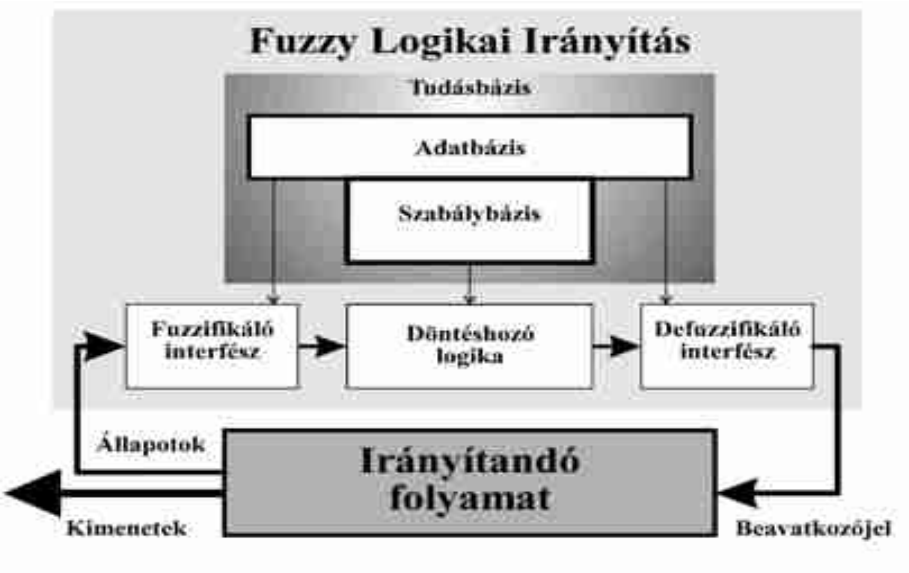
Az általam vizsgált egylépéses módszer a FIVE (**F**uzzy **I**nterpolation on **V**ague **E**nvironment) a 3. fejezetben kerül részletes bemutatásra. A FIVE egy könnyűsúlyú, gyors számítást lehetővé tevő fuzzy interpolációs módszer, amely a fuzzy szabályokat egy úgynevezett homályos térbe helyezi, ahol a távolságuk értelmezhető és így egymással összehasonlíthatók.

2.3. Irányítás fuzzy logikával

A fuzzy logika használata lehetővé teszi az irányítást megvalósító folyamatok leírását szabályokkal, így akár bonyolult folyamatok is kezelhetővé válnak egy, a hagyományos matematikai modellhez képest egyszerűbb leírási módszerrel. Egyszerű szabályozási folyamatok mellett összetett viselkedések is megvalósíthatók a segítségével, mint például egy állat viselkedésének modellezése. Utóbbi esetben azonban már a magas szabályszám miatt szükséges lehet az interpolációs módszerek alkalmazása. A szabályok leírásához szükséges a folyamat ismerete, azaz milyen bemenő értékekre milyen kimenő értékek az elvártak. Ezután lehet elkészíteni a kívánt értékeket előállító szabályokat. A fuzzy logikai szabályozó (FLC - **F**uzzy **L**ogic **C**ontroller) bemenő információi a megfigyelések, amelyek a folyamatot figyelő érzékelőkből és belső változókból származnak. A kimenő értékek a következtetések, amelyek a beavatkozó

szervek működtetéséhez szükségesek. Mind a megfigyelések és a következtetések már valós számok, nem fuzzy számok. Az alfejezet forrásai: [1], [15], [23], [24] és [25].

A fuzzy logikai irányító vázlata a következő ábrán látható (5. ábra):



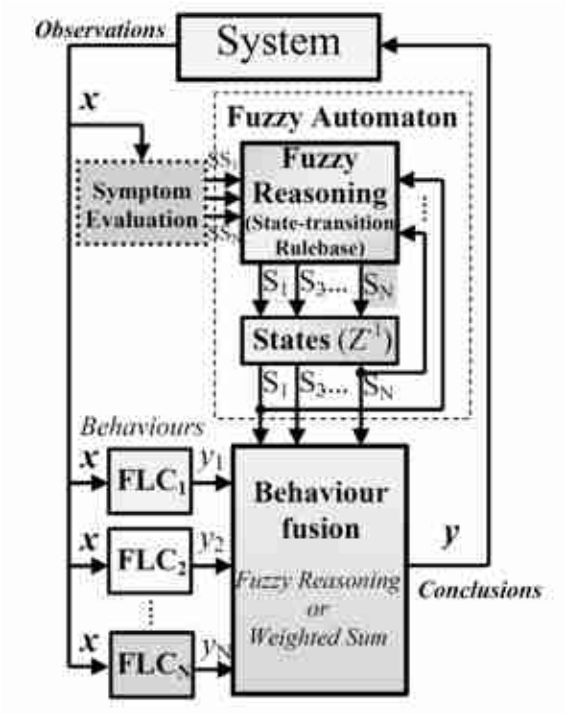
5. ábra A fuzzy logikai irányítás vázlata (FLC [15])

Az *Irányítandó folyamat* érzékelői, állapotjelzői szolgáltatják a megfigyeléseket a *Fuzzy Logikai irányítás* egységnek, amely első lépésként a valós adatokat fuzzifikálja. A Döntéshozó logika a *Tudásbázis* információit felhasználva előállítja a következtetéseket, amelyeket a *Deffuzzifikáló interfész* átalakít az Irányítandó folyamatnak megfelelő formára, ez lesz a *Beavatkozójel*.

A *Tudásbázis* tartalmazza a nyelvi változókat és a belőlük képzett szabályokat, ezeket biztosítja a *Döntéshozó logiká*nak. A megfigyelés és következtetés halmazok nyelvi változóit az *Adatbázis* tartalmazza. Az Adatbázis elemeiből képzett szabályokat a Szabálybázis tárolja. A fuzzy logikához szükséges számításokat a *Döntéshozó logika* végzi. Önmagában a *Fuzzy Logikai Irányítás* (FLC) összetett viselkedések leírására nem elégséges, további kiegészítse szükséges.

Az egyes viselkedéseket külön-külön *FLC blokkok* valósítják meg és a rendszer belső állapotait egy fuzzy állapotgép vezérli.

Az így felépített viselkedés-alapú vezérlő rendszer vázlata az alábbi ábrán látható (6. ábra) ahol a *System* blokk az irányítani kívánt rendszert jelöli, az *X* a rendszerből származó megfigyelések jele. Ezeket használják fel a különféle helyzetekre vonatkozó viselkedést megvalósító *FLC* blokkok. A *Fuzzy Automaton* (Fuzzy Automata) egy fuzzy állapotgép, amely az *X* megfigyelések alapján választ a lehetséges állapotok közül. Mivel fuzzy elven dönt, így az egyes állapotok különböző mértékben érvényesülnek. Ezek akár



6. ábra Viselkedés alapú fuzzy irányítás [67]

egymásnak ellentmondó állapotok is lehetnek, közvetlenül nem használhatók fel a rendszer irányítására. A *Behaviour fusion* blokk szintén fuzzy elven vagy súlyozással állítja elő azt a következtetést, amely a kívánt hatást éri el a rendszer működésében.

Például egy akadályelkerülés esetén egy robot lehetséges mozgási irányai: előre, hátra és fordulás illetve megállás. Lehetséges viselkedései a következők lehetnek:

1. Ha tiszta az út, akkor teljes sebességgel előre.
2. Ha tőle jobbra található akadály, balra tart (minél közelebb van hozzá annál inkább balra).
3. Ha tőle balra található akadály, jobbra tart (minél közelebb van hozzá annál inkább jobbra).

4. Ha az akadály nagyon közel van, akkor megáll (minél közelebb van, annál jobban csökkenti a sebességét, végül megáll).

Abban az esetben, ha tiszta az út a robot teljes sebességgel fog haladni. Amint feltűnik egy akadály, a sebességét csökkenteni fogja a 4. viselkedés miatt és az elhelyezkedéstől függően jobbra vagy balra kerüli meg az akadályt a 2. és 3. szabály szerint, ekkor az első szabály „súlya" csökken, a többieké növekszik. Látható, ha az akadály pontosan a robot előtt van, akkor nem kerüli ki, hanem megáll.

A négy viselkedést egy-egy *FLC* modul valósítja meg. Azt, hogy ezek közül melyek milyen mértékben lesznek érvényesek, arról a *Fuzzy Automaton* dönt. A robot működését tehát fuzzy szabályokkal leírt viselkedések határozzák meg.

2.4. Összegzés

Ebben a fejezetben megvizsgáltam a viselkedés alapú irányítást, összegyűjtöttem a témát érintő fontosabb fuzzy logikai fogalmakat. Továbbá a klasszikus fuzzy logika ritka szabálybázisok esetén fellépő problémáit orvosoló fuzzy interpolációs eljárásokat mutattam be. Ismertettem a viselkedés alapú fuzzy irányítás működését.

3. A FIVE módszer

Jelen fejezetben a [25], [26], [27] források alapján mutatom be a FIVE (**F**uzzy **I**nterpolation on **V**ague **E**nvironment – Fuzzy Interpoláció Homályos Környezetben) módszert, melynek célja egy gyors és egyszerű, közvetlen fuzzy szabály interpolációs módszer létrehozása kifejezetten beágyazott rendszereken való alkalmazásra. Továbbá szó lesz a hozzá kapcsolódó viselkedés leíró nyelvről, amely egyszerűbbé teszi a FIVE használatát és az irányítás megvalósítását beágyazott rendszereken.

A FIVE módszer a fuzzy partíciókat egy úgynevezett homályos térbe helyezi, ahol értelmezhető azok hasonlósága. A hasonlóság mértékét ebben az esetben a szabály döntését meghatározó fuzzy partíciók a homályos környezetben értelmezett skálázott távolsága jelenti. Az egymáshoz közelebb eső szabályok egymáshoz hasonlóak, míg a távoliak kevésbé hasonlóak. Ha egy megfigyelés két szabálypont közé esik és nincs hozzá rendelve döntés, akkor a közeli hasonló szabályok közötti interpolációval számolja ki a közel helyes konklúziót. Amennyiben mégsem mutatja a rendszer az elvárt működést, egy újabb szabály beszúrásával lehetséges a kimeneti érték javítása. A FIVE módszer a Shepard interpolációt használja.

A homályos környezet az elemek hasonlóságán vagy megkülönböztethetetlenségén alapul. Két elem a homályos környezetben akkor ε mértékben megkülönböztethetetlen, ha:

$$\varepsilon \geq \delta_S(x_1, x_2) = \left| \int_{x_2}^{x_1} s(x)dx \right| \tag{11}$$

ahol $\delta_S(x_1, x_2)$ a homályos távolsága az x_1 és x_2 értéknek, $s(x)$ a skála függvény az X halmazon.

A fuzzy tagságfüggvény $\mu_A(x)$ felfogható, mint a hasonlóság mértéke, mint az $\boldsymbol{a}$ szabályponthoz viszonyítva mennyire megkülönböztethetetlen az x megfigyelés. A fuzzy halmazok α-vágata tartalmazza azokat az elemeket, amelyek (1-α) mértékben megkülönböztethetetlenek az $\boldsymbol{a}$-tól:

$$\delta_S(a, b) \leq 1 - \alpha \tag{12}$$

ahol $\delta_S(a,b)$ az ***a*** és ***b*** homályos környezetben értelmezett távolsága. Az $1-\alpha$ az alfa vágat feletti tartomány.

$$\mu_A(x) = 1 - \min\left\{\left|\int_a^x s(x)dx\right|, 1\right\} \tag{13}$$

ahol a $\mu_A(x)$ a homályos környezetben értelmezett hasonlósági fok, az $s(x)$ pedig a homályos környezetben értelmezett skálafüggvény.

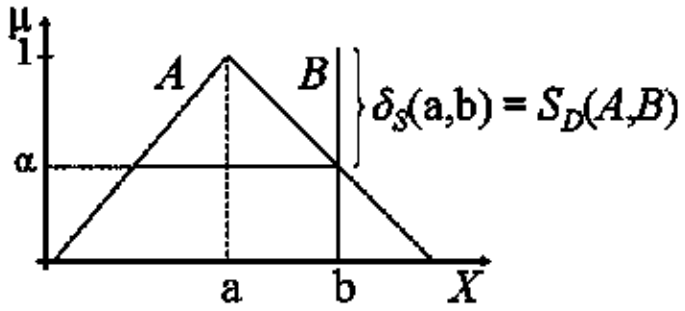
A pontok homályos távolsága azonos a **D**iszkonzisztencia mértékével (S_D) az A és B fuzzy halmaz között, ahol B egy singleton halmaz.

$$S_D = 1 - \sup_{x \in X} \mu_{A \cap B}(x) = \delta_S(a,b), \quad \text{ha } \delta_S(a,b) \epsilon [0,1] \tag{14}$$

ahol $A \cap B$ a minimum t-norma:

$$\mu_{A \cap B}(x) = \min\left[\mu_A(x), \mu_B(x)\right], \quad \forall x \in X \tag{15}$$

Mindezt grafikusan a 7. ábra mutatja be.

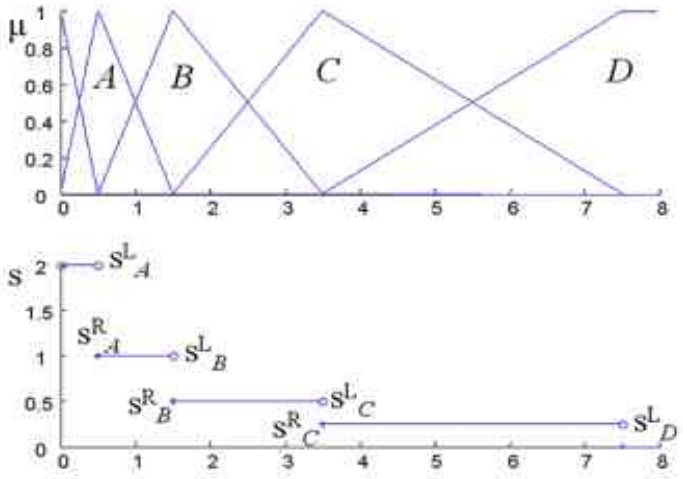


7. ábra Homályos környezetben értelmezett hasonlóság [67]

A homályos környezet létrehozásához olyan skálafüggvényre van szükség, amely képes kezelni az összes létező fuzzy halmaz formáját. Az előző fejezetben felsorolt interpolációs módszerek egy része bizonyos formájú fuzzy halmazok esetén hibás működést eredményez. Ruspini fuzzy partíciók esetén a FIVE módszer homályos környezetben értelmezett skálafüggvénye a következő:

$$s(x) = |\mu'(x)| = \left|\frac{d\mu}{dx}\right| \text{ létezik, ha: } \min\{\mu_i(x), \mu_j(x)\} > 0 \Longrightarrow |\mu'_i(x)| = |\mu'_j(x)| \tag{16}$$

ahol *s(x)* a skálafüggvény, *μ'(x)*, a tagsági függvény deriváltja. A fuzzy halmazok és a skálafüggvény kapcsolatát az 8. ábra mutatja.



8. ábra Ruspini fuzzy partíció és a skálafüggvénye [67]

Az ábra a Ruspini fuzzy partíciók skálafüggvényeit szemlélteti. A Ruspini partíciók jellemzője, hogy bármely pontban a tagsági értékeinek összege 1. Egy-egy partíció egyértelműen azonosítható a halmaz magjával (kernel). Az i-edik háromszög alakú partíció csak az (i-1) és az (i+1) magok között vesz fel 0-tól különböző értéket. A FIVE módszer Ruspini partíciókra való alkalmazása a módszer egy egyszerűsített példája.

A skálafüggvény az antecedensek és a megfigyelés értékét helyezi a homályos környezetbe, ahol megállapítható az egyes halmazok hasonlósága. A döntés (konklúzió, konzekvens) előállításához a FIVE módszer interpolációs módszert alkalmaz. A nyelvi elemek által leírt fuzzy halmazok, ha léteznek, akkor létezik a skálafüggvényük is, amely segítségével mind az antecedensek mind a konzekvensek elhelyezhetők a homályos környezetben, ahol összehasonlíthatóvá válnak. A hasonlóság alapja az egyes homályos környezetbeli pontok (fuzzy szabályok) súlyozott távolsága.

A szabálypontok által leírt felület közelítésére interpolációs vagy regressziós, átlagoló módszerek szolgálhatnak. A FIVE módszer a Shepard interpolációt alkalmazza, amely többdimenziós stabil interpoláció (stabil működést mutat a tartópontok

környezetében is) tipikusan a térben szétszórt mérési pontok esetén alkalmazott módszer. A homályos környezet szabálypontjai megfelelnek egy térben szétszórt ponthalmaznak. A Shepard interpoláció a pontok fordított súlyozott távolságát használja fel. Kétdimenziós változata a következő:

$$S_0(f,x,y) = \begin{cases} f_k, & \text{ha } (x,y) = (x_k, y_k) \text{ néhány } k-\text{ra} \\ \dfrac{\sum_{k=0}^{n} \dfrac{f(x_k,y_k)}{d_k^{\lambda}}}{\sum_{k=0}^{n} \dfrac{1}{d_k^{\lambda}}}, & \text{egyébként} \end{cases} \tag{17}$$

ahol a mérési pontok x_k, y_k $(k\epsilon[0,n])$ az $f\epsilon R^2 \rightarrow R$ leképzésben, a hatványkivető $\lambda>0$ és $d_k = \sqrt{(x-x_k)^2 + (y-y_k)^2}$, Euklideszi távolság.

A Shepard interpoláció FIVE módszerre módosított változata az Euklideszi távolságokat skálázott távolságokkal helyettesíti:

$$\delta_{s,k} = \delta_s(\mathbf{a}_k, \mathbf{x}) = \sqrt{\sum_{i=1}^{m} \left(\int_{a_{k,i}}^{x_i} s_{x_i}(x_i) dx_i \right)^2} \tag{18}$$

ahol s_{x_i} az i-edik skálafüggvény az m dimenziós antecedens univerzumból, x az m dimenziós megfigyelés és a_k képviseli az m dimenziós szabályantecedensek A_k magját. Tehát a:

$$\textbf{HA } x_1 = A_{k,1} \textbf{ ÉS } \ldots \textbf{ ÉS } x_m = A_{k,m} \textbf{ AKKOR } y = B_k \tag{19}$$

szabály esetén a skálázott távolság a következő:

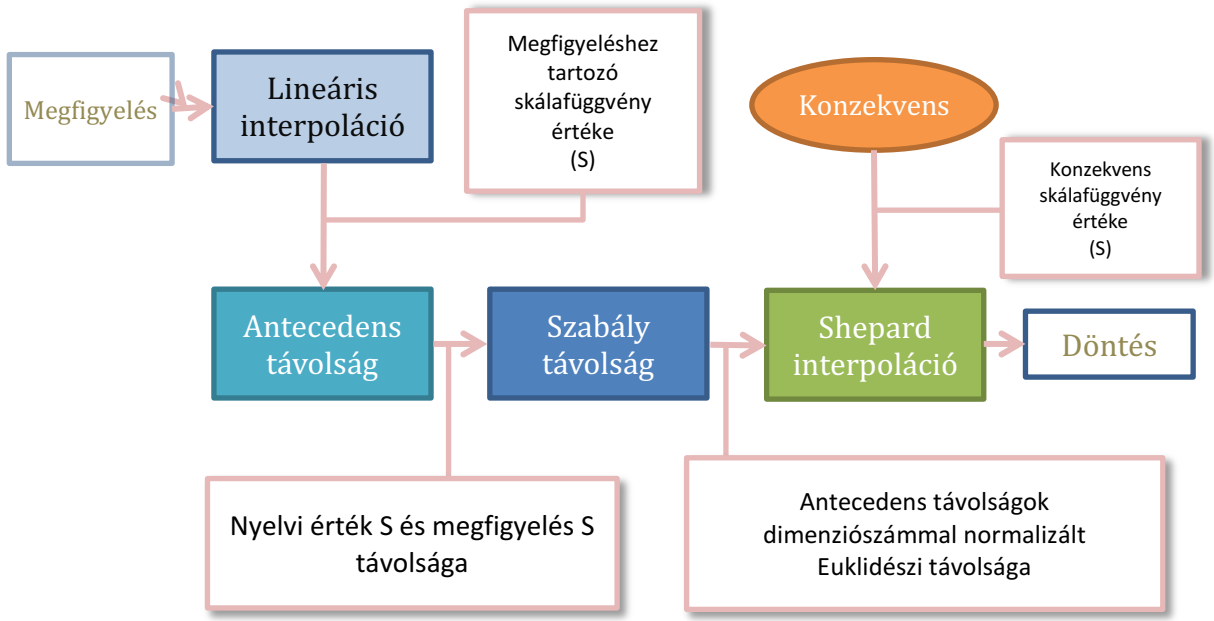
$$\delta_s(b_0, b_k) = \int_{b_0}^{b_k} s_Y(y) dy \tag{20}$$

ahol az s_Y az i-edik skálafüggvény az egydimenziós konzekvens univerzumban, b_k értékek pedig az egydimenziós konzekvens univerzum B_k magjai. Tehát ha az egy dimenziós konzekvens univerzum első eleme b_0 (Y: $b_0 \leq y$, $\forall y \epsilon Y$), az interpoláció az alábbi formában írható fel:

$$\delta_0(b_0, y(x)) = \begin{cases} \delta_s(b_0, b_k), & ha\ \boldsymbol{x} = \boldsymbol{a}_k\ néhány\ k-ra \\ \sum_{k=1}^{r} \left(\frac{\frac{\delta_s(b_0, b_k)}{\delta_{s,k}^{\lambda}}}{\sum_{k=1}^{r} \frac{1}{\delta_{s,k}^{\lambda}}} \right), & egyébként \end{cases} \tag{21}$$

A matematikai hátteret követően a számítási lépések bemutatása következik. A most bemutatásra kerülő számítási eljárást használtam fel a disszertáció téziseinek elkészítése során. Az egyes lépéseket a 9. ábra mutatja.

A *Megfigyelés* lehet a környezet egy érzékelt paramétere (az ágens érzékelője által biztosított adat) vagy egy belső állapot értéke. A *Lineáris interpoláció* eredménye adja a megfigyeléshez S (kumulatív skálafüggvény) értékét az adott antecedens értelmezési tartományán. Ez a korábban említett skálafüggvény primitív függvénye, amely segítségével az egyébként kiszámítani szükséges érték közvetlenül megadható az FBD nyelvvel (lásd: 3.1. fejezet). Ezzel a módosítással számítási sebességnövekedés is keletkezett a FIVE módszerben az eredeti verzióhoz képest. Részletesen lásd: [28]. Ezt a számítást a szabály minden antecedensére el kell végezni. A következő lépés az előzőleg kapott, a megfigyeléshez tartozó S (skálafüggvény) érték felhasználásával számítja ki az antecedens távolságát a megfigyeléstől (*Antecedens távolsága*). Ezt minden antecedensre el kell végezni. Az antecedensek által leírt n-dimenziós térben, ahol a dimenziószám az



9. ábra A FIVE módszer számítási lépései

antecedesek számával egyezik meg az antecedens távolságok által meghatározott vektor hosszát Euklideszi távolság-számítással kerül meghatározásra. Ez adja a szabály távolságot a dimenziószámnak megfelelő normálást követően (*Szabály távolság*). A szabálytávolság kerül felhasználásra a *Shepard interpolációban*, ahol a szabály *Konzekvens* skálafüggvény értékével kerül súlyozásra. A *Shepard interpoláció* kimenete adja a szabálybázis *Döntés*ét, amely egy szám és felhasználható például egy motor sebességének megadására.

A FIVE módszer egy gyorsan számítható, könnyűsúlyú fuzzy interpolációs eljárás. Jól alkalmazható akár beágyazott rendszerekben valós idejű szabályozó megvalósítására. Segítségével olyan összetett problémák megoldhatóak, mint egy állat viselkedésének modellezése és felhasználása etorobotikai rendszerekben [7] [8]. A megfigyelések számával együtt a FIVE módszer számítási igénye is növekszik. Bizonyos gyors beavatkozást igénylő alkalmazásokhoz szükséges az FPGA-val történő megvalósítása. Ezzel gyorsítható a FIVE módszerrel kialakított nagy sebességű nemlineáris szabályozók működése illetve a módszer hangolására használható Q-tanulás ideje is rövidíthető. A Q-tanulási módszerek egyik változata a fuzzy szabály interpolációt alkalmazó FRIQ-learning (Fuzzy Rule Interpolation Based Q-learning), amelynek a FIVE módszert felhasználó verzióját alkalmazó eljárásról bővebben a [29], [30] és [31].

3.1. A viselkedés leíró nyelv

Az FBDL (**F**uzzy **B**ehavior **D**escription **L**anguage – Fuzzy Viselkedésleíró Nyelv) bemutatásának alapjául a [28] és [32] cikk szolgált.

A FIVE alapú fuzzy automata segítségével bonyolult viselkedésmodellek valósíthatók meg egyszerű módon. Az automatának egy saját viselkedésleíró nyelve van, amellyel a viselkedés deklaratív módon fogalmazható meg. A leíró nyelv alapjául a már létező etológiai viselkedést leíró nyelvek szolgáltak, ezek többsége előre meghatározott mintákból épül fel a megfigyelt élőlények reakcióit megfigyelve különböző helyzetekben. A nyelv kialakításánál fontos szempont volt, hogy ne feltételezzen komolyabb programozói előismereteket. A viselkedések modellezéséhez egy szabály alapú tudásreprezentáció használható, amelyben egy eseménysorozat folytonos értékekkel és folytonos állapotokkal leírható. Az automata tehát időben diszkrét, viszont állapotban folytonos modellel számol. Maga a viselkedésmotor egy fuzzy automata, ahol az állapot a

tagsági értékek egy vektora, az állapot átmeneteket egy fuzzy szabálybázis irányítja, a megfigyelések és a következtetések folytonos értékek.

A leírónyelv néhány kötött nyelvi elemből, szabadon választott egyedi elnevezésekből áll. A leírás könnyen olvasható, módosítható. A tartományokból és szabályokból álló szöveges állományt egy Interpreter dolgozza fel, az eredményeket a Viselkedés Motor állítja elő.

Kötött nyelvi elemek:

- **rulebase**: a viselkedésleírás alapvető elemei a szabálybázisok. Ezeknek van antecedens (bemeneti) és konzekvens (következmény) oldala, amely a következőkben kerül majd taglalásra. Mivel a viselkedés kiértékeléshez az FRI módszer család kerül alkalmazásra, ezért a method (mint számítási mód definíciós) rész megadása nem szükséges. Az FBDL bármilyen FRI vagy egyéb módszert alkalmazhat.
- **universe**: A leírás során ennek a megadásával tulajdonképpen az adott dimenzió értelmezési tartományát adjuk meg. Ebben adhatjuk meg, hogy a nyelvi szimbólumokhoz milyen valós értékek tartozzanak. Továbbá, hogy mely értékeknek mennyi a kumulatív skálaértéke [32].
- **description**: opcionális elem, amely főként dokumentációs célokra használható.
- **rule**, **when**, **is**, **and**: szabály leírására használható szavak. Egy szabály a következő alakkban adható meg:
  rule „kimeneti szimbólum” ***when*** „antecedens szimbólum” ***is*** „antecedens érték” ***end***

Az *universe* segítségével egyidejűleg adhatjuk meg mindkét oldalnak az értékeit.

A nyelv felépítése az alábbi példán figyelhető meg, ahol egy egyszerű sebességszabályozó valósul meg a mért távolság függvényében, ami kis távolság esetén alacsony sebességet állít, míg nagy tárgytávolság esetén nagy sebességet:

```
universe "distance"
    "close" 0 0
    "far" 400 1
end
universe "speed"
    "low" 0 0
    "high" 3 1
end
rulebase "speed"
    rule
        "low" when "distance" is "close"
    end
    rule
        "high" when "distance" is "far"
    end
end
```

A *distance* univerzum tárolja a mért távolsághoz, mint bemenő adathoz tartozó értéktartományt például milliméterben [0; 400] és értékkészletet [0; 1]. Az értékkészlet nyelvi elem kumulatív skálafüggvény értéke [32]. A két legfontosabb nyelvi elem került megadásra: *close* és *far (*közeli és távoli az akadály távolsága). A *speed* univerzum a kimenő értékeket képviseli. A kimeneten megjelenő érték a *low*, alacsony sebességnek megfelelő érték: 0, és a magas sebességnek megfelelő *high*, ahol 3 lesz a kimeneten megjelenő érték. A szabálybázis kiszámítása után a kimeneti értéktartomány [0;3] tartományon belül vehet fel értéket. A szabálybázis leírása alacsony sebességet határoz meg, ha a mért távolság közelinek számít illetve magas sebességet, ha a távolságok nagyok.

A leíró nyelv szolgál alapul a következő fejezetben ismertetésre kerülő µFRI API számára.

3.2. Összegzés

Ebben a fejezetben bemutattam a doktori témám alapját képező FIVE módszer matematikai hátterét, a működését és a hozzá kapcsolódó viselkedés leíró nyelv (FBDL) felépítését, szintaktikáját. Az FBDL alapjául szolgált az általam készített implementációk felépítésének.

4. A μFRI könyvtár

Ebben a fejezetben bemutatom azt az általam C nyelven elkészített programkönyvtárat, amelyet a disszertációmban a kísérletek és tesztek elvégzéséhez, ellenőrzéséhez használtam.

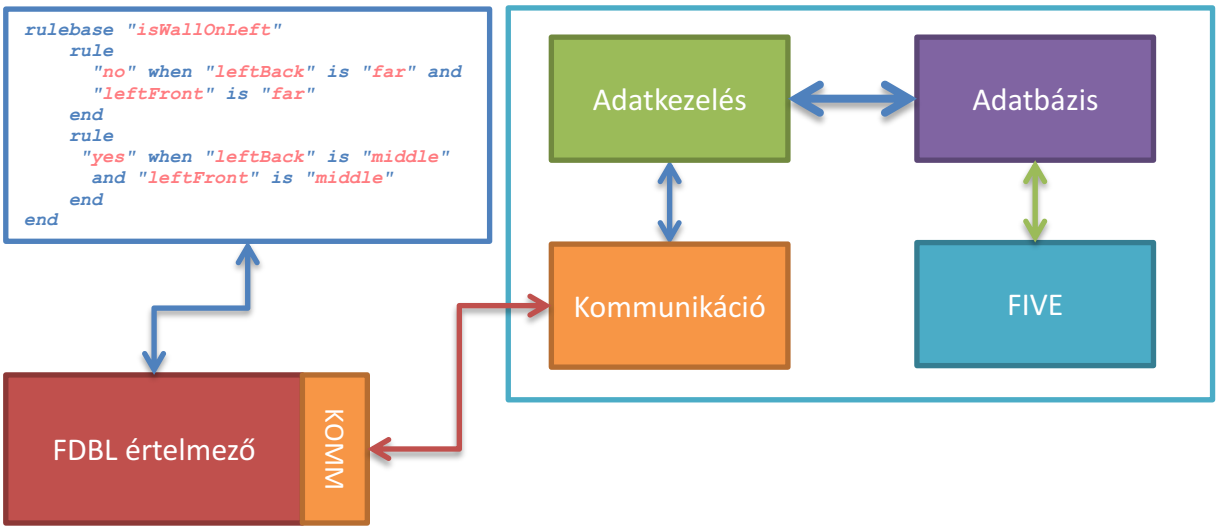
A könyvtár létrehozásának célja: A FIVE módszer már meglévő implementációi mellé elkészítettem egy minimális, C nyelvű megvalósítást, amely beágyazott rendszerek korlátozott erőforrásait figyelembe véve végzi el a FIVE számítási lépéseit és helytakarékosan tárolja paramétereket. Eddig a FIVE módszer implementációja a következő nyelveken létezik: C++, Python, JavaScript, Matlab. A Python és JavaScript változatok tartalmazzák a nyelvi értelmezőt is, a többi változat az tudásbázisát egy állományból tölti fel. A μFRI könyvtár ezzel szemben *egy futás közben, dinamikusan módosítható struktúrát használ a paraméterek tárolására*. Ez lehetővé teszi, hogy futás közben már teljesen új szabálybázissal végezze a számítást. Továbbá a C nyelvű megvalósítással lehetővé válik Java, C#, Python nyelveken is a FIVE használata, külső könyvtárként történő betöltésével. Jelen dolgozat tartalmazza a hardveres, FPGA megvalósítást, amelyet az 6. fejezet mutat be.

A μFRI (mikro FRI, **F**uzzy **R**ule **I**nterpolation, további jelölése: uFRI) könyvtár egy könnyűsúlyú, kis memória igényű megvalósítása a FIVE módszernek. Adatstruktúrái a viselkedés leíró nyelv (FBDL) elemeivel kerültek összhangba. A megvalósítás támogatja a dinamikus és statikus adatbázis feltöltést illetve a szabálybázisok hangolását paraméterek változtatásával vagy a szabálystruktúra módosításával.

A μFRI részei:

- *Adatbázis*: az univerzumok és szabálybázisok paramétereinek tárolására.
- *Adatkezelés*: az adatbázis feltöltése, univerzumok és szabálybázisok hozzáadása, törlése, paramétereinek kezelése. Dinamikus memóriafoglalást nem támogató rendszerek esetén ez a rész kikapcsolható.
- A *FIVE* módszer számítási lépései.

- *Kommunikáció*: az adatbázis módosítását segítő bináris és szöveges parancsértelmező, amely külső modul. Segítségével az FBDL-ből vagy egy hangolási módszerből tetszőleges kommunikációs csatornán feltölthető a tudásbázis az *Adatbázis*ba. Továbbá a tudásbázis exportálható a rendszerből. A *KOMM* és a *Kommunikáció* jelölés ugyan azt a modult jelöli.
- *FBDL értelmező*: a leírónyelv értelmezését végzi és előállítja a megfelelő adatokat a kommunikációs blokknak. Az *FBDL értelmező* nem a könyvtár része.



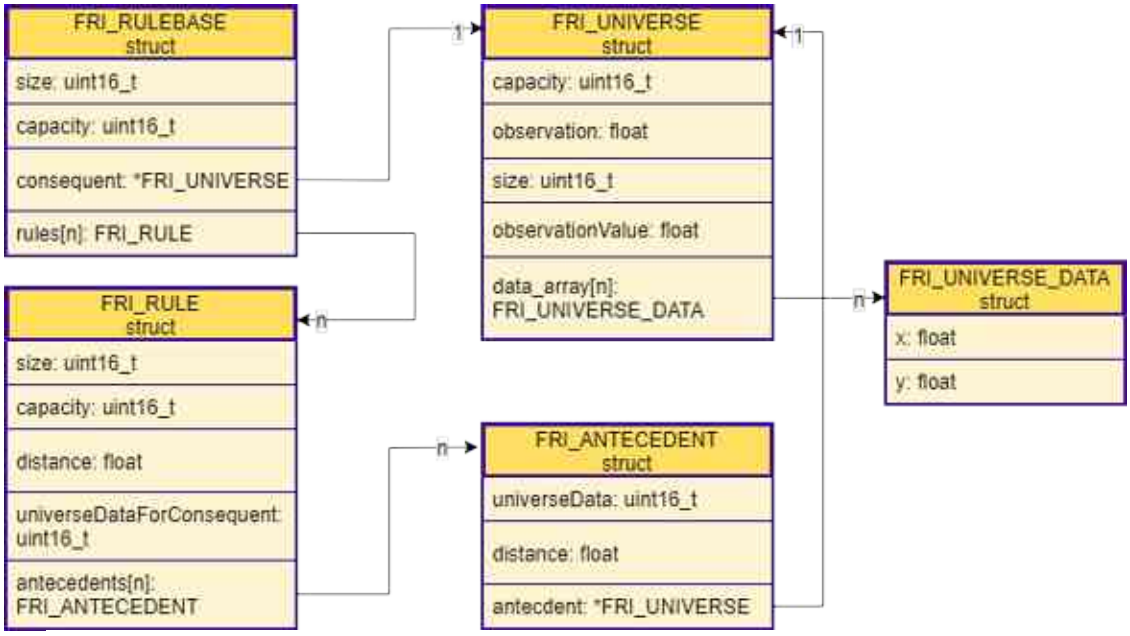
10. ábra A µFRI tömbvázlata

A µFRI könyvtár adatbázisa láncolt struktúrákból áll. A két befoglaló alapelem az *FRI_UNIVERSE* az univerzumok (nyelvi elemekből képzett paraméter lista) tárolására és az *FRI_RULEBASE*, amely a szabályokat fogja össze. Az előbbi két elem egy-egy tömbben került tárolásra. További elemei az *FRI_UNIVERSE* adatstruktúrája, az *FRI_UNIVERSE_DATA*, amely az adott nyelvi elemhez tartozó megfigyelés értékét és a hozzá tartozó tagsági értéket tartalmazza. Ezen felül tartalmazza a megfigyelések, érzékelések értékeit és a hozzá a megadottakból lineáris interpolációval számított tagsági értéket. Az *FRI_RULEBASE* a szabálybázis döntés adathalmazát – *FRI_UNIVERSE*, a **rulebase** utáni szó a leíró nyelvben – és a szabályokat az *FRI_RULE* struktúra tartalmazza.

Az *FRI_RULE*-ban találhatóak a megfigyelések predikátumai, az antecedensek (*FRI_ANTECEDENT*), a szabály konzekvens –a **when** kulcsszó előtti rész a leíró nyelvben - és a szabály távolsága a megfigyelt értektől.

Az *FRI_ANTECEDENT* struktúra a szabály bemenő értékeit tárolja. A leíró nyelv példájában ez a **when** kulcsszó után következő rész. Ez a struktúra tartalmazza mely *FRI_UNIVERSE* biztosítja a bemenő paramétereket és a konzekvensként használt *FRI_UNIVERSE* melyik adattagja lesz a szabályhoz tartozó döntés. A *distance* adattag az antecedens távolságát adja meg a megfigyelt, érzékelt értéktől. Az adatstruktúra felépítését a 11. ábra mutatja.

A µFRI könyvtár a 3. fejezetben már ismertetett számítási lépéseket alkalmazza. Amikor egy *FRI_UNIVERSE* elem *observation* értéke frissül, vagyis új megfigyelés vagy konzekvens érték kerül az adatstruktúrába, a hozzá tartozó µ érték kiszámítása lineáris interpolációval is megtörténik, amely az *observationValue* változóba tárolódik.



11. ábra Az µFRI könyvtár adattároló struktúrájának ábrázolása

Az aktuális szabályhoz tartozó összes antecedens *observationValue* értékét az n-dimenziós Euklideszi-távolság számítás használja fel, amely a szabálytávolságot számítja ki. A dimenziószám megfelel a szabályt alkotó antecedensek számának. A részeredmény normálásra kerül az alábbi (22) egyenlet szerint [28]:

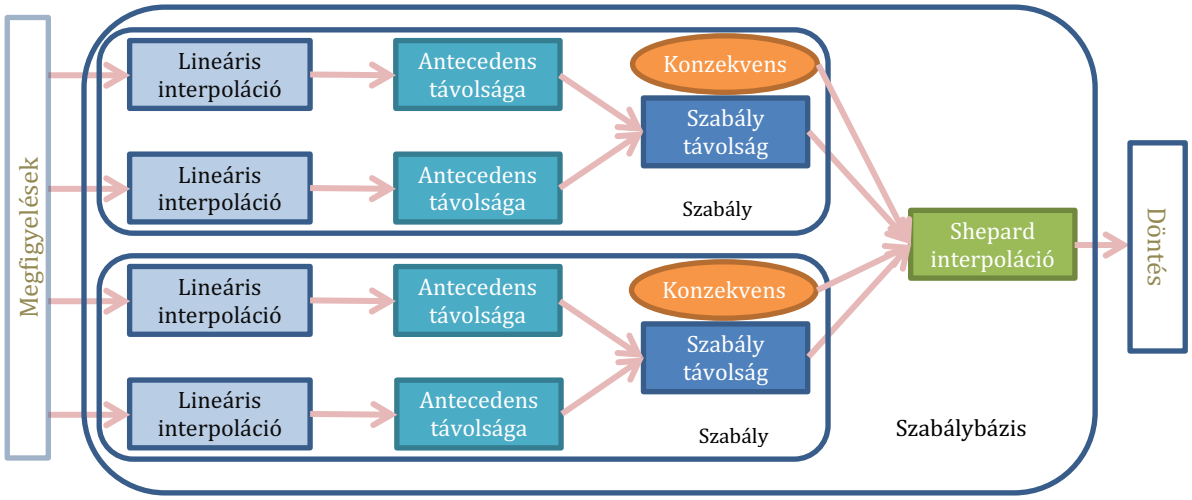
$$szabálytávolság = \frac{antecedensek\ Euklideszi\ távolsága}{\sqrt{antecedensek\ száma}} \tag{22}$$

A szabálytávolság az *FRI_RULE* struktúra *distance* változójába kerül. A gyökvonás eredményét 1 és 256 antecedens darabszám között egy keresőtábla tartalmazza ezzel is gyorsítva a számítást. Ekkora számú antecedensre az eddigi tapasztalatok alapján ritkán

van szükség viszont a keresőtáblával jelentősen javul a számítási idő olyan rendszereken, ahol nincs lebegőpontos számítási egység.

Végül a Shepard interpoláció következik, amely az *FRI_RULEBASE* adatstruktúrán végighaladva az összes *FRI_RULE distance* változóját az adott szabály konzekvens értékével súlyozva kiszámítja a szabálybázis döntését, amelyet a konzekvensként megjelölt *FRI_UNIVERSE* struktúra *observation* változóba ír be. Ennek az előnye akkor jelentkezik, amikor a szabálybázisok egymásba láncoltak és egyik a másik kimenetét használja fel. Mivel egy univerzum csak egy szabálybázis kimenete lehet egyszerre, de többnek bemenete, így használható ez a megoldás.

A két darab két antecedenses szabályt tartalmazó szabálybázis működési sémája a következő, amelyet a 12. ábra szemléltet. Az elsőként a *Megfigyelések* alapján a *Lineáris interpolációt* számolja minden *Universe*-re. Ezt követően az *Antecedens* távolságokat minden antecedensre. A következő lépés a *Szabály távolság* számítása szabályonként. Végül a *Shepard interpoláció* a szabály távolságokat felhasználva a *Konzekvens*sel súlyozva számítja a kimeneti értéket, a *Döntést*.



12. ábra A µFRI könyvtár számítási lépései

A µFRI könyvtár memóriaigénye minimális a már meglévő megvalósításokhoz képest. A byte-ban megadott adatméret az alábbi összefüggéssel számítható:

$$T = 8N_{UE} + 12N_U + 12N_A + 14N_R + 12N_{RB} \tag{23}$$

ahol az N_{UE} az összes univerzum elem száma, N_U az összes univerzumok száma, N_A az összes antecedens száma a szabályokban, N_R a szabályok száma, N_{RB} a szabálybázisok száma,

T pedig a teljes szükséges tárterület mérete byte-ban. Ez alapján a legkisebb elfoglalt memóriaterület 58 byte, amikor $N_{UE}=N_U=N_A=N_R=N_{RB}=1$.

A többi ismert megvalósítással történő összevetést az 1. táblázat tartalmazza. Az egyes megvalósítások környezeti jellemzőit is figyelembe vettem az összehasonlítás során. Például JavaScript és Python esetén szükséges a futtató hardveren működjön a nyelv interpretere, viszont ez nem probléma, ha egyéb algoritmusok is futnak a FIVE mellett. Az objektum orientált vagy objektumokat használó nyelvek esetén szükséges a dinamikus memóriafoglalás, amelyet nem minden beágyazott rendszer támogat, de sok igen. Az objektumok, amelyek tárolják a FIVE tudásbázisát a saját működésükhöz szükséges további információkat tartalmaznak, amelyek tovább növelhetik a memóriaigényt. Ez a C nyelvű megvalósításban kevesebb további adatot jelent.

1. táblázat A µFRI memóriaigényének összevetése más megvalósításokkal

Implementáció	µFRI	C++	Python	JavaScript	MATLAB
Memóriaigény µFRI -hez képest	egység	struktúrákban tárolt tudásbázis	nagyobb, objektumokban tárolt tudásbázis, muszáj a dinamikus memóriafoglalást támogassa a rendszer	sokkal nagyobb, a Python adattárolási sajátossága és a szükséges interpreter miatt	sokkal nagyobb, interpreter szükséges, objektumokban tárolt tudásbázis + nyelvi értelmező	nem ismert

Mivel µFRI lehetővé teszi az egyes szabálybázisok egymástól független feldolgozását, a teljes viselkedésmodell párhuzamosan is feldolgozható, csökkentve ezzel a számítási időt. A legtöbb számítási időt igénylő rész az antecedensek távolságszámítása és az ezt megelőző lineáris interpoláció, mivel ezeket kell a legnagyobb számban elvégezni minden új megfigyelés beérkezésekor. Az eredmények tárolásra kerülnek így eltérő mintavételezései sebesség esetén lehet csökkenteni a processzor terheltségét.

A µFRI könyvtár párhuzamosításával részletesen A FIVE módszer párhuzamos számítási sebességének gyorsítása fejezet foglalkozik.

4.1. Kommunikáció

A szabálykészlet valós idejű paraméterezése és módosítása céljából a következő kommunikációs keretek használatát javasolom. A keretek egyszerű szöveges vagy bináris formában továbbíthatók például UART-on keresztül a mikrovezérlőhöz, amelyen a fuzzy viselkedésmodell fut. További erőforrást csökkent, ha az adatok bináris adatsorként kerülnek továbbításra. A továbbított adatsor az aktuálisan használt FBDL-ből generált, az aktuális szabálybázis leírása alapján generálható. Fontos a szabályok és univerzumok sorrendje, mert az alapján kerülnek az paraméterek sorszámozásra. Az adatok frissítése alatt a számításokat szüneteltetni kell, elkerülve a hibás működést.

4.1.1. Szöveges üzenetsor

A szöveges változat nagyobb erőforrással rendelkező beágyazott rendszerek esetén használható. Előnye, hogy ember által olvasható, könnyen módosítható. A kommunikációhoz az alábbi struktúrákat javaslom. Minden keretre a beágyazott rendszer OK vagy ERROR üzenettel válaszol annak megfelelően, hogy a kért változtatásokat végre tudja hajtani vagy sem.

A kezdő keret (I initialization) hordozza az információt, hogy mennyi darab univerzumot és mennyi szabálybázist tartalmaz a viselkedés leírás (13. ábra). Ez alapján tudja a rendszer, hogy mennyi memóriát kell foglalnia, vagy a struktúra változtatása nélkül végre tudja hajtani a változtatásokat. Abban az esetben, ha a beágyazott rendszer támogatja a dinamikus memóriafoglalást vagy előre le volt foglalva a megfelelő terület OK üzenettel válaszol egyéb esetben ERROR a válasz. Ez az első keret, amelyet a beállításhoz küldeni kell.

I	univerzumok száma	:	szabálybázisok száma
Init jel	[0-255]	elválasztó	[0-255]

13. ábra Kezdeti állapotot beállító keret struktúrájának ábrázolása

Univerzum elemek átvitelére az 14. ábra mintájára létrehozott keret alkalmas, amely az I típusú keretet követi. Az U betű jelöli az univerzumot FBDL nyelvnek megfelelően, amelyet a kettőspontok által határolt elempárok száma követ. Az egyes elempárokat a függőleges vonal karakter választja el egymástól. A szöveges átvitel esetén mindig *float* típusú értékek átvitele történik. Válaszul OK vagy ERROR érkezik rá.

U	szorszám	:	elemek száma	:	x érték	|	y érték	:	...	|	...	:
universe jel	[0-255]	elválasztó	[0-255]	elválasztó	universe x érték	elválasztó	universe y érték	elv.	elemek számának megfelelően folytatódik			
					1. elem				2. elem			

14. ábra Univerzum elemek átviteli keret struktúrájának ábrázolása

Az U típusú kereteket az R típusú rulebase keret követi, amely tartalmazza melyik sorszámú szabálybázisban mennyi szabály lesz, illetve melyik univerzum tartozik a szabálybázis konzekvenséhez. Vagyis melyik univerzum lesz a szabálybázis kimenete, lásd: 15. ábra. Ha létrehozható a szabálybázis OK, ha nem hozható létre vagy nem paraméterezhető a meglévő szabálybázis ERROR a válasz.

R	sorszám	:	szabályok száma	:	universe id
Rulebase jel	[0-255]	elv.	[0-255]	elv.	Szabály konzekvenshez

15. ábra Rulebase keret szerkezetének ábrázolása

Az R típusú kereteket az E típusú keret követi, amely a szabályokat, vagyis a szabálybázis elemeit tartalmazza (16. ábra). A keret tartalmazza az FBDL-ben leírt szabály sorszámát, az antecedensek számát és ismétlődően függőleges vonallal elválasztva az antecedensekhez tartozó univerzum sorszámát és az univerzum antecedensehez tartozó elemének sorszámát. Ezek az antecedensek számának megfelelően ismétlődnek. A keretre OK vagy ERROR üzenet érkezik válaszul.

E	sorszám	:	antecedensek száma	:	universe id	|	universe data element	:	...
Rule jel	[0-255]	elv.	[0-255]	elv.	Antecedensben szereplő univerzum azonosítója	elv.	Antecedensben szereplő Univerzum elemazonosítója	antecedensek számának megfelelően folytatódik	

16. ábra Szabály keret szerkezetének ábrázolása

ERROR üzenet a válasz, amikor:

- a rendszer nem támogatja a további memória foglalását,
- helytelen sorrendben érkeznek a keretek

A keretek helyes sorrendje: I-U-R-E, ahol az U, R, E keretekből egymás után több is érkezhet.

4.1.2. Bináris üzenetsor

A bináris átvitel előnye, hogy olyan beágyazott rendszereken is használható, amelyek nem rendelkeznek elegendő erőforrással szöveges adatok feldolgozására. Kedvezőbb a bináris módot választani az FPGA megvalósítás esetén is, így lehetséges

akár közvetlenül az IP-t konfigurálni az üzenetsorral. A keretek és a kommunikáció hasonlóak a szöveges átvitelhez. Azonban adatok 16 bites előjeles számokra korlátozódnak. Az OK és ERROR üzenetek az előző pontban leírtak szerint érkeznek. A bináris keretek esetén egy-egy cella egy byte-nak felel meg, amely előjel nélküli 8 bites egész szám.

A bináris keret a kezdeti értékek beállítására, I keret (17. ábra):

255	1	univerzumok száma	szabálybázisok száma
Keretfej	Init jel	[0-255]	[0-255]

17. ábra Kezdeti értékek beállítása, I keret

Az univerzumok átvitelére alkalmas, U keret (18. ábra), ahol a keret adatrésze az elemek számának megfelelően ismétlődik:

255	2	szorszám	elemek száma	felső byte	alsó byte	felső byte	alsó byte	...	...
Keretfej	universe jel	[0-255]	[0-255]	universe x érték 16 bit egész		universe y érték		elemek számának megfelelően folytatódik	
				1. elem				2. elem	

18. ábra Univerzumok átvitele, U keret felépítésének ábrázolása

A szabálybázisok átvitelére a következő keret használható (19. ábra):

255	3	sorszám	szabályok száma	universe id
Keretfej	Rulebase jel	[0-255]	[0-255]	Szabály konzekvenshez

19. ábra Szabálybázis keret, R típusú keret felépítésének ábrázolása

A szabálybázis antecedens elemeinek átvitelére az alábbi keret alkalmas (20. ábra), az universe id és universe data az antecedensek számának megfelelően ismétlődik:

255	4	sorszám	antecedensek száma	universe id	universe data element	...
Keretfej	Rule jel	[0-255]	[0-255]	Antecedensben szereplő univerzum azonosítója	Antecedensben szereplő Univerzum elemazonosítója	antecedensek számának megfelelően folytatódik

20. ábra Szabályok és antecedensek átvitele, E keret felépítésének ábrázolása

Az OK (21. ábra) és ERROR (22. ábra) a válaszkeretek bináris átvitel esetén:

255	10
Keretfej	OK

21. ábra OK keret felépítése felépítésének ábrázolása

255	20
Keretfej	ERROR

22. ábra ERROR keret felépítése felépítésének ábrázolása

4.2. Korlátozások a használat során

A dinamikus szabálymódosítás olyan rendszerek esetén, amelyekben nem támogatott a dinamikus memóriafoglalás csak korlátozott keretek között érhető el. Mivel a lefoglalt memóriaterület nem növelhető a kézi megadáskor kell számolni a lehetséges növekedéssel további üres szabályok, szabálybázisok lefoglalásával.

4.3. Felhasználási minta

A µFRI könyvtár használatára mutat példát ez a szakasz. A mintapéldában látható, hogyan lehet az FBDL-ben készített leírást a µFRI számára átalakítani.

A felhasznált szabálybázis már korábban bemutatásra került a *3.1. A viselkedés leíró nyelv* című fejezetben. A következő FBDL leírásban szereplő „#” szimbólum nem az FBDL része, csak az átírás megértését szolgálja. Az átírás során az egyes elemek 0-tól kezdődően sorszámozásra kerülnek, ezek a számok kerülnek a µFRI könyvtár kezdeti érték beállító szakaszába.

```
universe "distance" # 0 sorszámú universe
    "close" 0 0     # 0 sorszámú universe elem
    "far" 400 1     # 1 sorszámú universe elem
end
universe "speed"    # 1 sorszámú universe
    "low" 0 0       # 0 sorszámú universe elem
    "high" 3 1      # 1 sorszámú universe elem
end
rulebase "speed"    # 0 sorszámú rulebase
    rule            # 0 sorszámú rule
        "low" when "distance" is "close" # 0 sorszámú antecedens
    end
    rule            # 1 sorszámú rule
        "high" when "distance" is "far"  # 0 sorszámú antecedens
    end
end
```

A leírás 2 darab universe elemet és 1 darab rulebase-t tartalmaz. A rulebase 2 darab rule-t és egy-egy rule 1 darab antecedenst. A rulebase az 1 sorszámú universe-t használja kimenetként. A rule-ok a 0 sorszámú universe elemeit használják bemenetként.

A fentieknek megfelelően elsőként az universe elemek számát és a rulebase darabszámát kell megadni:

```
FRI_init(2, 1); //első paraméter az universe darabszám, második a rulebase darabszám.
FRI_initUniverseById(0,2); //első paraméter az universe sorszáma, második az universe által tartalmazott nyelvi elemek darabszáma.
```

Ezt követően az egyes universe-ek által tartalmazott universe elemek számát (nyelvi elemek):

Az universe-hez az egyes nyelvi elemek hozzáadását a következő részlet tartalmazza. Abban a sorrendben kell az elemeket felsorolni, ahogy az FBDL-ben is található.

A rulebase elem hozzáadása az alábbi kódrészlettel történik. Itt már hozzáadásra kerül a rule és a rule antecedense is:

```
FRI_initRuleBaseById(0, 1, 2); // rulebase létrehozása; első paraméter a szabálybázis sorszáma, második a kimeneti universe sorszáma, harmadik a rule elemek száma.
FRI_addRuleToRulebase(0, 1); // rule hozzáadása az előzőleg létrehozott rulebase-hez; első paraméter a kimeneti universe nyelvi elemének sorszámát, második paramétere az antecedensek darabszámát adja meg.
FRI_addAntecedentToRule(0, 0); // antecedens hozzáadása az előzőleg létrehozott rule elemhez, első paraméter az universe sorszámát, második paraméter az universe nyelvi elemének sorszámát adja meg.
```

Az összes adat felvitelét követően lehetőség van a megfigyelések frissítésére és az eredmények lekérdezésére. Ezt szemlélteti a következő kódrészlet:

```
float observation = 4; //minta megfigyelés értéke
float consequent = 0; //döntés értékének tárolása
FRI_setObservationForUniverseById(0, observation); //0. universe megfigyelésének beállítása
FRI_calculateAllRuleBases(); // minden szabálybázis kiszámítása
consequent = FRI_getObservationById(1); //döntés értékének lekérdezése
```

A mintaként megadott szabálybázishoz tartozó kód teljes egészében az alábbi:

```
float distance=0;
float speed=0;

FRI_init(2, 1);
FRI_initUniverseById(0,2);
FRI_addUniverseElement(0, 0);
FRI_addUniverseElement(400, 1);
FRI_initUniverseById(1,2);
FRI_addUniverseElement(0, 0);
FRI_addUniverseElement(3, 1);
FRI_initRuleBaseById(0, 1, 2);
FRI_addRuleToRulebase(0, 1);
FRI_addAntecedentToRule(0, 0);
FRI_addRuleToRulebase(1, 1);
FRI_addAntecedentToRule(0, 1);

observation=4; //cm

FRI_setObservationForUniverseById(0, observation);

FRI_calculateAllRuleBases();

consequent = FRI_getObservationById(1);
```

4.4. Tézis

A µFRI könyvtár alkalmas a FIVE FRI mikrovezérlőkön való implementációjára. Lehetőséget biztosít a tudásbázis rendszerleállás nélküli dinamikus paraméter megváltoztatására, beágyazott rendszerbeli adaptív alkalmazások kialakítására.

4.5. Új eredmények

A µFRI könyvtár lehetővé teszi a FIVE alapú viselkedés leírás használatát beágyazott rendszerekben úgy, hogy a szabálybázisok futás közben dinamikusan megváltoztathatók. Ezzel lehetővé válik alacsony energiaigényű, komplex viselkedést megvalósító rendszerek létrehozása a FIVE segítségével.

A µFRI könyvtár adatstruktúrája az FBDL nyelvet veszi alapul. Egy fordító segítségével az [S8] cikkben bemutatott üzenetmintával a µFRI-t futtató rendszer viselkedése megváltoztatható vagy valós időben hangolható.

4.6. Gyakorlati felhasználás

A µFRI könyvtár felhasználásával az alábbi eszközök vezérlése került megvalósításra:

- Inga elvű gömbrobot stabilizálása: [33],
- Fordított inga elvű robot stabilizálása: [34],
- Micromouse robot akadály felismerése: [S8] és [S10],
- Holonomikus robot (KR1) játék a labdával viselkedés: [S5] és [S7].

4.7. Összegzés

Ebben a fejezetben bemutattam az általam készített C nyelven készített minimális memóriaigényű FIVE változatot, amelyet felhasználtam a kutatások során elvégzett tesztek elvégzéshez.

A részletes működés és használata az [S8] és [S10] cikkekben került ismertetésre.

Az fejezethez tartozó tézist a 4.3. míg az elért új eredményeket a 4.5. fejezet tartalmazza.

5. A FIVE módszer párhuzamos számítási sebességének gyorsítása

Ebben a fejezetben a FIVE módszer vizsgálatát mutatom be többprocesszoros beágyazott rendszereken, kifejezetten a cache használat figyelembe vételével és a µFRI könyvtár felhasználásával.

A modern rendszer a chip-en (SoC – System on Chip) eszközök egyre inkább többprocesszoros rendszerek. Ezek teljesítményfelvételét és fizikai méretét figyelembe véve alkalmazhatók autonóm mobil robotok központi vezérlő hardvereként. A számítási teljesítményük jobb kihasználása érdekében szükséges megvizsgálni a viselkedés leírás alapjául szolgáló FIVE interpolációs módszer működését ezeken a rendszereken. Annak a képletnek a meghatározása, amely megadja, milyen arányban érdemes szétosztani a szabálybázisokat az egymással versengő processzorok között. Ennek oka, hogy a többprocesszoros rendszerekben esetén az egymással a gyorsító tárért egymással versengő processzorok —bizonyos adatmennyiségek esetén—, számítási hatékonysága alacsonyabb lehet, mint az egyprocesszoros rendszer számítási hatékonysága. Ez a jelenség megfelelő méretű bemenő adatok és a processzorok közötti megfelelő terhelés elosztással elkerülhető. A rendszer számítási gyorsulása ellenőrizhető Amdahl törvénye segítségével a számítási idők alapján.

5.1. Célkitűzés

A FIVE módszernek a µFRI könyvtár egy alacsony számítási-, és memóriaigényű megvalósítása. Az eddigi gyakorlati alkalmazások és Q-tanulás eredményei alapján a szabálybázisok száma egy-egy problémára 2-3 szabálybázistól egészen 1000-2000 darabra tehető. [31], [35] és [36].

A vizsgálat célja, hogy a jelenleg használatos, népszerű többmagos beágyazott rendszerek esetén hogyan lehet a számítási hatékonyságon javítani esetleg meghatározni egy optimális értéket arra nézve, hány processzormag vegyen részt a számításban. Ehhez a könyvtár memóriaigénye alapján kerül vizsgálatra számítás hatékonysága a rendszer cache memória méretéhez viszonyítva. Illetve az egyprocesszoros végrehajtáshoz képest elért gyorsulás mértéke Amdahl összefüggése alapján meghatározva a számítási időkből.

Mivel a µFRI számítási lépései önmagukban viszonylag rövid időt vesznek igénybe, az algoritmus nem kerül további felbontásra. Egy szabályhalmaz egy egységként kerül kezelésre a tesztek során.

A számítási hatásfok növelésének eredménye, hogy főként a szimulációk és a tanítási eljárások időigénye csökkenthető nem csak kicsi, hanem nagy szabálymennyiség esetén is.

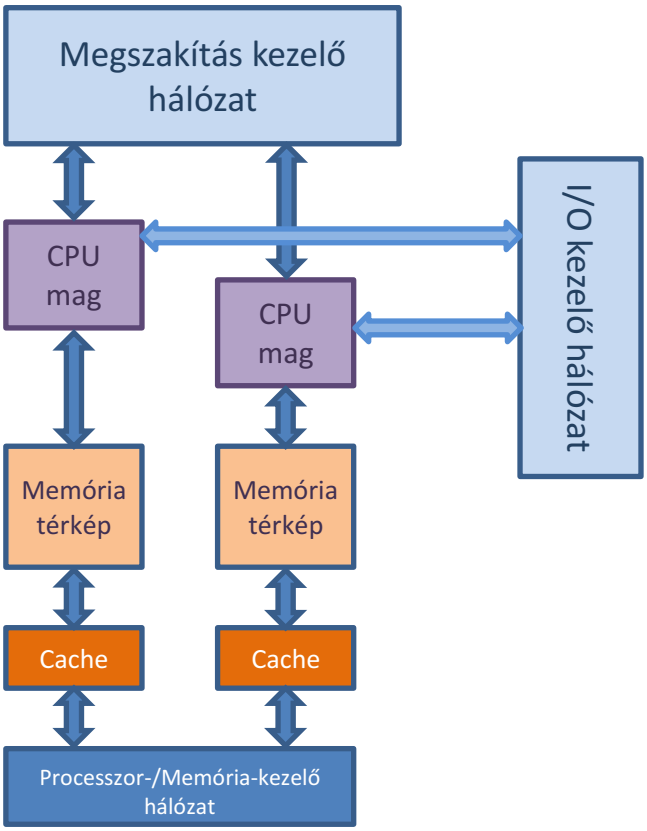
5.2. A többprocesszoros beágyazott környezet

A számítási kapacitás növelését az órajel növelése mellett a processzorok számának növelésével lehet megoldani. A több processzor együttes munkájával javítható a válaszidő, elérhető a valós párhuzamos végrehajtás. A számítási problémát, feladatot több kisebb részre bontva megfelelő tervezés mellett, a különböző számítási egységek az egyprocesszoros rendszerhez képest akár rövidebb idő alatt is elvégezhetik a feladatot, lásd: [37].

A beágyazott rendszerek esetén is alkalmazzák a több CPU-t használó megoldásokat, egy vagy több nagy számítási teljesítményű CPU integrálásával egy tokba vagy nagy energiaigényű és alacsony energia igényű CPU-k vegyes felhasználásával. Így elérhető az is, hogy bizonyos feladatokat, a rendszer alapvető működését képes ellátni az alacsony energiaigényű mag, míg a bonyolult számításokat a nagyobb energiaigényű, nagyobb számítási teljesítményű processzormag fogja elvégezni rövid idő alatt a válaszidőt csökkentve. Ezzel a megoldással megfelelő számítási teljesítményt lehet elérni alacsony energiaigény mellett, az akkumulátoros üzemidő növelésével, lásd: [38], [39] és [40].

A felhasznált számítási egységek közötti kommunikációt kétfajta megoldással lehet megvalósítani: a processzorok laza illetve szoros kapcsolásával. Szoros kapcsolás esetén a processzormagok osztoznak az órajelen, a buszvezérlő logikán, a teljes memórián és az I/O rendszeren. A processzormagok közötti adatcsere megosztott memórián keresztül valósul meg. Tehát a rendszer memóriát a processzormagok egymással versengve érik el. Alapvetően magas számítási igény, valós idejű feldolgozás esetén alkalmazzák a szorosan csatolt többprocesszoros rendszereket. Általános cache memóriával kiegészített rendszer felépítése az alábbi ábrán látható (23. ábra).

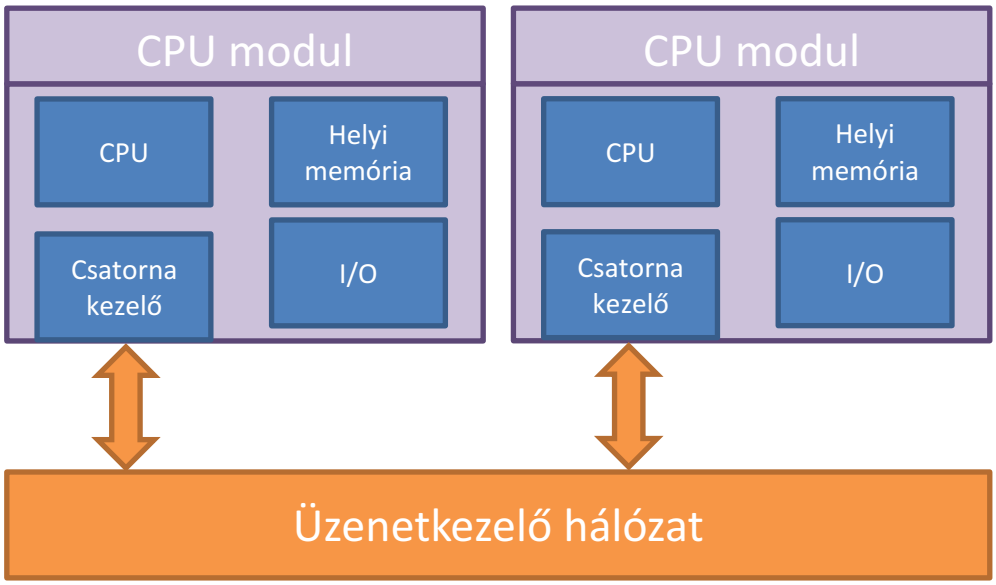
A szorosan csatolt többprocesszoros rendszer főbb részei a 23. ábra alapján kerülnek bemutatásra. A *CPU mag* az általános célú processzor, amely kapcsolódik a *Megszakítás kezelő hálózat*hoz, *I/O kezelő hálózat*hoz amellyel, a külvilággal tartja a kapcsolatot és a *Memória térkép*hez. A *Memória térkép* látja el a rendszermemória címzési feladatot és a helyi, nem a térképhez tartozó memória kezelését. A *Cache* memória az adatok elérésnek gyorsításáért felel azáltal, hogy a rendszermemóriából a gyorsabb és a CPU számára gyorsabban elérhető memóriában tárolja az éppen szükséges adatokat. A *Processzor-/Memória-kezelő hálózat* felelős a megfelelő órajelek biztosításáért és a rendszermemória kezeléséért. [41]



23. ábra Szorosan csatolt többprocesszoros rendszer, egyszerűsített ábra

Szorosan csatolt többprocesszoros rendszerek beágyazott rendszerekben a többmagos ARM processzorok (A53 SoC, Xilinx Zynq SoC), mikrovezérlők, amelyeket a vizsgálatokhoz használtam. A vizsgálatok során a cache memóriáért és rendszermemóriáért folytatott versengéses helyzetet vizsgáltam a FIVE módszer számítása közben.

Lazán csatolt többprocesszoros rendszerek esetén a processzormagok egy belső hálózaton keresztül kommunikálnak egymással. Minden processzor tartalmaz saját



24. ábra Lazán csatolt többprocesszoros hálózat

memóriát és egyéb, a működéshez szükséges perifériát. A belső hálózaton a szorosan csatolt többprocesszoros rendszerhez képest sokkal kisebb adatmennyiséggel dolgozik.

A lazán csatolt többprocesszoros rendszer leginkább egy elosztott rendszernek tekinthető. Felépítését a következő ábra szemlélteti (24. ábra).

Az előző ábrán a *CPU modul* egy önálló számítási egység, saját *I/O* felülettel, *CPU*-val és saját memóriával. Képes önállóan egy program futtatására. A többi *CPU modul*lal az *Üzenetkezelő hálózat*on keresztül tartja a kapcsolatot. A *Üzenetkezelő hálózat* segít egy esetleges felügyelő rendszerrel történő kapcsolattartásra vagy a rendszermemória elérésére. A lazán csatolt többprocesszoros rendszer elosztott memóriakezelést használ. A CPU modulok versengése a memóriáért nem jelentős, az esetleges rendszermemória irányába játszódik le, amelyet az Üzenetkezelő hálózat felügyel. [41]

Egyik ilyen beágyazott processzortömb az Adapteva Parallella nevű kártyájára integrált Epiphany processzortömb. Segítségével azt vizsgáltam, hogy az operációs rendszer milyen mértékben lassítja egy számítás végrehajtását a rendszer fenntartásáért futtatandó folyamatok kezelésével. A vizsgálat kiterjedt és a FIVE módszer által használt Shepard interpolációs eljárás párhuzamosíthatóságának visgálatára is. Bővebben: [S13].

5.3. A párhuzamos működés hatásfokának vizsgálata

Egy számítási feladat párhuzamos feldolgozása, több processzorra történő szétosztása nem feltétlenül jelenti a feladat gyorsabb vagy jobba hatásfokú megoldását.

Az, hogy egyáltalán érdemes-e az adott számítási feladatot egynél több processzorral megoldani, függ a feldolgozandó adat mennyiségétől, a számítási időtől és a rendszer sajátosságaitól is. Könnyen előfordulhat olyan melegedési probléma vagy, versenyhelyzet valamilyen erőforrásért, aminek következtében a párhuzamosított rendszer rosszabbul teljesít, mint a nem párhuzamos működésű megoldás, lásd [37] és [42].

A párhuzamosítás hatékonyságának vizsgálatára használható Amdahl összefüggése, amely a következő szakaszban kerül részletes leírásra.

Az általam vizsgált jelenség a többprocesszoros rendszer közös cache memóriáért folytatott versengése. Ez a FIVE módszer esetén fontos, mert a μFRI könyvtár alacsony memóriaigénye miatt előfordulhat, hogy tovább tart az adatok mozgatása a kis mennyiség miatt, mint azok kiszámítása.

5.3.1. Amdahl törvénye

Az 1967-ben G. M. Amdahl által leírt összefüggés modellezi egy algoritmus, párhuzamosítással elérhető számítási sebesség növekedését (azaz a számítási idő csökkenése), a felhasznált processzorok számának függvényében [37]. Ez egy elméleti gyorsulás érték, amely egy felső korlátot ad arra nézve, hogy mekkora sebességnövekedés érhető el, ha egy algoritmus egynél több processzoron fut. Amdahl törvényének alkalmazásához szükséges a programkód párhuzamos futásban töltött idejének és soros futásban töltött idejének ismerete. Utóbbiba tartozik az az idő, amelyet a rendszer egyéb feladatok ellátásával tölt, mint a memória kezelés ideje vagy egy hardver elérésének ideje esetleg a folyamatok ütemezésének ideje az operációs rendszer működésére szánt idő. Ezek azok a rejtett idők, amelyek akár jelentősen növelhetik is egy programkód végrehajtási idejét.

Amdahl törvénye az egyprocesszoros rendszerek korlátaira mutat rá, de kiterjeszthető többprocesszoros rendszerre is.

Az Amdahl formulához a párhuzamosított algoritmus számítási idejét két részre kell osztani, párhuzamos futásban eltöltött időre és soros futásban eltöltött időre. Az algoritmus soros futásban eltöltött ideje azon időszakaszokból adódik össze, amelyek tartalmazzák a feladatkiosztás idejét és az egyéb várakozási időket, például valamely hardverelemre várakozás vagy késleltetési időket, stb.

$$S(P) = \frac{1}{1 - t_P + \frac{t_P}{P}}, \quad t_p = \alpha = \frac{T_p}{T} \tag{24}$$

ahol, P a magok számát jelenti, A T_P a párhuzamos számításban eltöltött időt, T pedig a teljes számítás idejét jelöli. A t_P a párhuzamos számításban töltött idő hányadát jelöli. Az $(1-t_P)$ a soros szakaszban eltöltött időt jelöli. S a számítási sebességnövekedés száma.

Az Amdahl törvény rámutat az egyprocesszoros szemlélet korlátaira, ha azt kiterjesztik több processzorral történő számításra. [37]

5.3.2. A párhuzamosítottság számítása

Az Amdahl törvény feltételezi, hogy a program felépítése ismert és megvan a lehetőség a párhuzamosítás szempontjából hasznos idők (t_P) és az egyéb idők ($1-t_P$) mérésére. Azonban ha csak a gyorsítás ismert, akkor meghatározható az α_{eff}, ami azt adja meg, hogy mekkora a processzorok átlagos kihasználtsága:

$$\alpha_{eff} = \frac{k}{k-1} \cdot \frac{S-1}{S} \tag{25}$$

ahol a k a processzorok száma, S a gyorsulás, α_{eff} pedig a processzorok kihasználtsága [42] és [43].

A párhuzamosítás hatékonyságát szokásosan az egy processzorra eső gyorsítással lehet jellemezni:

$$E = \frac{S}{k} = \frac{1}{k(1-\alpha)+\alpha} \tag{26}$$

ahol az E az egy processzorra eső gyorsítás, S az elért gyorsítás, k a processzorok száma, α pedig a párhuzamosan futó kódrészlet végrehajtásának ideje (t_p) [42] és [43].

Az egy processzorra eső gyorsítás értéke csökken a párhuzamosított végrehajtási időn kívüli végrehajtási időktől függően és a processzorok számának növekedésével. Az E reciproka megadja az (1- α) jellemzőt.

Az α_{eff} származtatható a (26) egyenletből:

$$\alpha_{eff} = \frac{Ek - 1}{E(k - 1)} \tag{27}$$

ahol az α_{eff} a gyorsítás hatékonysága, E az egy processzorra jutó gyorsítás, k a processzorok száma. Ennek az összefüggésnek az előnye, hogy különböző processzorszámú és gyorsítású rendszerek válnak összehasonlíthatóvá [42] és [43].

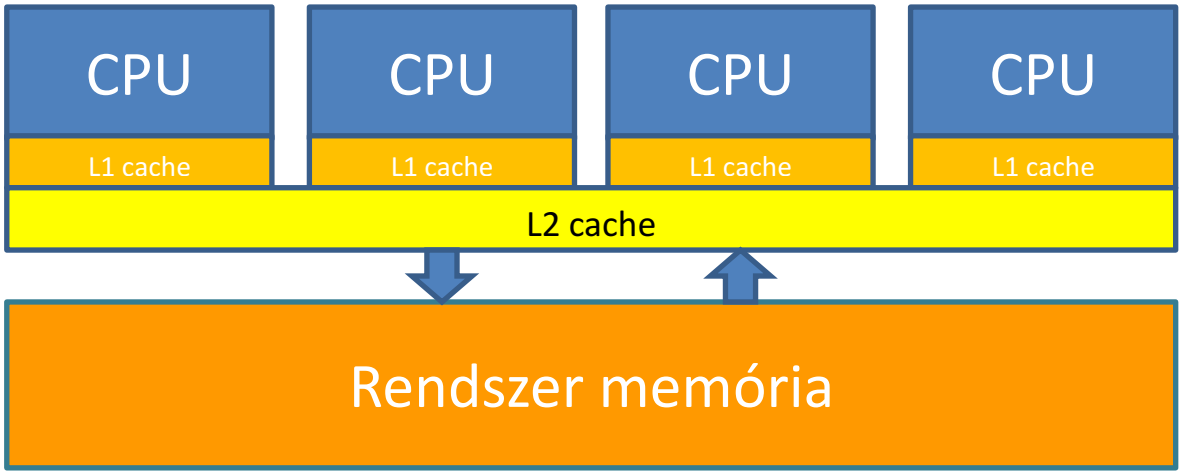
Ez utóbbi tulajdonság hasznos az általam végrehajtott vizsgálatok szempontjából is, mert így több, különböző sokmagos SoC válik összehasonlíthatóvá. A bekezdés az következő forrás alapján készült: [42] és [43].

5.4. Cache mechanizmus

A CPU számítási sebességének növekedését (gyorsulását) nem tudta követni a tároló elemek írási és olvasási sebessége (memória hozzáférés ideje), így az általánosan használt memóriatípusok sem tudnak időben válaszolni az írás/olvasás kérésre. Léteznek azonban nagy sebességű memóriák, de ezek túl költségesek és bonyolultak nagy kapacitású modulok gyártására. A jelenséget, amikor a rendszermemória sebessége nem tudja kiszolgálni a CPU „adatéhségét”, ezáltal lassulást okoz a számításban, memória korlát, Memory Wall jelenségnek nevezik. Orvoslására létezik olyan megoldás, hogy a mikroprocesszor/mikrovezérlő az utasítás végrehajtásakor várakozó ciklusokat illeszt be a programkódba bizonyos órajel frekvencia esetén (STM32F4 sorozatban 144 MHz feletti órajelnél) [44]. Más megoldás, hogy a CPU mellé úgynevezett cache memóriát illesztenek. A cache memória szerepe és a cache mechanizmus feladata megbecsülni az éppen futó programrésznek mely adatszeletre van szüksége a rendszermemóriából és azt betölteni a köztes (cache) memóriába. A cache memória a CPU szempontjából átlátszó, a CPU nem tudja, hogy az adat már a cache-ben van vagy a RAM-ból olvasta.

A cache mechanizmus alapvetően gyors rendszerműködést biztosít, azonban ha a futó folyamat számára szükséges adat nem áll rendelkezésre a cache-ben annak a keresése, betöltése összességében több időt igényel, mintha a processzor egyszerűen a rendszermemóriából dolgozott volna. Rosszul megválasztott adatméret esetén vagy túl gyakori adatcsomag váltáskor fordulhat ez elő.

Másik probléma a többprocesszoros rendszerek esetén fordul elő. A szorosan csatolt többprocesszoros rendszerben az egyes processzormagok közös cache memóriát használnak. Ezek több csatornás memóriák, de egy közös rendszermemóriát használnak forrásnak. Adatmérettől függően az egyes magok versenyhelyzetbe kerülhetnek egymással, amely a rendszer lassulását okozza. A többprocesszoros rendszerek esetén alkalmazzák a többszintű gyorsító tárat ezzel is csökkentve a konkurens hozzáférés esélyét (25. ábra). A CPU tehát nincs közvetlen kapcsolatban a rendszermemóriával.



25. ábra Többprocesszoros cache megoldás

A FIVE módszer µFRI megvalósítása alacsony memóriaigényű. A számítási idők is rövidek, ezért szükséges vizsgálni, hogy sok szabály esetén milyen arányban kell szétosztani a modern sokmagos rendszereken a processzormagok között az egyes részszámításokat.

A FIVE módszer számítási igényét legnagyobb mértékben az anecedensek száma és a szabályok száma növeli. A legtöbb számítási időt az antecedensek és a szabályok távolságának számítása teszi ki [45]. A szakasz további forrásai: [46], [47] és [48].

5.5. A µFRI könyvtár memóriaigénye

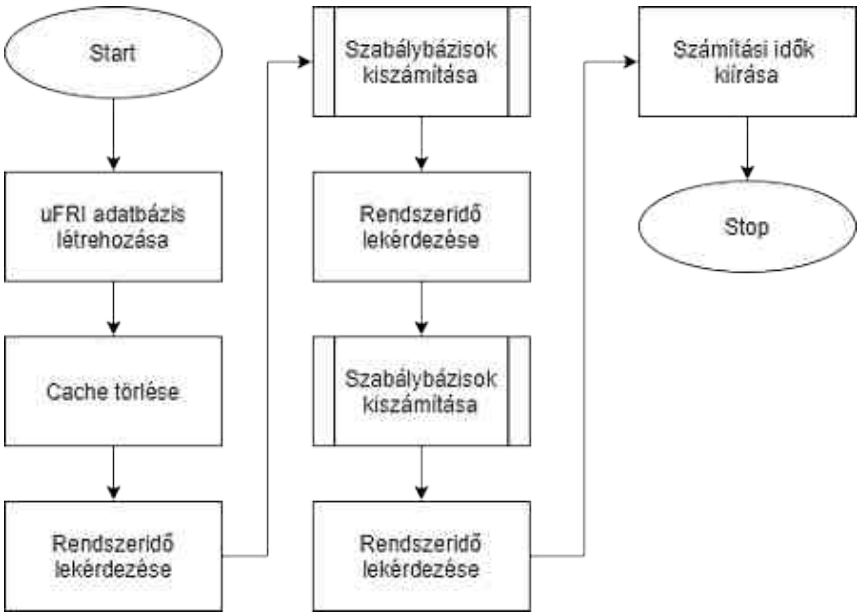
A µFRI könyvtár az adatait struktúrákban tárolja. *A µFRI könyvtár* című fejezet részletesen leírja az adattípusokat és a számítási módot, amellyel a könyvtár memóriaigénye meghatározható adott szabályhalmazhoz.

5.6. Vizsgálat operációs rendszer nélkül

A vizsgálathoz a Zybo kártyát használtam, amely az alábbi SoC-t (**S**ystem **o**n **C**hip) tartalmazza:

- Zybo: 2 db ARM Cortex A9 (650 MHz) + Artix 7 FPGA, L2 cache: 256 kB, L1 cache: 2*16 kB utasítás és adat memória [49]

A Zybo segítségével referencia tetsztet végeztem. A Vivado SDK fejlesztőkörnyezet biztosít a cache memória kiürítésére szolgáló parancsot. A tesztet operációs rendszer nélkül közvetlenül az ARM processzor egyik magján futtattam. Az operációs rendszer akár 30 %-al is ronthatja a számítási időt mivel muszáj fenntartania a saját működését és kezelnie kell a rendszer folyamatait, perifériáit [S13]. A számítás lépései a következő folyamatábrán (26. ábra) láthatók.



26. ábra Számítási eljárás a Zybo-n

A vizsgálatok alatt az L2 cache ki volt kapcsolva, mivel a feldolgozandó adatmennyiség viszonylag kicsi. A program az indulásakor hozza létre az adatbázist előre megadott adatokkal és tetszőlegesen választható szabálybázis darabszámmal. A teszteléshez használt szabálybázis kettő darab, két antecedenst tartalmazó szabályból áll, ezekhez tartozik három darab univerzum. Ebből a szabálybázisból kerül létrehozásra a kívánt darabszámú a teszteléshez szükséges adat. A számításokhoz mindig másik megfigyelést használtam, hogy az eredményt biztosan ne tudja a rendszer tárolni a gyorsítótárban. Az adatstruktúrák feltöltése után a cache ürítése következik és a

rendszeridő lekérdezése. Utóbbi művelet is a Xilinx könyvtára által biztosított függvény. Mivel az operációs rendszer hiánya miatt nincs más folyamat, amely használná a processzort vagy a memóriát és a rendszeridő megfelelően finom felbontású, így egyetlen szabálybázis-halmaz számítása is megfelelő eredményt adott.

Az operációs rendszer nélkül végzett tesztekben használt szabálybázis felépítése a következő:

```
universe "antecdent0"
    "p0" 0 0
    "p1" 5 2
    "p2" 10 10
end

universe "antecdent1"
    "p0" 0 0
    "p1" 5 2
    "p2" 10 10
end

universe "consequent"
    "p0" 0 0
    "p2" 10 10
end

rulebase "consequent"
    rule
      "p0" when "antecdent0" is "p0" and "antecdent1" is "p0"
    end
    rule
      "p1" when "antecdent0" is "p2" and "antecdent1" is "p2"
    end
end
```

A minta kódban az *antecedent0* és *antecedent1* a megfigyeléseket jelölik, míg a *consequent* a generált szabálybázis kimenetét. A *p0*, *p1*, *p2* elnevezések a nyelvi értéket jelölik. A minta szabályrendszer alapján kerül előállításra a szükséges számú szabálybázis. Minden szabálybázis 2 darab, 2 antecedenst tartalmazó szabályt tartalmaz. A vizsgálatkor előállított szabálybázisok száma az alábbi képletek szerint állítja elő a többi elemet (28), (29), (30) és (31):

$$univerzumok\ száma = 2 \cdot szabálybázisok\ száma + szabálybázisok\ száma \tag{28}$$

$$nyelvi\ elemek\ száma = 3 \cdot univerzumok\ száma\text{-}szabálybázisok\ száma \tag{29}$$

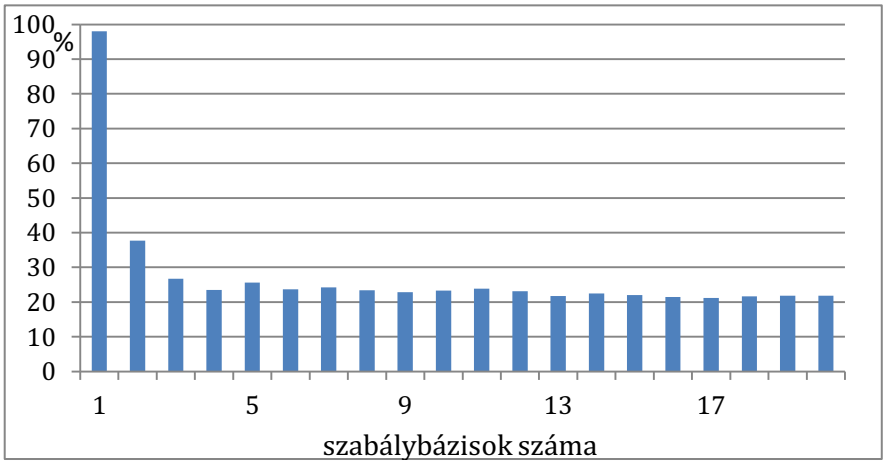
$$szabályok\ száma = 2 \cdot szabálybázisok\ száma \tag{30}$$

$$antecedensek\ száma = 2 \cdot szabályok\ száma \tag{31}$$

ahol a *nyelvi elemek száma* a *p0, p1, p2* jelölésű nyelvi elemek darabszámát jelöli. Az *antecedensek száma* pedig az *„antecdent0" is "p0"* formájú nyelvi értékeket. A fenti képletek alapján számítható a (23) alapján a generált szabálybázisok memóriaigénye a µFRI könyvtárral megvalósítva. Az 1 szabálybázist tartalmazó minta memóriaigénye: 108 byte.

5.6.1. Eredmények

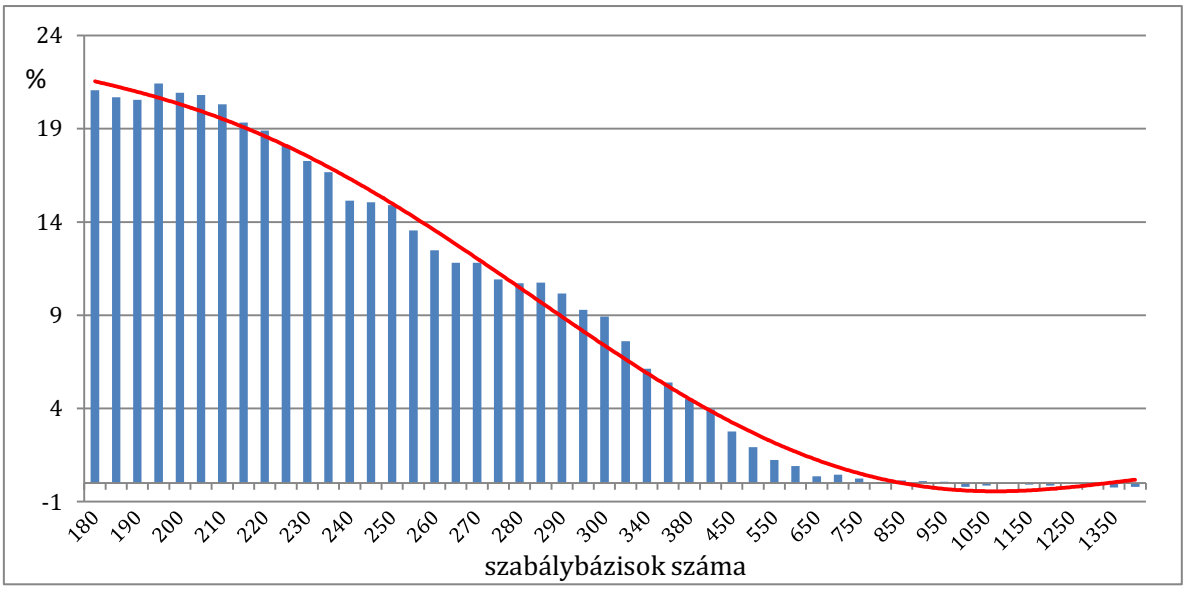
Kis adatméretnél a gyorsítótárban tárolt adatok feldolgozása átlagosan 21,8 %-al volt gyorsabb, mint a hidegen indított számítás, vagyis a cache memória még nem tartalmazza a FIVE számításához szükséges adatokat. Az adatmennyiség növekedésével, folyamatos csökkenés tapasztalható a gyorsítótárban tárolt és külső memóriában tárolt adatokkal végzett számítási idők közötti különbségben (27. ábra). Azonban, még a legkisebb adatcsomagnál sem lesz pontosan kétszer olyan gyors a számítás, mint gyorsítótár nélkül, csak 98 %-os lesz a különbség. A szabályszám növekedésével pedig hirtelen zuhan és beáll a 21 % környékére. Egészen 215 szabálybázisig tart ez az állapot, ahol az adatmennyiség (szabályszám) növekedésével, csökkenésnek indul a gyorsítótár okozta számítási gyorsulás mértéke. A 215 szabálybázis 22,7 kB, ami közel a másfélszerese az L1-es adat gyorsítótárnak.



27. ábra Kis adatmérettel végrehajtott számítási idők közötti különbség. (1. oszlop: 98%)

Az adatmennyiség további növelésével, ahogy telítődik az L1-es cache az alábbi ábrán látható jelenség játszódik le (28. ábra).

Amikor az adatmennyiség eléri a cache memória három és félszeresét (ez 454 szabályt jelent 229 univerzummal 227 szabálybázisban tárolva), elhanyagolható a gyorsítótár hatása. A cache méret hatszorosának megfelelő adatmennyiségnél már a cache memória, lassulást okoz: a görbe negatív tartományba lép.



28. ábra A gyorsítótár hatása az adatmennyiség növekedésével

5.6.2. Összegzés

A teszt során operációs rendszer nélkül közvetlenül a processzormagon történő futtatással vizsgáltam a cache gyorsító hatását a számítási folyamatra. A mért eredmények a feltételezésnek megfelelően megmutatták, hogy egy bizonyos adatmennyiség felett a cache gyorsító hatása már nem érvényesül, sőt minimális lassulás is tapasztalható. Általában nincs szükség ekkora méretű szabályrendszerekre, amelyeket 1 processzoron kell futtatni így az esetek többségében nem fog jelentős lassulást okozni a jelenség csak kivételesen összetett rendszerek esetén.

5.7. Vizsgálatok operációs rendszerrel

Az operációs rendszer működése közben végzett vizsgálatokhoz az alábbi népszerű SoC-t használtam fel:

- Raspberry PI4 B: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8), 64-bit SoC @ 1.5GHz, L2 cache: 1024 kB, L1 cache: 48 kB utasítás és 32 kB adat [50] [51].

Az operációs rendszer az erőforrások elosztását végzi a processzorok között. A Raspberry PI-n a Raspbian 2020 operációs rendszer futott.

Az operációs rendszeren végzett tesztekben használt szabálybázis felépítése a következő:

```
universe "antecdent0"
    "p0" 0 0
    "p1" 5 2
    "p2" 10 10
end

universe "antecdent1"
    "p0" 0 0
    "p1" 5 2
    "p2" 10 10
end

universe "consequent"
    "p0" 0 0
    "p1" 5 2
    "p2" 10 10
end

rulebase "consequent"
    rule
      "p0" when "antecdent0" is "p0" and "antecdent1" is "p0"
    end
    rule
      "p1" when "antecdent0" is "p2" and "antecdent1" is "p2"
    end
    rule
      "p2" when "antecdent0" is "p2" and "antecdent1" is "p2"
    end
end
```

A minta viselkedés az 5.6. -os fejezetben ismertetett képletek alapján került generálásra az alábbi módosítással (32), (33) és (34):

$$szabályok\ száma = 3 \cdot szabálybázisok\ száma \tag{32}$$

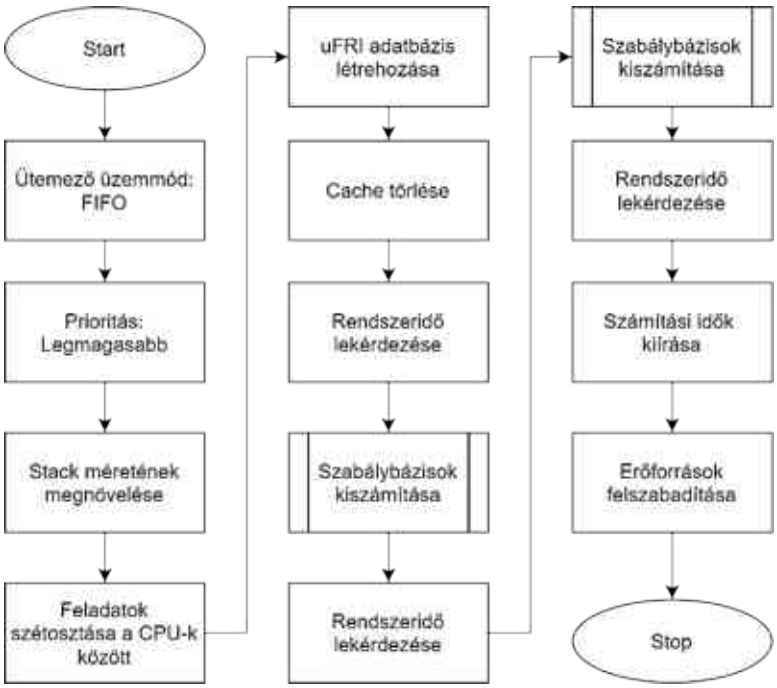
$$nyelvi\ elemek\ száma = 3 \cdot univerzumok\ száma \tag{33}$$

$$szabályok\ száma = 2 \cdot szabálybázisok\ száma \tag{34}$$

Minden szabálybázisban 3 szabály található 2 darab antecedenssel. A szabálybázisok száma változtatható.

Az 1 szabálybázist tartalmazó minta memóriaigénye: 258 byte.

A vizsgálathoz használt program működését az alábbi folyamatábra szemlélteti (29. ábra):



29. ábra Számítási eljárás többmagos, operációs rendszert futtató SoC-n

Ahhoz, hogy a tesztprogram a lehető legnagyobb valószínűséggel kerüljön sorra, az ütemezőt FIFO módra állítottam és a tesztprogram a legnagyobb prioritást kapta. Így a program futása alatt az operációs rendszer semmilyen választ nem tud adni, tehát nagy valószínűséggel ez a program jut a legtöbbször CPU-hoz. A következő lépés a stack megnövelése, hogy a program által használni kívánt adatmennyiség elférjen. Ezt követi a kívánt számú gyerekprocessz létrehozása, maximális számuk a processzormagok számával egyezik meg. Mivel a cache használatát akarom vizsgálni ezért döntöttem úgy, hogy az egyes gyerekprocesszek maguknak hozzák létre a megfelelő mennyiségű szabálybázist. A szabálybázisok felépítése megegyezik a Zybo-n végzett teszttel a

„Vizsgálat operációs rendszer nélkül” című fejezetben. A cache ürítésének módja ebben az esetben a cache teleírása a FIVE szempontjából haszontalan adattal. Az idő lekérdezésére a *time.h clock_gettime* nevű függvényét használtam.

A laphibák figyelése helyett a számítási időben bekövetkező változást figyeltem, amelyet aztán az Amdahl törvénnyel vagy annak módosításával lehet elemezni. Így egy arányszámot kaphatok arról, mekkora számítási gyorsulást jelent, amikor a cache memória feladatának megfelelően tud működni. Illetve milyen mértékben kezdenek vetélkedni a processzorok a cache memóriáért. A Raspberry PI esetén az L1 cache kettő írási és kettő olvasási vonallal van ellátva, amelyért 4 processzormag verseng.

Az alábbi végrehajtási idők kerültek rögzítésre:

- nem gyorsítótárazott adattal végzett számítás ideje processzoronként,
- gyorsítótárazott adattal végzett számítás ideje processzoronként,
- a teljes végrehajtáshoz szükséges idő.

A teljes végrehajtási időben benne van a gyerek folyamatok létrehozásának ideje és végső időpont az, amikor az utolsó gyerek folyamat is befejezte a számítást. Az gyorsítótárazott és nem gyorsítótárazott időben nincs benne, hogy az eredményt fájlba írja a gyerek folyamat. Viszont megjelenik az összes időben, ez befolyásolhatja a teljes végrehajtási időt. Mivel kifejezetten a cache memória által okozott jelenségek megfigyelése a cél, minél jobban a hasznos számítás idejére korlátoztam az időmérést.

5.7.1. Vizsgálatok összefoglalója

A vizsgálatok célját és eljárását az alábbi táblázat foglalja össze (2. táblázat). Minden vizsgálatot elvégeztem 1, 2 és 3 processzor együttes munkájával.

2. táblázat Vizsgálati módszerek összefoglalója

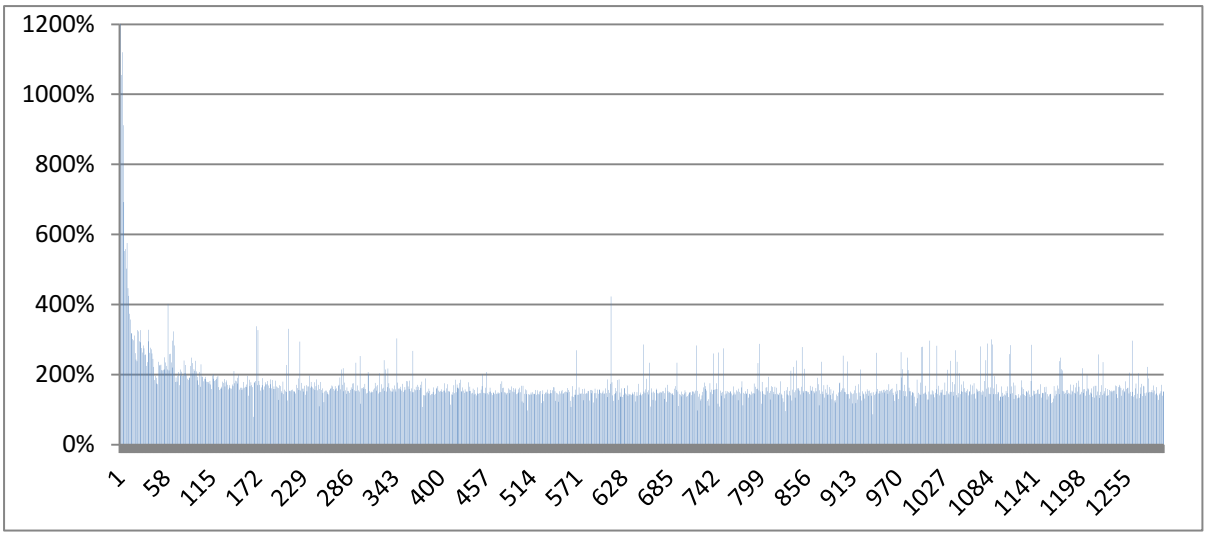
Vizsgálat címe	Cél	Eljárás
Cache vizsgálati eredmények	Mekkora a cache gyorsító hatása? Mekkora adatmennyiségnél éri el a 0 gyorsítást?	Üres cache-el és cache-elt adattal végzett számítások idejének aránya.
Számítási idők összehasonlítása	Hol van jelentős változás a számítási időben vagy azok arányában?	Teljes számítási idők és azok arányának vizsgálata cache-elt és nem cache-elt esetben.
Gyorsítás vizsgálata	Gyorsítás vizsgálata	Amdahl képletének használata. 1 processzorhoz képest számított gyorsítás. Összes számítási idők aránya alapján 1 processzorhoz képest számított gyorsulás 2 és 3 processzor esetén. A processzorok hasznos számítással töltött idejének vizsgálata.

5.7.2. Cache vizsgálati eredmények

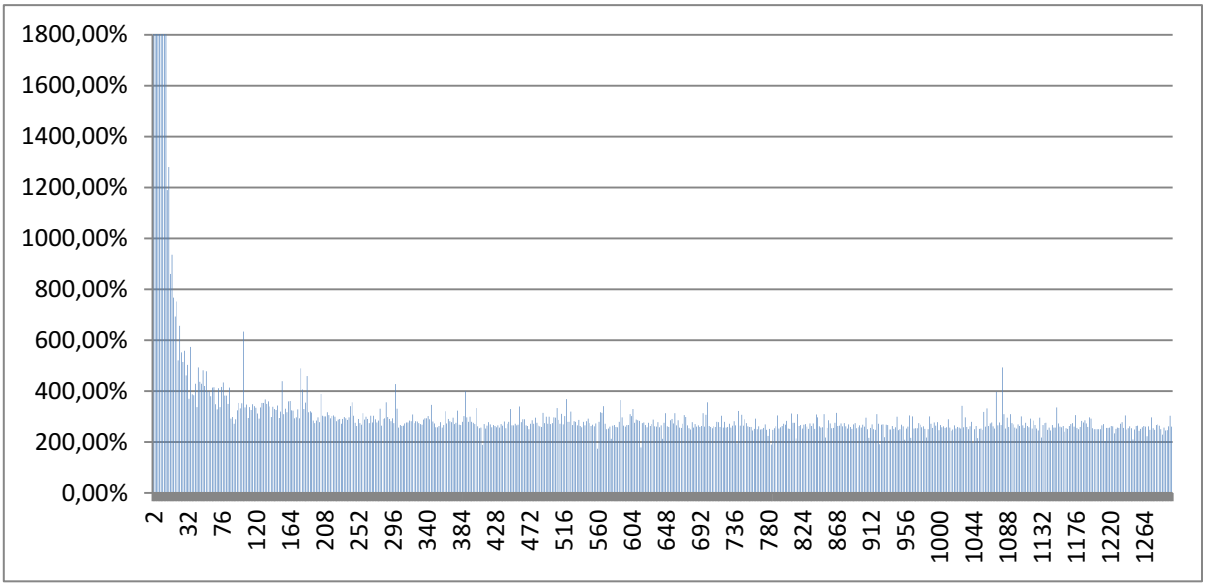
Az első tesztet egy processzormaggal végeztem, amely eredménye hasonló a Zybo-n végzett teszthez. A cél meghatározni hány szabálybázis, vagyis mekkora adatmennyiség esetén esik vissza a cache okozta gyorsulás.

A 31. ábra grafikonja a 27. ábra grafikonjához hasonlóan azt mutatja, hogy mennyivel tart tovább a számítás, ha nincs a cache memóriában a szükséges adat, mintha már megtörtént volna a cache-elés. A számításban 1 CPU vett részt, 10 ezer szabályt számolva. A kezdeti közel 1800 %-os maximális különbséget követően átlagosan 160 %-al tart tovább a számítás, mintha cachelt adattal dolgozna a processzor. A grafikon csak az első 1300 szabálybázist tartalmazza a jobb olvashatóság érdekében. Az operációs rendszer nélküli vizsgálathoz képest látszik, hogy nem csökken 0% közelébe a gyorsítótárazott számítások előnye, hanem beáll egy átlagos értékre.

A cache vizsgálat kiterjesztve 2 processzormagra a 30. ábra szerint alakult. Az átlagos többlet idő megnőtt átlagosan 275 %-ra míg a maximum, amivel lassabb volt az



31. ábra A nem gyorsítótárazott adattal végzett számítások idejének hossza a gyorsítótárazott adattal végzett időkhöz képest 1 processzormagon. A grafikon vízszintes tengelyén a számításban résztvevő szabályok száma látható. Maximum érték: 1844%, átlag: 163%

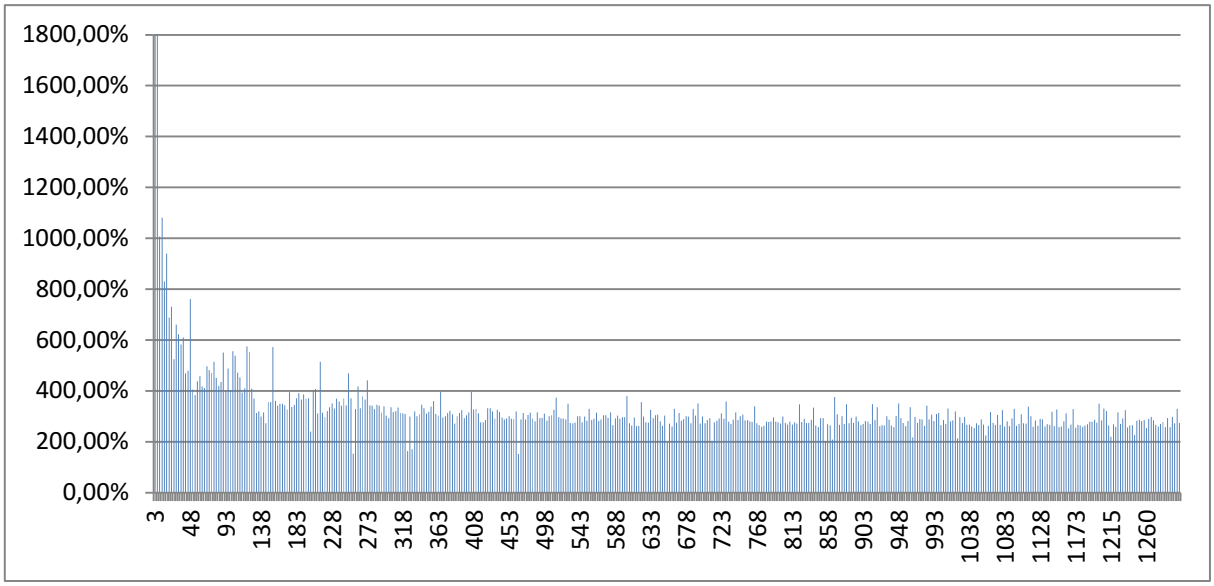


30. ábra A nem gyorsítótárazott adattal végzett számítások idejének hossza a gyorsítótárazott adattal végzett időkhöz képest 2 processzormagon. A grafikon vízszintes tengelyén a számításban résztvevő szabályok száma látható. Maximális érték: 2899 %, átlagérték: 275 %

üres gyorsítótárral indított számítás 2800 %. Utóbbi érték méréssorozatonként kis mértékű eltérés előfordulhat.

A 3 processzormagon végzett számítások eredményeit a 32. ábra szemlélteti. A 2 processzoros vizsgálathoz képest nincs jelentős eltérés az értékekben. Ebben az esetben hasonló viselkedést mutatott, mint 2 processzoros feldolgozás esetén.

A 4 processzormagra végzett teszteknél problémát okozott, hogy a rendszer válaszképtelenné válik a számítások ideje alatt. Ez újraindulást okozott így nem sikerült megfelelő mennyiségű adatot előállítani. 10 ezer szabályszámhoz közel már az ütemező



32. ábra A nem gyorsítótárazott adattal végzett számítások idejének hossza a gyorsítótárazott adattal végzett időkhöz képest 3 processzormagon. A grafikon vízszintes tengelyén a számításban résztvevő szabályok száma látható. Maximális érték: 2384 %, átlagérték: 279 %

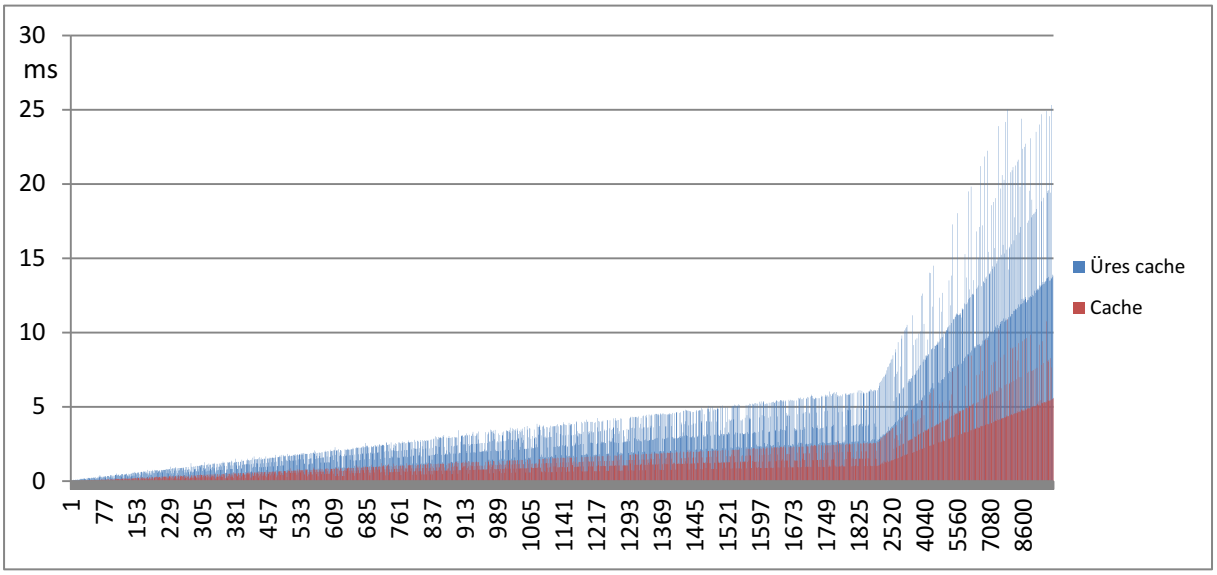
főleg 3 processzormag esetén megszakítást okozott és a számítási idő jelentős növekedését jelentette mind gyorsítótárazott mind pedig nem gyorsítótárazott alkalommal.

Összefoglalva, a tesztben használt algoritmus olyan esetnek felel meg, amikor egy hangolási eljárás egynél több lépést számol ki előre, jelen esetben kettőt. Az eredmények alapján elérhető a számítási idő jelentős csökkenése, ha a hangoló algoritmus figyelembe veszi a gyorsítótárazott adatokkal végzett számítások jobb számítási idejét.

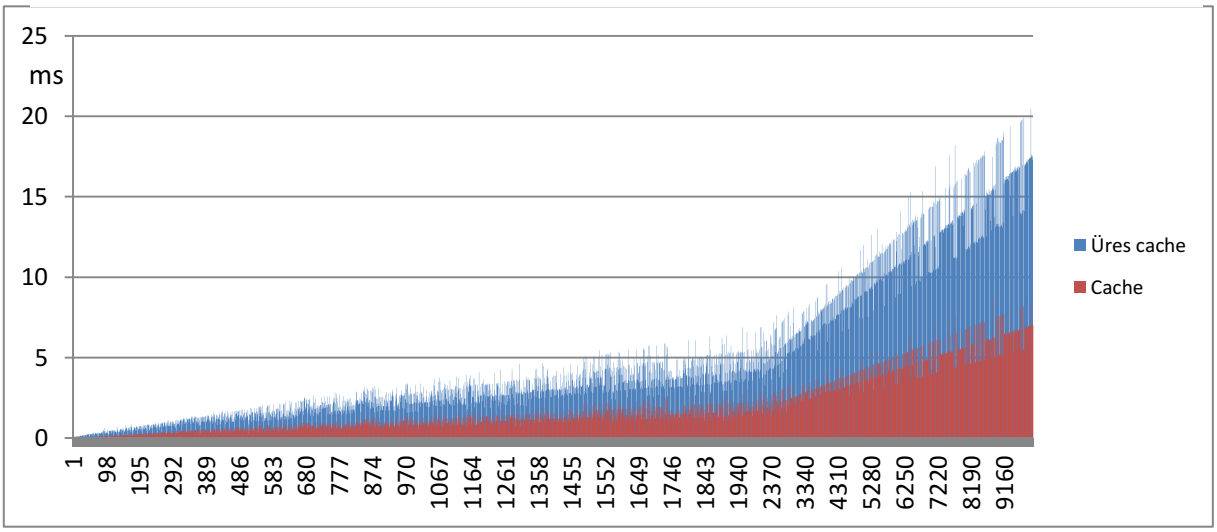
5.7.3. Számítási idők összehasonlítása

A µFRI könyvtára számítási ideje viszonylag rövid, 10 ezer a mintának megfelelő szabálybázis esetén körülbelül 20 ms. Néhány szabálybázis esetén, amely 10-100 szabálybázist jelent 1-10 ns.

Az 33. ábra grafikonja 1 CPU számítási időit szemlélteti. Körülbelül 1850 szabálybázisnál, amely 318 kB adatot tartalmaz és 7400 antecedenst, egy törés alakult ki. Ennél a pontnál 4 ms-ot közelíti a számítási idő és az alapbeállítás szerint 250 Hz-en (5 ms) működő ütemező [52] megszakítja a folyamatot, ezzel jelentős számítási idő növekedést okozva.



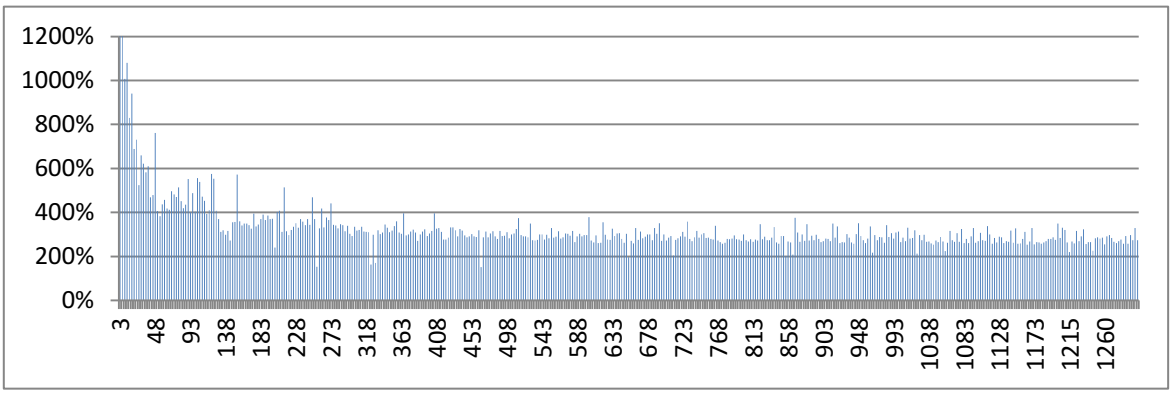
33. ábra A számítási idő változása 1 CPU esetén a szabálybázis számának növekedése függvényében. A vízszintes tengely a szabálybázisok számát mutatja



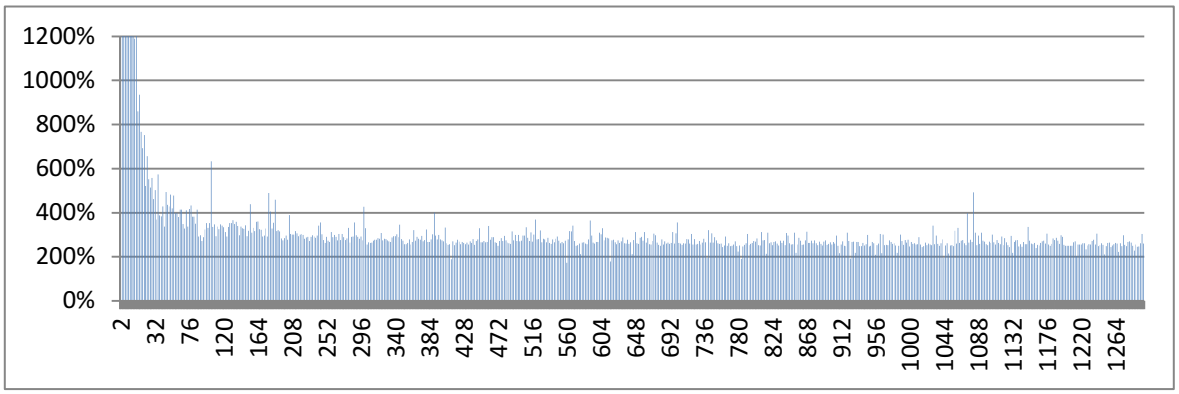
34. ábra A számítási idő változása 2 CPU esetén a szabálybázis számának növekedése függvényében. A vízszintes tengely az 1 CPU-ra eső szabálybázisok számát mutatja

A jelenség megfigyelhető 2 CPU használata esetén is (Lásd: 34. ábra). Mindkét esetben az ütemező a szabad processzormagokat használta. Az összes processzormag használata a rendszer újraindulását okozta, így az ütemező nem tudta ellátni a feladatát.

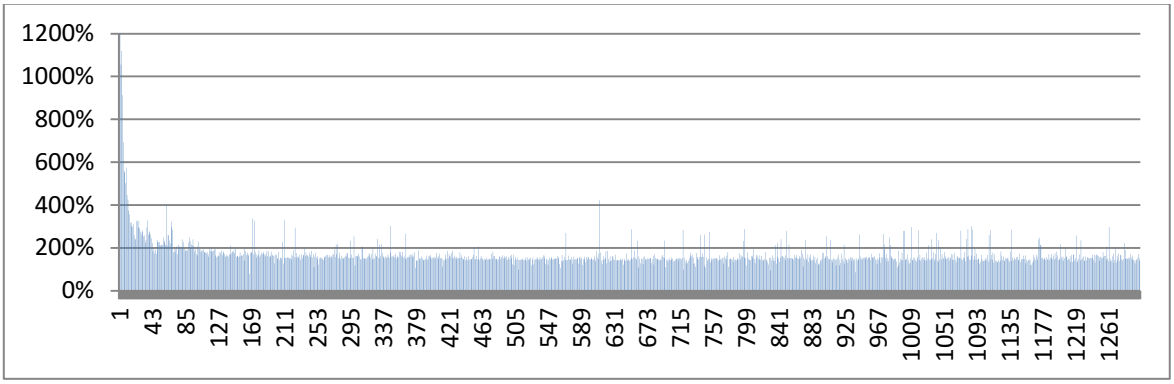
A nem gyorsítótárazott adatokkal végzett számítások időigénye nagyobb adatmennyiség esetén 150-300%-al nagyobbak, mint a gyorsítótárazott adatokkal végzett számítások ideje. Nem volt megfigyelhető az a jelenség, ami a Zybo esetén, hogy eltűnik a különbség nagyobb adatmennyiség esetén (lásd: 37. ábra, 36. ábra, 35. ábra).



35. ábra 2 CPU használata esetén a nem gyorsítótárazott adatokkal végzett vizsgálatok hosszabb ideig tartanak. Az x tengely a számításban részt vevő szabálybázisok számát jelzi.
Átlag: 279 % Maximum: 2384 %



36. ábra 2 CPU használata esetén a nem gyorsítótárazott adatokkal végzett vizsgálatok hosszabb ideig tartanak. Az x tengely a számításban részt vevő szabálybázisok számát jelzi.
Átlag: 275 % Maximum: 2899 %



37. ábra 1 CPU használata esetén a nem gyorsítótárazott adatokkal végzett vizsgálatok hosszabb ideig tartanak. Az x tengely a számításban részt vevő szabálybázisok számát jelzi.
Átlag: 163 % Maximum: 1844 %

Több processzormag használata esetén a kezdeti különbség is nagyobb, ahogy magok versengenek az erőforrásokért, de az átlagos időigény is nagyobb a több szabálybázissal végzett hosszabb ideig tartó számításokkal. A kezdeti különbség több processzormag használata esetén több szabálybázisra is fennáll.

Az időigény növekedése a legnagyobb, ha 1 helyett 2 vagy 3 processzormag dolgozik egyszerre kevés szabálybázison viszont kisebb a különbség, ha 2 helyett 3 processzormag kerül kihasználásra.

Mivel a számítások valós körülmények között többnyire üres cache-el kerülnek feldolgozásra így szükségtelen párhuzamosítással jelentős számítási időt lehet spórolni.

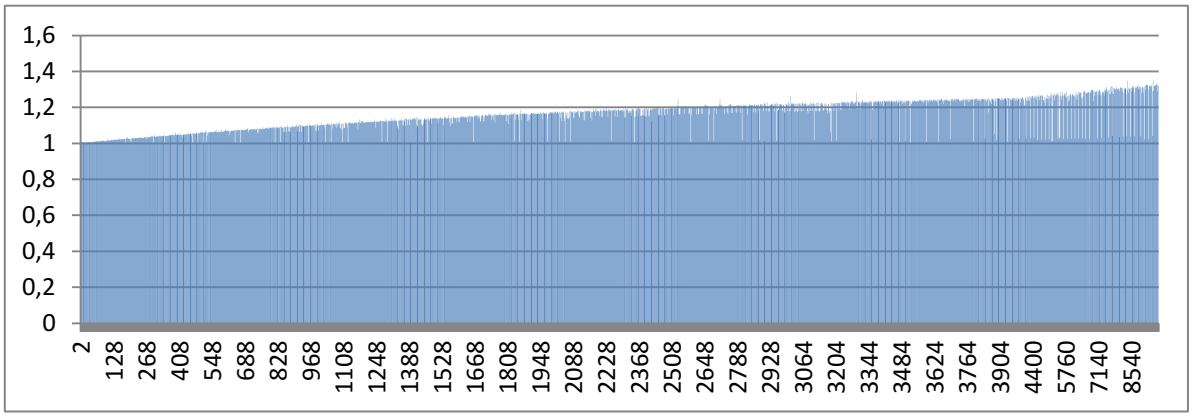
Kivételt képez az az eset, ha már eleve egy-egy processzormag dedikálva van valamelyik szabálybázis-halmazhoz a jobb válaszidő érdekében.

5.7.4. Gyorsítás vizsgálata

A gyorsulás mértékét a számítási idők hányadosával és az Amdahl képletével határoztam meg.

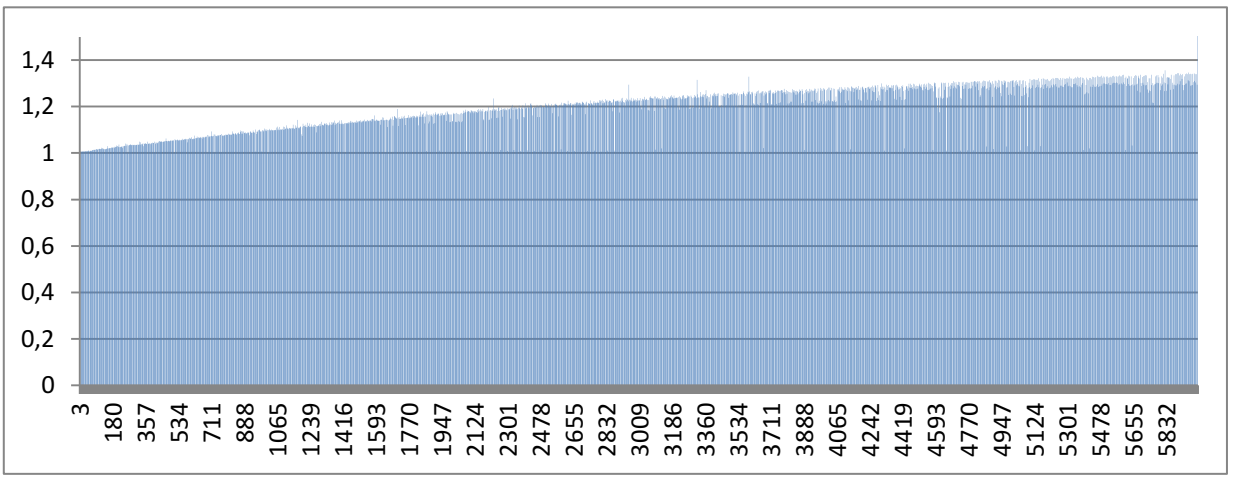
Az Amdahl képlete az egyprocesszoros rendszerhez viszonyított elméleti gyorsulást adja meg több processzor használata esetén. Általánosan a nem párhuzamosított végrehajtáshoz képest mennyivel gyorsabb a párhuzamosított végrehajtás.

Két processzor esetén Amdahl képletébe helyettesítve a mért időket 2200 szabály felett már 1,2-szeres a gyorsulás mértéke és 10 ezer szabálynál megközelíti a 1,4-szeres gyorsulást, (lásd: 38. ábra).



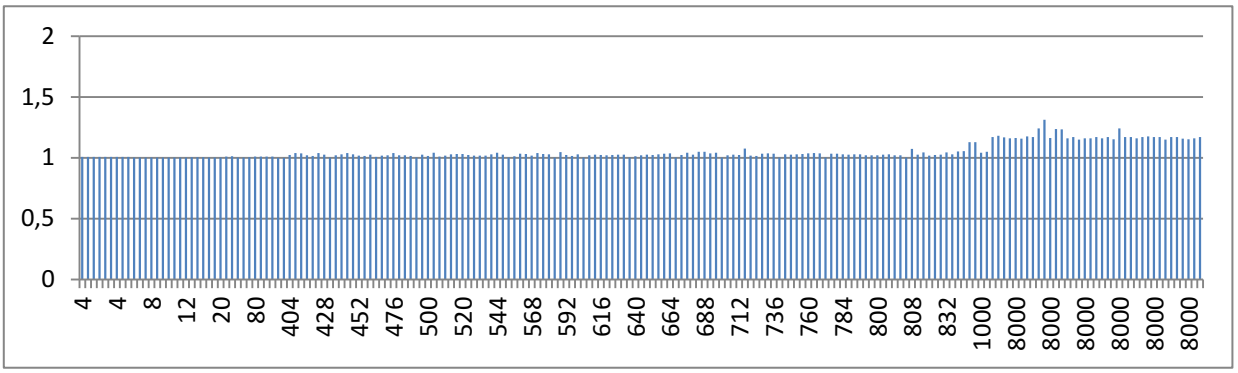
38. ábra Az 1 processzorhoz képest Amdahl képlete alapján számított gyorsulás 2 processzor használata esetén. A vízszintes tengelyen a 2 processzor által együtt számított szabálybázisok száma látható.

Három processzormag esetén a vizsgált szabályszám mellett az elméleti gyorsulás az Amdahl képlete szerint elérheti a 1,2-1,3-szeres értéket,(lásd: 39. ábra).



39. ábra Az 1 processzorhoz képest Amdahl képlete alapján számított gyorsulás 3 processzor használata esetén. A vízszintes tengelyen a 3 processzor által együtt számított szabálybázisok száma látható.

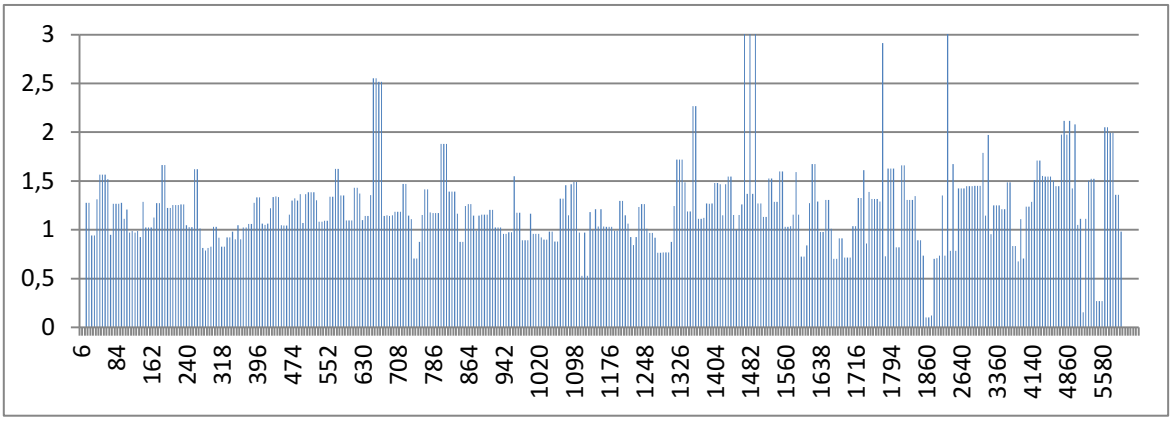
A tesztelő algoritmus a működése alatt a legnagyobb prioritással működik és a résztvevő processzormagokat teljes mértékben lefoglalja. Így a 4 processzormagos vizsgálatok esetén a rendszer működése instabilitást és rendszeres újraindulást produkált, ezért nem sikerült az 1, 2 illetve 3 processzormaggal elvégzett vizsgálatoknak megfelelő adatsort előállítani. Az 1000 szabálybázisig végzett vizsgálatok eredményén látszik, hogy a processzorok vetélkedése miatt nincs jelentős javulás az 1 processzoros végrehajtáshoz képest, (lásd: 40. ábra).



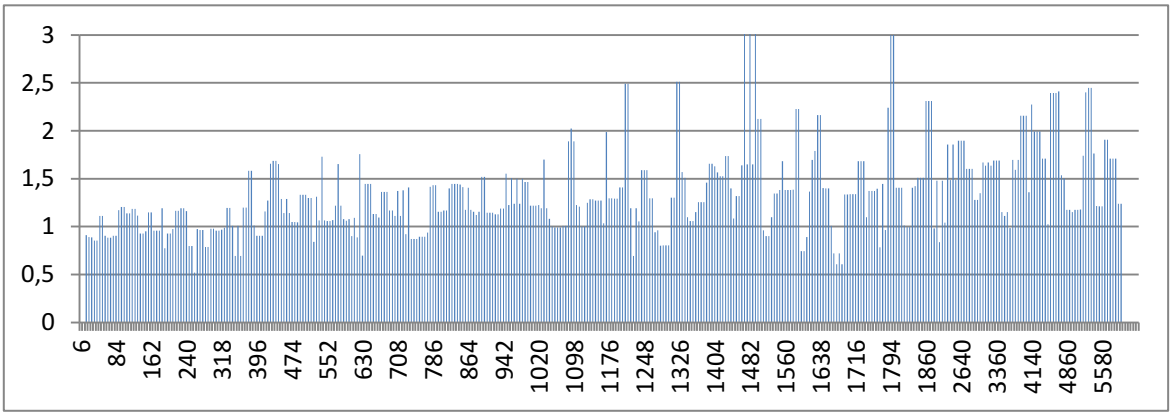
40. ábra Az 1 processzorhoz képest Amdahl képlete alapján számított gyorsulás 4 processzor használata esetén. A vízszintes tengelyen a 4 processzor által együtt számított szabálybázisok száma látható.

A fenti eredmények a tesztelő algoritmus működését tükrözik, viszont a µFRI viszonylag rövid számítási ideje miatt lehet következtetni a FIVE módszerrel végzett számítások hatékonyságára is. A több processzorral végzett számítások esetén az egyes magok által előállított számítási idők átlaga szerepelt Amdahl képletében, tehát a

maximális érték használatához képest egy optimistább eredmény látható a fenti grafikonokon. A következő grafikonokon az 1 processzorhoz viszonyított gyorsulás mértéke jelenik meg 2 dolgozó processzormag (41. ábra), 3 dolgozó processzormag esetén (42. ábra) és mekkora gyorsulás keletkezett, ha 2 helyett 3 processzormag vesz részt a számításban (43. ábra). Az átlagos érték az Amdahl képlettel egybevágóan 1,2-1,3-szeres gyorsulást mutat. Az ábrázolt adatokon mediánszűrést végeztem a jobb olvashatóság érdekében.



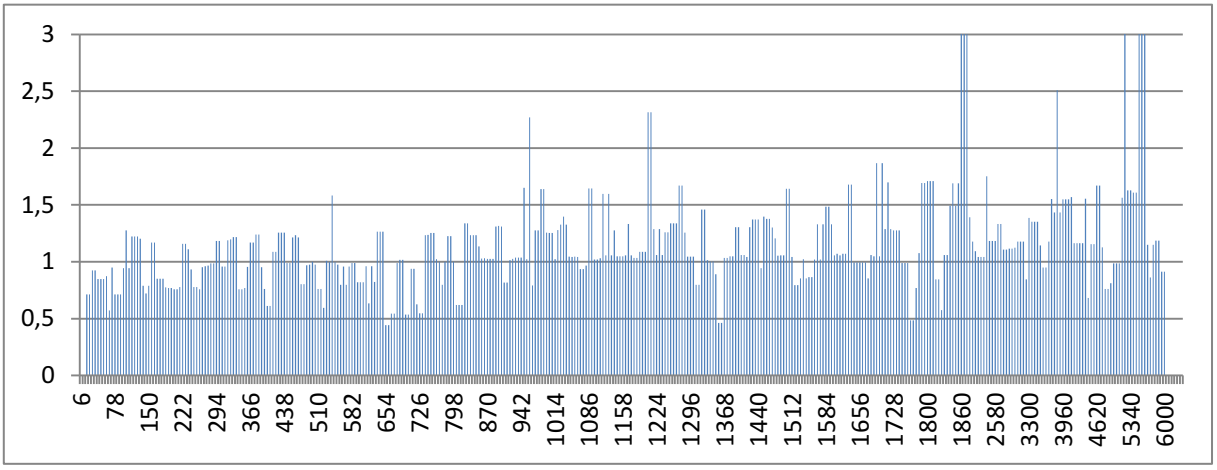
41. ábra Az 1 processzorhoz képest össz számítási idők alapján számított gyorsulás 2 processzor használata esetén. A vízszintes tengelyen a 2 processzor által együtt számított szabálybázisok száma látható. Átlag: 1,42 Medián: 1,16 Módusz: 1,29



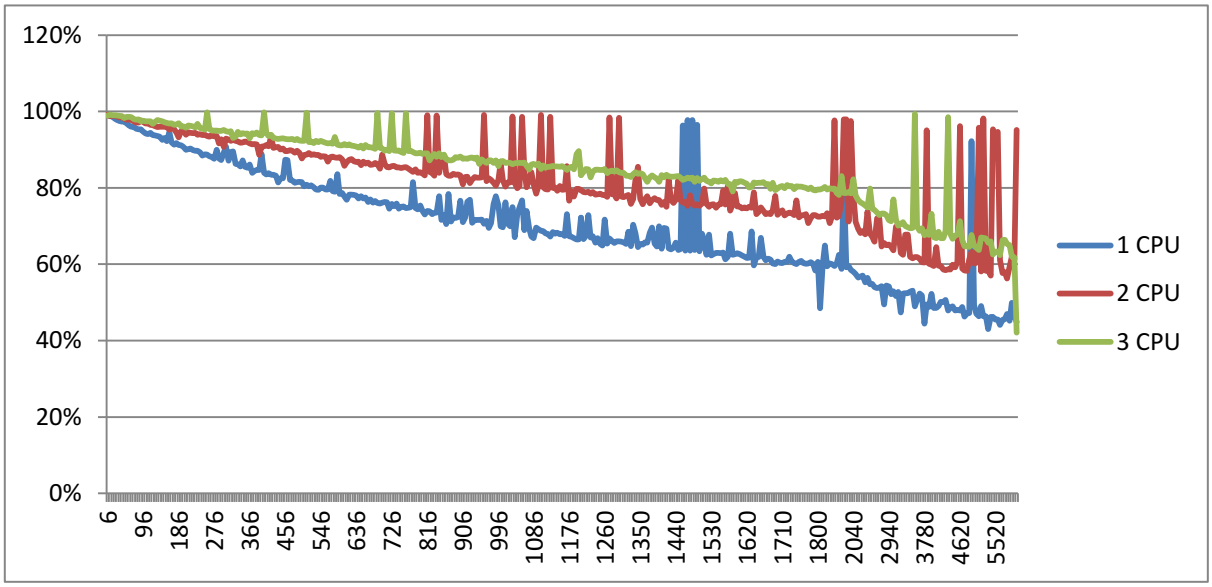
42. ábra Az 1 processzorhoz képest össz számítási idők alapján számított gyorsulás 3 processzor használata esetén. A vízszintes tengelyen a 3 processzor által együtt számított szabálybázisok száma látható. Átlag: 1,5 Medián: 1,25 Módusz: 1,4

A számítások elvégzéshez szükséges egyéb műveletek, mint a gyerek folyamatok létrehozása, memória foglalás stb. időbeli költsége jelentős. Kis szabályszám esetén 99%-ot is elérheti, csak ezres nagyságrend környékén csökken 80% alá. Ahogy a grafikonon is látszik (44. ábra), a legjobb időbeli költség 1 processzormag használata esetén érhető el 10 ezer szabálybázis alatt. A processzorok által végzett számítások

idejének itt az az átlagát használtam fel. A kérdéses 1800 szabálynál látszik a fordulópont, ahogy az operációs rendszer 5 ms számítási idő felett már egyre többször szakítja meg a számítási folyamatokat. Amikor csak egy processzormag végzi a műveleteket ennek hatása nem jelentős, de több processzormag esetén már többletköltséget okoz ezzel növelve a számítási időt.



43. ábra A 2 processzorhoz képest össz számítási idők alapján számított gyorsulás 3 processzor használata esetén. A vízszintes tengelyen a 3 processzor által együtt számított szabálybázisok száma látható. Átlag: 1,9 Medián: 1,05 Módusz: 1,08



44. ábra A számításhoz kapcsolódó időbeli költségek százalékos megjelenítése a teljes számítási időhöz viszonyítva. A vízszintes tengelyen a processzorok által együtt számított szabálybázisok száma látható.

Az eddigi eredmények az egész tesztelésre használt algoritmus időértékei alapján készültek el. Az alábbi táblázat (3. táblázat) csak a µFRI könyvtár számítási idejéből számított gyorsulásokat tartalmazza az előbbiekhez hasonlóan 10 ezer szabálybázisig

ciklikusan végrehajtva a műveleteket. Ha 1 helyett 2 processzormag dolgozik üres gyorsítótár esetén és gyorsítótárazott adatok esetén is az átlagos gyorsulás 1,07. A leggyakoribb érték, a módusz 1,01 üres gyorsítótár esetén míg gyorsítótárazott adatok esetén 0,83, tehát hosszabb számítási időt kapunk 1 processzorhoz képest. Abban az esetben, ha 1 helyett 3 processzor dolgozik ugyan annyi szabálybázison az átlagos gyorsulás 0,76-ra csökken, míg a módusz 1.

3. táblázat A µFRI könyvtár számítási időben bekövetkező gyorsulás, amikor 1 helyett 2 (1 -> 2 CPU) vagy 1 helyett 3 processzormag (1 -> 3CPU) végzi a számításokat

	Üres gyorsítótár		Gyorsítótárazott	
	1 -> 2 CPU	1 -> 3 CPU	1 -> 2 CPU	1 -> 3 CPU
Átlag	1,07	0,76	1,07	0,71
Módusz	1,01	1	0,83	1
Medián	1,03	0,71	1,01	0,67
Minimum	0,46	0,31	0,35	0,26
Maximum	2,55	1,85	2,61	1,83

5.8. Eredmények összefoglalása

Az eddigi tapasztalatok alapján a tesztek során alkalmazott szabálybázisszám jóval magasabb, mint amire szükség van egy rendszer működtetése során. A néhány szabálybázistól 100-300 szabály fordul elő a megerősítéses tanulás során generált tudásbázisban. Ezek alapján a µFRI könyvtár a rövid számítási ideje miatt nem ajánlott többprocesszoros végrehajtásra alacsony szabálybázisszám esetén. Mivel a szabályok és antecedensek száma változhat a szabálybázisokban, memóriaigénnyel meghatározva a vizsgálatokban alkalmazott szabálybázisok együttes mérete 1720 kB a (23) számú képlet alapján.

5.9. Tézis

A µFRI könyvtár szorosan csatolt ARM A72 párhuzamos architektúrán való implementációja esetén a dinamikusan változó elosztott fuzzy tudásbázis összesen 1720 kB memóriaigényig hatékonyan allokálható 1 processzormagon.

Tetszőleges szabálybázis memória igénye kiszámítható a következő képlettel:

$$T = 8N_{UE} + 12N_U + 12N_A + 14N_R + 12N_{RB}$$

ahol az N_{UE} az összes univerzum elem száma, N_U az összes univerzumok száma, N_A az összes antecedens száma a szabályokban, N_R a szabályok száma, N_{RB} a szabálybázisok

száma, T pedig a teljes szükséges tárterület mérete byte-ban. Ez alapján a legkisebb elfoglalt memóriaterület 58 byte, amikor $N_{UE}=N_U=N_A=N_R=N_{RB}=1$.

5.10. Új eredmények

A fejezetben a µFRI könyvtár különböző tárigények esetén vizsgáltam meg a cache gyorsító hatását és a többmagos processzors rendszerek (ARM A53/A72) viselkedését a számítás által igényelt adatmennyiség esetén.

A vizsgálatok során kiderült, hogy a µFRI könyvtárat több processzormagon futtatva főként a rövid számítási ideje miatt sebességcsökkenés figyelhető meg illetve nincs jelentős gyorsulás az 1 processzoros végrehajtáshoz képest szorosan csatolt többprocesszoros rendszer esetén.

5.11. Gyakorlati felhasználás

A FIVE módszer µFRI C nyelvű implementációja beágyazott rendszereken történő felhasználásra készült. Az eredmények segítségével megspórolható a többprocesszoros implementáció alacsony szabályszám esetén kivéve, amikor a rendszer gyors válaszideje szükséges és lazán csatolt többprocesszoros rendszeren kerül alkalmazásra.

5.12. Összegzés

A µFRI könyvtár számítási idejének vizsgálatát végeztem el abból a célból, hogy a különböző adatmennyiségek esetén a cache méretet figyelembe véve mekkora szabálybázis méretnél érdemes több processzorra bízni a számítást. Az eredmények alapján kiderült, hogy szorosan csatolt többprocesszoros rendszerek esetén nem javul a hatékonyság, ha több processzor végzi a számításokat. Illetve az ütemező megszakításai okozhatnak jelentős, korábbi tesztek alapján akár 30%-os növekedését a számítási időnek, ha a számítások hosszabb időt vesz igénybe, mint az ütemező által kiosztott időszelet.

6. A FIVE módszer hardveres implementálása

A FIVE interpolációs eljárás könnyűsúlyú, jól alkalmazható beágyazott rendszer környezetben. A további gyorsítása indokolt a szimulációs és öntanulási eljárások időigényének csökkentése mellett a gyors beavatkozást igénylő folyamatok szabályozása miatt is. Ennek egy lehetséges megoldása a FIVE módszer hardveres implementációja, amely lehetővé teszi a számítás egy egyszerűsített formájának rendkívül gyors végrehajtását. Az ehhez kialakított adatstruktúrák pedig lehetővé teszik a hardveresen megvalósított szabálybázisok dinamikus konfigurálását és hangolását.

6.1. Célkitűzés

A FIVE módszer számítási lépéseinek hardveres megvalósítása hardverleíró nyelven úgy, hogy a szabálybázisok, univerzumok paraméterei rendszerleállás nélkül megváltoztathatóak legyenek.

6.2. FPGA és SoC áramkörök

Az FPGA (**F**ield-**P**rogrammable **G**ate **A**rray) áramkörök egyik előnye, hogy a már beültetett áramkör belső struktúrája bármikor megváltoztatható működés közben (dinamikus függvény módosítás – Dynamic Function Exchange DFX). A rendelkezésre álló áramköri erőforrások korlátain belül tetszőleges logikai hálózat kialakítható, legyen az egyszerű logikai hálózat, matematikai függvény vagy akár processzoros rendszer is. Előnyös tulajdonsága az általános célú processzorokhoz képest a relatíve alacsony energiaigény és a magas számítási sebesség, amelyet az algoritmusok párhuzamosításával lehet elérni. A kialakított áramkör újrakonfigurálható, akár működés közben részenként (parciálisan), amely tovább növeli a rugalmas felhasználhatóságát. [53]

A Xilinx 2020-ban létrehozta a Vitis Software Platform nevű fejlesztő rendszerét, amely lehetővé teszi összetett, adaptív rendszerek és hardveresen gyorsított alkalmazások fejlesztését. A Vitis lehetővé teszi a fejlesztett modulok hordozhatóságát és újrafelhasználhatóságát platformok között. Támogatja a szimulációt, emulációt és

hatékony hibakeresési lehetőségeket biztosít. Az ARM vagy x86 architektúrán futó programkódot futási sebességét a rendszerhez PCIe sínrendszeren vagy MPSoC (**M**ulti**P**rocessor **S**ystem **o**n a **C**hip – többprocesszoros rendszer a chip-en) megoldásokhoz integrált újrakonfigurálható eszközök támogatják. A Vitis segítségével olyan komplex rendszer hozható létre, amely egyrészt processzoron futó kódból és az azt támogató gyorsító hardverből áll. Utóbbi újrakonfigurálható áramkörön kerül megvalósításra az alkalmazásnak megfelelően. A rendszer képes közvetlenül operációs rendszer nélkül a processzoron működni vagy FreeRTOS és Linux alapú rendszerekhez kapcsolódni. [54]

Az FPGA fenti előnyös tulajdonságai lehetővé teszik fuzzy alapú etorobotikai viselkedésmodellek megvalósítását (részletesen: [1]), hogy az akár rövid válaszidejű rendszerek esetén is valós időben számítható legyen. A hardveres gyorsító által biztosított számítási idő csökkenés és párhuzamosított végrehajtás jelentősen csökkentheti a Q-tanulás idejét is [31].

6.3. A FIVE módszer megvalósításának követelményei

A FIVE fuzzy interpolációs módszer alapvetően gyors működést biztosít, de bizonyos esetekben akár korlátozásokkal is szükséges lehet a még rövidebb számítási idő. A módszer jól párhuzamosítható, amivel tovább csökkenthető a számítási idő. Azonban ahogy az előző tézisben bemutatásra került a alapvetően kevés adattal, rövid idejű számításokat végez így egy operációs rendszert is futtató processzoron a feladat felosztása akár lassulást is okozhat, lásd: „A FIVE módszer párhuzamos számítási sebességének gyorsítása” című fejezet.

Az áramkörrel szemben támasztott követelmények:

- működés közben hangolható, vagyis a szabálybázis paraméterei megváltoztathatók,
- egyszerű kapcsolódási lehetőség a külvilággal,
- skálázható felbontás,
- hordozhatóság, többfajta FPGA-n is implementálható megvalósítás,
- az áramkör előállítható az FBDL-ből.

A fenti követelmények alapján készítettem el a FIVE módszert megvalósító számítási egység tervét.

6.4. A FIVE módszerhez kapcsolódó adattárolási struktúrák

A hardveres megvalósítás követelményei közé tartozik a gyors működés mellett, hogy a szabálybázis hangolható legyen. Ez azt jelenti, hogy az FBDL-ből örökölt paraméterek megváltoztathatók: az *Universe* elemeinek és *Rule* elemeinek értékei. Pontosabban a megfigyelések, a szabályok antecedensei és konzekvens értékei változtathatóak. A szabályok és univerzumok száma nem változhat.

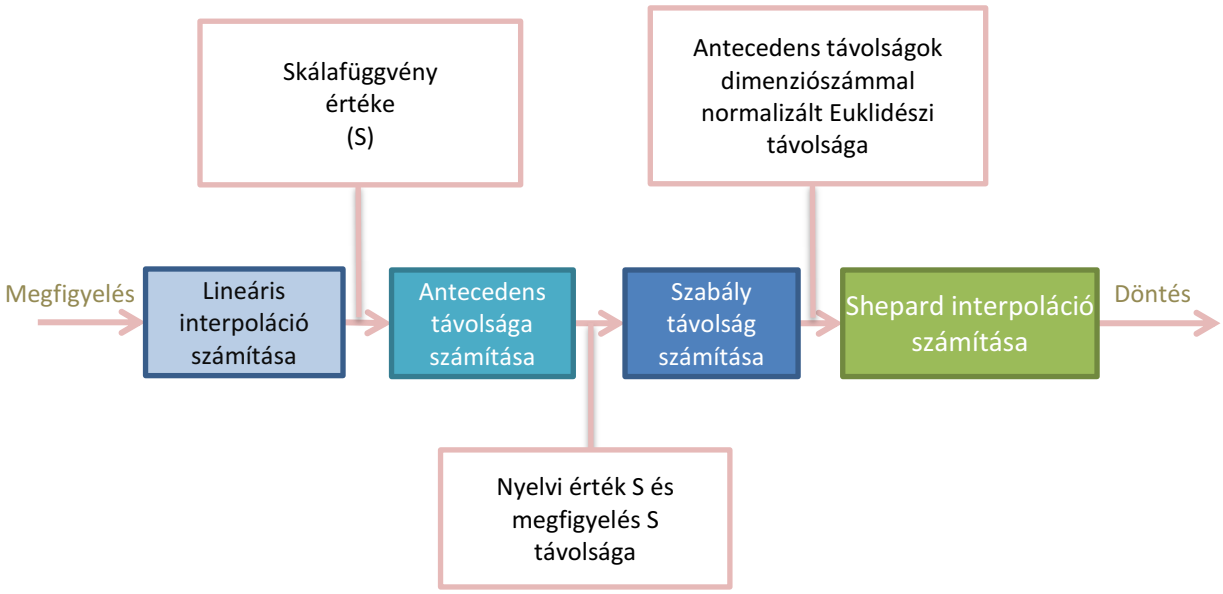
Az FPGA lehetőséget ad az adatok tárolására Block RAM-ban (FPGA-ban használt memória blokk nagy mennyiségű adat tárolására), ahol írható és olvasható a tárolt adat, viszont párhuzamos hozzáférésnél kialakuló versenyhelyzet ronthatja a számítási sebességet hasonlóan a processzoros környezetben. Ehhez hasonló eset, amikor az esetlegesen telepített rendszer memóriában tárolja az adatokat. Másik lehetőség a regiszterek használata. Ebben az esetben a közös tároló helyett kialakítható az egyes moduloknak saját regiszterkészlet, amelyhez egyedüli hozzáférésük van. Mindkét esetben a konfiguráció tartalmazza a kezdőértékeket, a hangolás után vagy amikor a szabályok száma változik, az értékeket ki kell olvasni és elmenteni, beírni az új konfigurációba.

A FIVE módszer megvalósítása a regisztereket használja, így olyan FPGA-kon is használható, amelyek nem tartalmaznak Block RAM-ot. A számítási blokkonként létrehozott regiszterek a blokkok számára bármikor hozzáférhetőek. Hátrányként említhető, hogy el kell készíteni a kérdéses regiszterekből kialakított memóriák írási és olvasási megoldásait. Az írási és olvasási eljárásokat pedig összehangolni a számítási feladatokkal: a tárolók írásakor szünetel az eredmények előállítása. FRIQ-tanulás esetén, amikor változik a szabálybázis szerkezete (szabály beszúrása szükséges vagy szabályok összevonása [55]) [29], a pillanatnyi paramétereket szükséges menteni és az új konfigurációba feltölteni. Ez két módon lehetséges, már eleve a konfiguráció tartalmazza a paramétereket vagy pedig az alapállapotra beállított értékek kerülnek felülírásra.

6.5. A FIVE megvalósítási terve

A FIVE módszer 4 számítási lépéssel képes az eredmény előállítására, amely 2 interpolációs számításból és 2 távolságszámításból áll.

A 45. ábra szerinti adatfolyam feldolgozás ábrázolása egy szabálybázis számítási lépéseit tartalmazza. Ahol a *Shepard interpolációs* modul kivételével az egyes modulok száma függ attól, hogy hány darab szabályt illetve antecedenst és megfigyelést tartalmaz a szabálybázis. A *Lineáris interpoláció számítások* számát a megfigyelések száma határozza meg, az *Antecedens távolság* számítások száma az egyes szabályokban található antecedensek száma alapján változik. A *Szabály távolság* számítása modulból minden szabályhoz egy darab tartozik. A *Shepard interpoláció számításá*ból egy szabálybázis egyetlen darabot tartalmaz.



45. ábra A FIVE módszer számítási lépései. A nyíl az adatfolyamot jelzi.

A *Lineáris interpoláció számítása* a végső megvalósításban Universe néven szerepel. A többi modul a magyar elnevezés angol megfelelőjével került jelölésre.

A megvalósításra a Verilog hardver leíró nyelvet választottam. Emellett kísérleti jelleggel elkészítettem a Vivado High Level Synthesis eszközzel is a hardvert az alap µFRI könyvtárat módosítás nélkül és optimalizálási módosításokkal felhasználva.

A FIVE FPGA megvalósítása egy szinkron számítási lánc. A teljes működésre jellemző, hogy az áramkör minden egysége a bemenő adatok megváltozására reagálva állítja elő a kimeneti értéket.

A hardver változtatható bitszélességgel képes működni, alapesetben 8 bites előjel nélküli egész számokkal dolgozik a bemenetein és a kimenetén. A Verilog nyújtotta paraméterezés segítségével a bitszélesség változtatható, a paraméternek megfelelően az érintett sínrendszerek szélességét átszámolja. Továbbá a számításhoz szükséges adatok is paraméterben megadhatók. A Verilog kód az FBDL-ben leírt viselkedés alapján

készült. A paraméterek segítségével az értékek megváltoztatásához nem szükséges újragenerálni a Verilog kódot az FBDL-ből.

Az egyes számítási egységek és a befoglaló egység is állandó és változó részekből épül fel. Azok a részek, amelyek száma a szabályok, szabálybázisok tulajdonságai alapján változnak (szabályok száma, megfigyelések száma, antecedensek száma stb.) a kódban kiemelve, az FBDL-ből előállítható formában kerültek a Verilog kódba.

A számítási egységek szerinti felosztás biztosítja a pipeline működést. Az egyes egységek tartalmaznak egy belső bemeneti és egy kimeneti tároló regisztert, ahol a beérkező adat és az eredmény tárolható. A jelenlegi terv szerint minden egység 1 órajelen belül ad eredményt kivéve az Universe egység, amely két lépésben állítja elő az eredményét. A továbbiakban ismertetem az egyes egységeket/blokkokat.

6.5.1. Az FRI_FIVE befoglaló modul

A következőkben felsorolt modulokat az FRI_FIVE modul fogja össze. Ez a modul valósít meg egyetlen szabálybázist és tartalmazza a ki-, és bemeneteket, amellyel az irányító rendszerhez kapcsolódik. A belső felépítése teljesen generált kód az FBDL-ben leírt szabálybázis alapján. Egy FRI_FIVE modul egyetlen szabálybázist valósít meg. Blokk-rajzát a 46. ábra mutatja.

Bemenetei a következők:

- *clk*: A rendszer órajel bemenete.
- *rst*: Reset jel az alaphelyzetbe állításhoz.
- *observation*_X: A rendszerre kapcsolt megfigyelések bemenete. Az X a bemenetek sorszáma darabszámtól függően 0-tól indulva.

Kiementei:

- *consequent*: A szabálybázis döntésének értéke.

- *result_ready*: A számítás kész állapotát jelző port, hibakeresési célokat szolgál.

46. ábra Az FRI_FIVE befoglaló modul portjai

Az FRI_FIVE jelenlegi változata nem tartalmazza a kommunikációs blokkot így a hozzá tartozó portok sincsenek feltüntetve. Csak a számítás működőképességének vizsgálatára szolgál. (lásd Melléklet)

6.5.2. Az Universe modul

Az *Universe* modul tartalmazza az FBDL *universe* kulcsszóhoz tartozó nyelvi változók értékeit. Fogadja az bemenetről érkező adatokat, amely származhat valamilyen érzékelőből vagy másik szabálybázisból esetleg a processzortól. Eltérő bitszélességek esetén egy skálázó hardvert érdemes beiktatni a bemenetek elé.

A *Universe* modul két megoldással készült el:

- Keresőtáblás megoldás: a lehetséges értékeket egy keresőtábla tartalmazza a teljes tartományra (túl nagy bitszélesség esetén nem ajánlott a túl sok variáció miatt). Ekkor nincs benne a lineáris interpoláció, a teljes tartomány az FBDL-ben megadott értékek alapján kerül kiszámításra.
- Lineáris interpolációs megoldás: csak az FBDL-ben értékeket tartalmazza az adatbázis. A köztes értékek a FIVE lineáris interpolációjának megfelelően működés közben kerülnek kiszámításra.

A modul az alábbi bemeneti portokat tartalmazza:

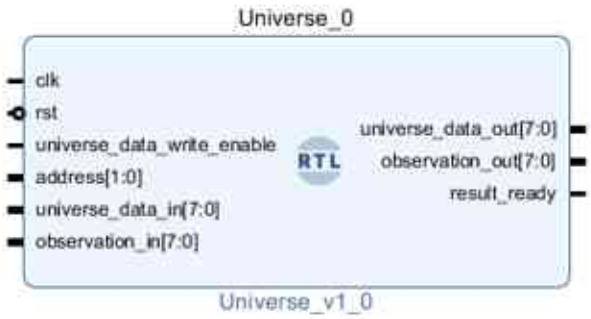
- *clk*: Órajel bemenet.
- *rst*: Reset jel.
- *universe_data_write_enable*: Az adatbázis frissítésére szolgál: 0 esetén olvasás, 1 bemenő jel esetén írás a megadott címre.

- *universe_data_in*: Az adatbázisba írandó adatot fogadó port.
- *address*: Az adatbázisba írandó vagy onnan olvasandó adat címe.
- *observation_in*: A megfigyelések értékét fogadó bemeneti port.

A modul az alábbi kimeneti portokat tartalmazza:

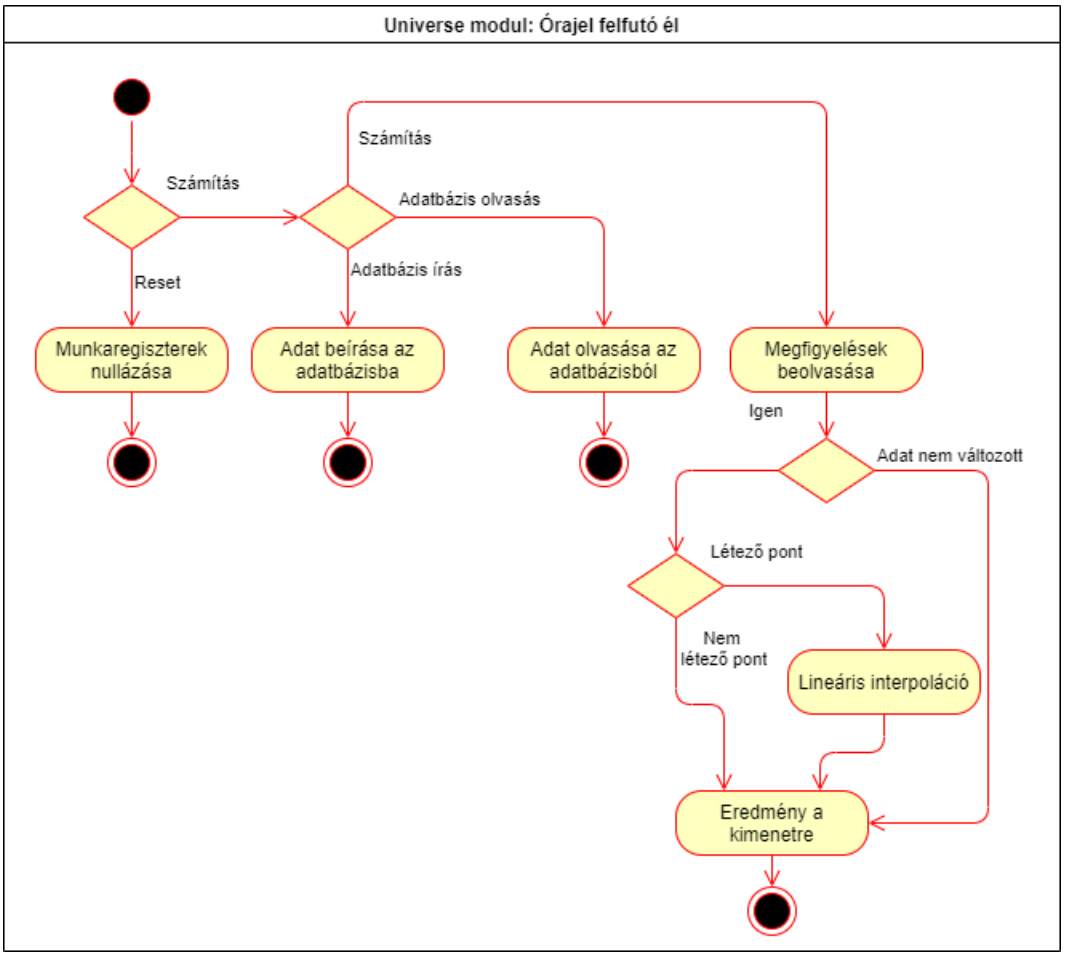
- *universe_data_out*: Az adatbázisból olvasott adatok jelennek meg rajta. Az alapvető működéshez nem szükséges bekötni.
- *observation_out*: A megfigyeléshez tartozó tagsági érték jelenik meg rajta.
- *result_ready*: A számítás elkészültét jelzi. Az alapvető működéshez nem szükséges bekötni.

A modul blokk-rajzát a 47. ábra mutatja:



47. ábra A FIVE *Universe* moduljának blokk-rajza

A modul az órajel felfutó élére reagál, ekkor végzi el a szükséges műveleteket. Reset jelre alaphelyzetbe állítja a munkaregisztereket. Az adatbázis írása alatt nem végez számításokat. A beolvasott megfigyeléseket regiszterbe helyezi miután megvizsgálta, hogy változott-e az előző bemenő adathoz képest. Ha igen, megvizsgálja, hogy létezik-e megfelelő érték az adatbázisban. Ha igen, azt íratja a kimenetre, ha nem elvégzi a lineáris interpolációt és az eredményét kiírja a kimenetre. A változás figyelése az energia felvétel csökkentése miatt került a modul megvalósításába. A működés vázlatát a 48. ábra UML (**U**nified **M**odeling **L**anguage) állapot diagramja tartalmazza.



48. ábra A FIVE Universe moduljának működési vázlata

6.5.3. Az AntecedentDisctance modul

Az AntecedentDistance modul az FBDL rule kulcsszavához tartozó antecedens értékek alapján számítja ki a megfigyelés tagsági értéke és az érintett szabály adott antecedens tagsági értékének távolságát. Az FBDL-ben és az µFRI-ben is az universe által leírt adatok alapján kerülnek meghatározásra. A Verilog megvalósításban minden modul tartalmazza a hozzá tartozó antecedens értéket.

A modulon az alábbi bemenetek találhatóak:

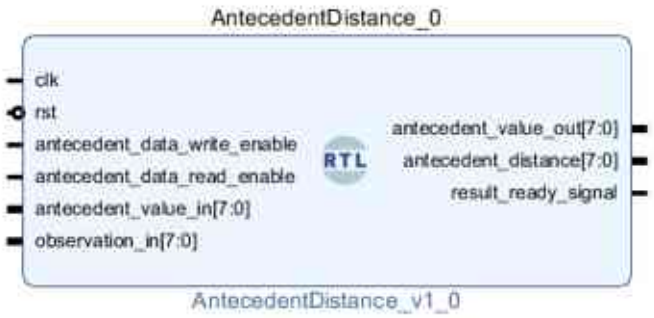
- *clk*: Órajel bemenet.
- *rst*: Reset jel.
- *antecedent_data_write_enable*: Az adatbázis (1 db regiszter) írásának engedélyezése.
- *antecedent_data_read_enable*: Az adatbázis (1 db regiszter) olvasásának engedélyezése.

- *antecedent_value_in*: A számításhoz használt antecedens tagsági értéke.
- *observation_in*: Az *Universe* modulból származó tagsági érték.

A modulon az alábbi kimenetek találhatók:

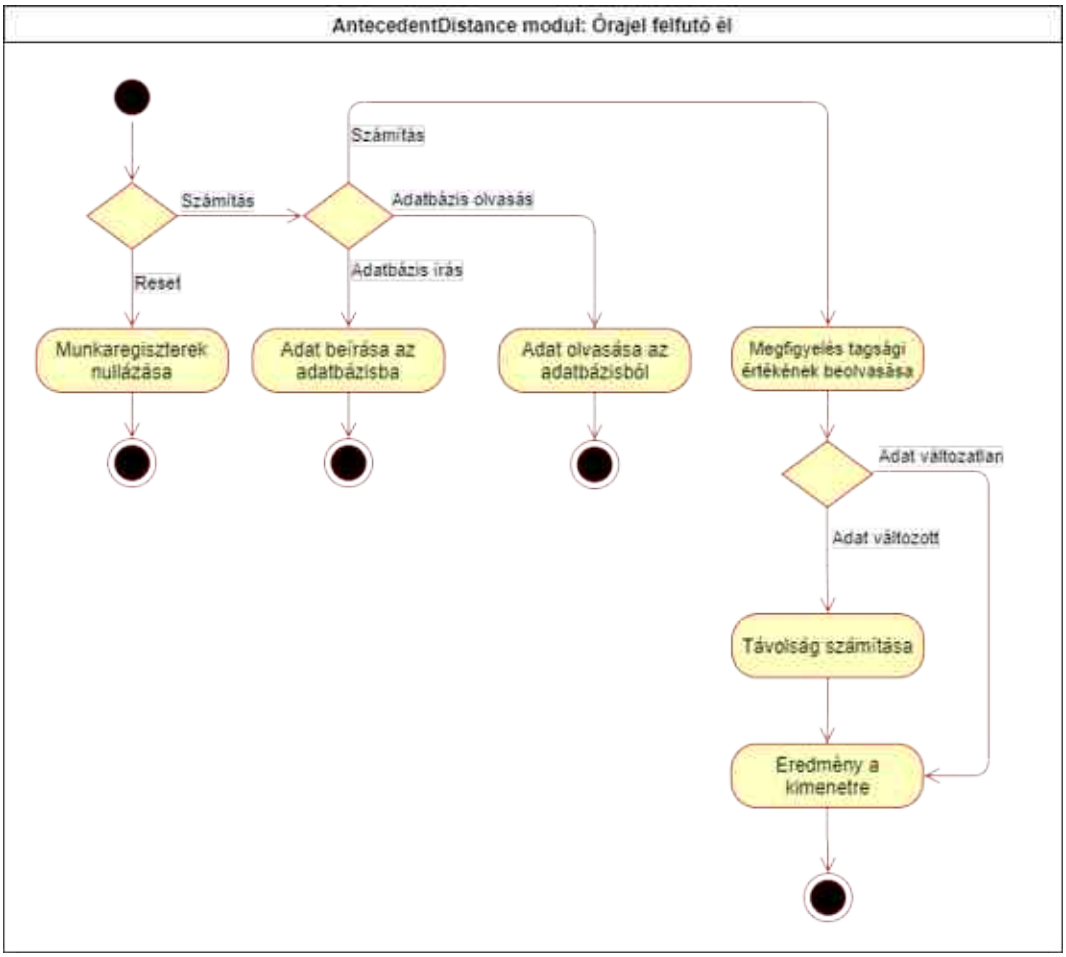
- *antecdent_value_out:* Az adatbázis lekérdezésekor a tárolt antecedens értéke jelenik meg a kimeneten.
- *antecedent_distance:* Az antecedensek közötti távolság eredményé jelenik meg ezen a kimeneten.
- *result_ready_signal:* A számítás végét jelző bit.

A modul blokk-rajza a következő ábrán látható (49. ábra):



49. ábra Az AntecedentDistance modul blokk-rajza

Az órajel felfutó élére reagálva ha a reset jel is aktív, alapállapotba állítja a munka regisztereket. Inaktív reset esetén az előző modulhoz hasonlóan megvizsgálja kell-e írni az adatbázisba, erre az időre a számítások szünetelnek. Ha az olvasás aktív, akkor beolvassa az antecedens tagsági értékét a modulból. A megfigyelések aktuális tagsági értékének olvasása után a modul ellenőrzi, hogy történt-e változás. Ha igen, elvégzi a távolságszámítást és kiírja az eredményét a kimenetre. A működési vázlat rajza a 50. ábra UML diagramján látható.



50. ábra Az AntecedentDistance modul működési vázlata

6.5.4. A RuleDistance modul

A *RuleDistance* modul a hozzá tartozó antecedens távolságok Euklidészi távolságát határozza meg. Ennek a modulnak nincs adatbázis része.

A bemenetei a következők:

- *clk*: Az órajel bemenet.
- *rst*: Reset jel.
- *antecedent_distance_in_X*: a hozzá kapcsolt AntecedentDistance modulokkal összekötendő bemenet. Annyi darabot tartalmaz a modul ebből a bemenetből, ahány antecedense van az adott szabálynak. Az X-tól indulva indexeli a bemeneteket.

Az alábbi kimenettel kapcsolódik a Consequent modulhoz:

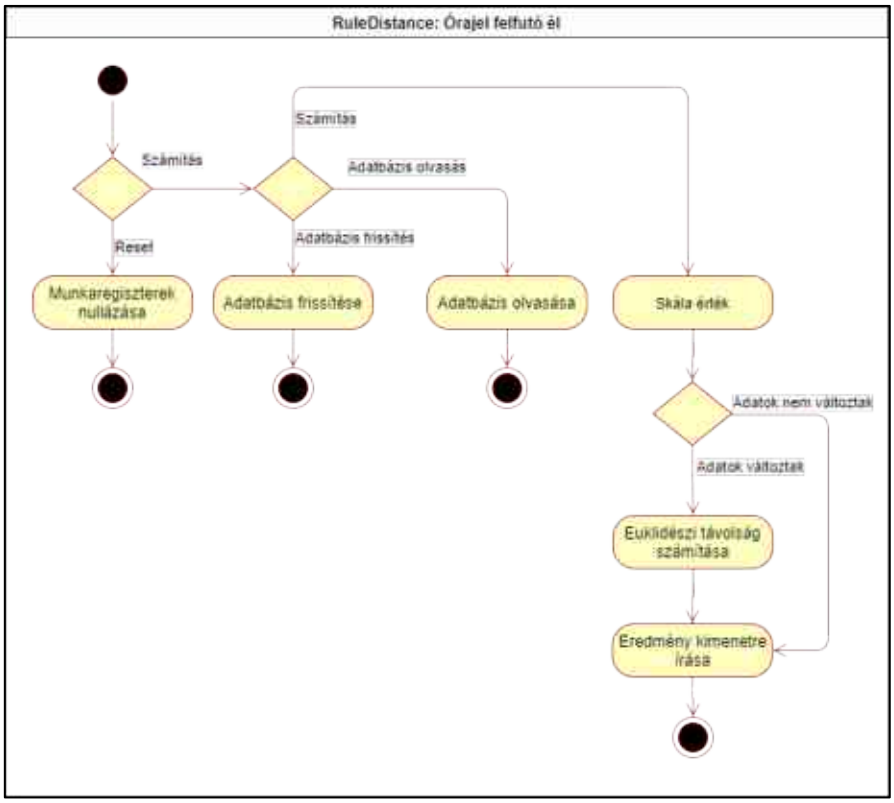
- *rule_distance*: Az Euklidészi távolság eredménye.

- *result_ready*: A számítás befejezését jelzi, nem kötelező bekötni, csak hibakeresési célt szolgál.

A ki-, és bemenetek összefoglaló ábrája (51. ábra):

51. ábra A RuleDistance modul portjai

A működési ábra (52. ábra) a modul feladatainak megfelelően egyszerű. A reset jel ebben az esetben is a munkaregisztereket helyezi alaphelyzetbe. Ha nincs reset a bemeneteken megjelent aktuális távolságokat olvassa be, ellenőrzi, hogy történt-e változás. Ha történt, elvégzi a négyzetre emelést és az összeadás után a gyökvonást, majd kiírja az eredményt a kimenetre.



52. ábra A *RuleDistance* modul működési vázlata

6.5.5. A Consequent modul

A *Consequent* modul állítja elő egy szabálybázis döntését, a kimeneti eredményt. Az adatbázisa az FBDL *rule* részének első adatát tartalmazza, amely az FBDL-ben és az µFRI-ben is a kimeneti *universe*-ből veszi az értékeket. Ebben az esetben a konklúziókat a modul külön tárolja regiszterben, amelyek módosíthatóak is.

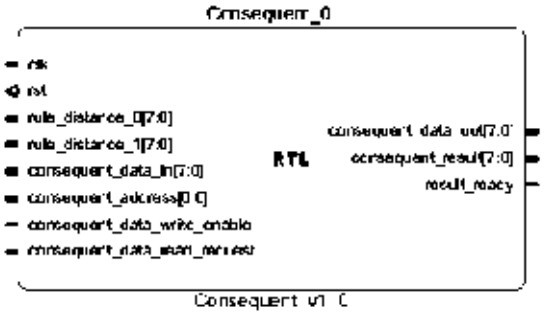
A portokat is tartalmazó blokk-rajz szemlélteti a kapcsolódását a környezetével (53. ábra).

Bemenetei a következők:

- *clk*: Órajel
- *rst*: Reset jel
- *rule_distance_X*: Kapcsolódási pont a RuleDistance modullal. Az X a bemeneteket különbözteti meg 0-tól számozva. Annyi darab van ebből a portból ahány szabály van a szabálybázisban.
- *consequent_data_in*: Az adatbázis frissítéséhez adatbemenet.
- *consequent_address*: A frissíteni vagy olvasni kívánt konzekvens sorszáma 0-tól számozva.
- *consequent_data_write_enable*: Az adatbázis írását engedélyező port.
- *consequent_data_read_request*: Az adatbázis olvasását engedélyező port.

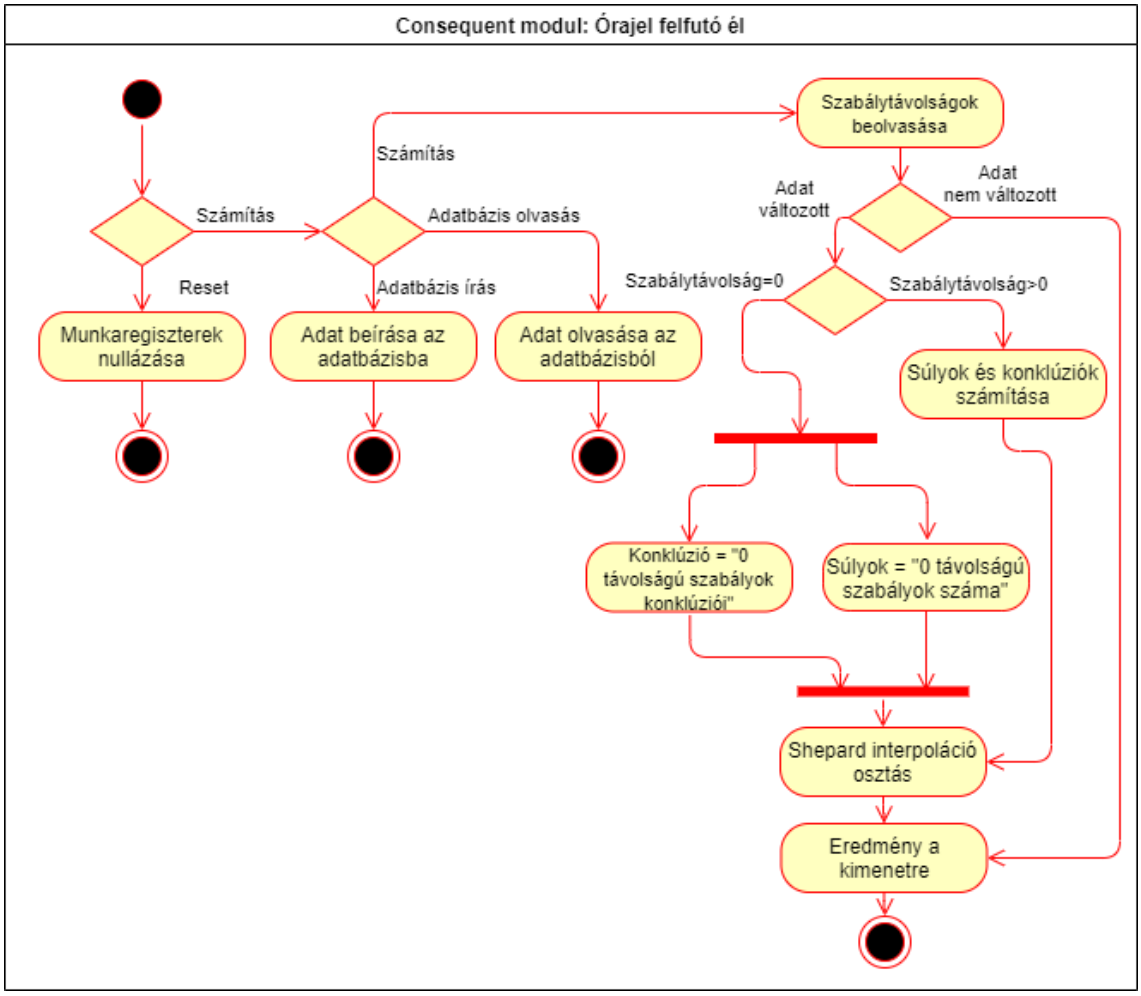
A kimenetei a következők:

- *consequent_data_out*: Olvasáskor itt jelenik meg a kiolvasott adat.
- *consequent_result*: A szabálybázis eredménye jelenik meg ezen a porton.
- *result_ready*: A számítás végét jelző port, főként hibakeresési célokat szolgál.



53. ábra A *Consequent* modul blokk-rajza

A *Consequent* modul működése hasonló az előzőekhez a reset és adatbázist eljárást tekintve illetve a bemenet változását is ellenőrzi. A bemenet változása esetén hajtja végre a számítást. Megvizsgálja a szabálytávolságokat. Amikor a szabálytávolság 0, akkor a megfigyeléshez tartozik szabály, ilyenkor a végeredményt a találatot ért szabályokhoz tartozó konzekvensek távolsággal súlyozott átlaga adja. Ez a megoldás a FIVE módszer része. Ha nincs 0 szabálytávolság az eredeti számításnak megfelelően a Shepard interpolációt hajtja végre a modul. A működését grafikusan az 54. ábra foglalja össze.



54. ábra A Consequent modul működési rajza

6.6. A mintarendszer felépítése

Az egységek teszteléséhez egy egyszerű 2 szabályból, szabályonként 2 antecedensből álló szabálybázist használtam. Az FBDL szerinti *universe* 3 elemet tartalmaz. Az FPGA tervnek megfelelő szabálybázis felépítése a következő:

```
universe "antecdent0"
    "p0" 0 0
    "p1" 127 127
    "p2" 255 255
end

universe "antecdent1"
    "p0" 0 0
    "p1" 127 127
    "p2" 255 255
end

universe "consequent"
    "p0" 0 0
    "p1" 255 255
end

rulebase "consequent"
    rule
      "p0" when "antecdent0" is "p0" and "antecdent1" is "p0"
    end
    rule
      "p1" when "antecdent0" is "p2" and "antecdent1" is "p2"
    end
end
```

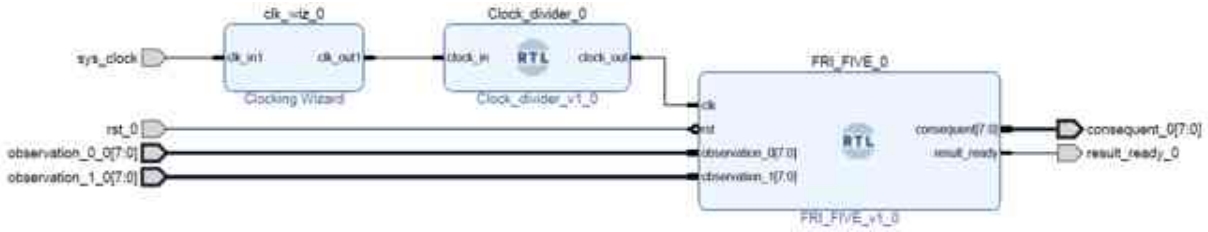
A mintarendszer 8 bites bitszélességgel működik. Ennek megfelelően a szabálybázisban a bemeneti és kimeneti értékek is [0;255] tartományban vannak megadva. A mintául használt szabálybázis felépítése formális, csak a rendszer előállítására készült. Az *antecedent0* és *antecedent1* a megfigyelésekhez tartozó univerzumok, a *consequent* pedig a döntéshez tartozó univerzum. A *p0*, *p1* és *p2* a nyelvi értékek, amelyeknek az első paramétere az érzékelőtől érkező adat vagy *consequent* esetén a kimenő érték a 8 bites felbontásnak megfelelően [0;255] tartományon. A nyelvi értékek második paramétere szintén a 8 bites [0;255] tartományon az első paraméterhez kapcsolódó skálafüggvény értéke.

Az FBDL-ből a következő táblázatban (4. táblázat) összefoglalt megfeleltetéssel állítható elő a hardveres rendszer. Az FBDL egyes részei tartalmazzák a számítási blokkok szükséges paramétereit és összeköttetéseit.

4. táblázat Az FBDL elemek megfeleltetése a FIVE FPGA moduloknak

FBDL elem	FIVE FPGA modul	Átvitt paraméterek	Összeköttetés
universe nyelvi értékekkel	Universe modul	A nyelvi értékek darabszáma és paraméterei	A blokk bemenete, érzékelők felé. Kimenet a hivatkozott AntecedentDistance felé
„antecedent0” is „p0” formájú predikátum	AntecedentDistance modul	A predikátumban megjelölt nyelvi érték skálafüggvény értéke (második paraméter)	Bemenet a kapcsolódó Universe felé. Kimenet a RuleDistance felé
A szabályt alkotó predikátumok	RuleDistance modul	Nincs	Bemenetei a kapcsolódó AntecedentDistance felé. Kimenet a Consequent felé.
Rulebase kimeneti universe és a rule első paramétere	Consequent modul	Az rulebaseben megadott universe nyelvi elemeinek mindkét paramétere	Bemenetei a RuleDistance modulok felé. Kimenete a rendszer kimenete

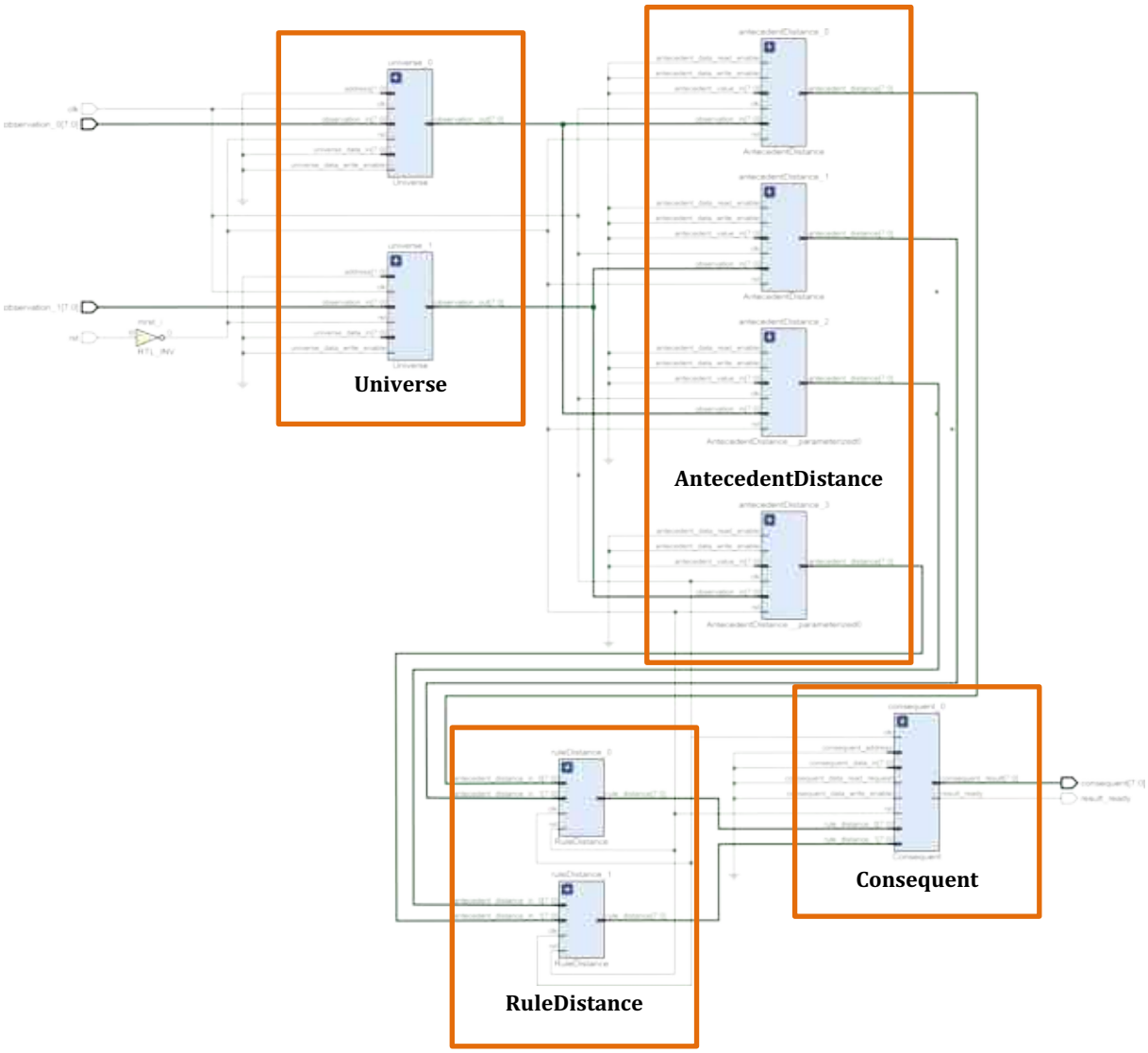
A teljes működtető rendszer a Vivado 2018.1-es verzió Block Design ábrája a 55. ábra szerinti. Ezen az ábrán a FIVE mellett az órajelet beállító IP-k kaptak helyet. A Xilinx *Clocking Wizard* 100 MHz-es órajelet állít elő. A *Clocking Wizard* által legkisebb előállítható órajel 6,25 MHz, ennél kisebb órajel előállításához szükséges a *Clock Divider*. A FIVE kimenetei és bemenetei külső portokra vannak kötve.



55. ábra A mintában megadott rendszer diagramja

A mintarendszer nem tartalmazza a kommunikációs blokkot, csak a számítást végző részeket. A fenti szabálybázishoz tartozó kapcsolási rajzot az 56. ábra szemlélteti. A szabadon maradt portok az inaktív állapotnak megfelelő logikai jelszintre vannak bekötve. A sötétzöld vonalak az adatbuszokat jelölik, míg a világoszöld vonalak az 1 bites vezetékeket. A Vivado a be nem kötött részeket, amelyek az adatbázis frissítéséhez tartoznak nem jeleníti meg. A *reset* bemeneten található egy invertáló kapu a megfelelő reset logikai szint előállításához. A nagyobb méretű ábrát a Melléklet tartalmazza.

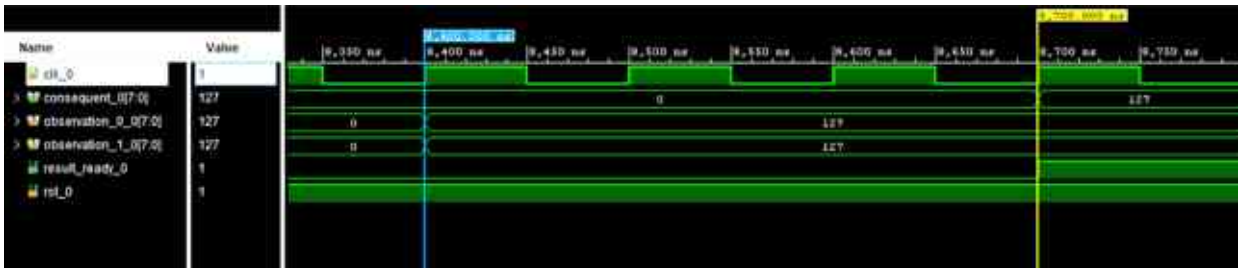
A számítást a pipeline kialakítása miatt bontottam több részre. Így a végeredmény 4 órajel alatt jelenik meg a kimeneten.



56. ábra A FIVE elvi kapcsolási rajza

6.7. Szimulációs eredmények

A rendszer szimulációját a Vivado Design Suite 2018.1 verziójával, XSim programmal végeztem. A 57. ábra a teljes rendszer szimulációja, amelyen látható a 4 órajelnyi késleltetés. A szimulációhoz nincs bekötve a Clock Divider és a Clock Wizard (ezek az IP-k a rendszer alapórajelét osztják a kívánt értékre, a Clock Wizard 6,25 MHz minimális órajelet képes előállítani ezt szükséges tovább osztani a Clock Diviter IP-vel a szükséges 4 MHz-re), hogy a közvetlenül a működtető órajelek láthatók legyenek. A szimuláció periódus ideje 100 ns. Az ábrázolt tartományban a bemenő adatok (megfigyelések) változása utáni állapotot vizsgáltam. A két 127-es értékre válaszul a rendszernek is 127-et kell adnia, amelyet a 4. órajelre előállított. Az kék jelzésnél az első felfutó élnél hajtódik végre az Universe számítása. A következő felfutó élre az AntecedentDistance modul számításai, ezt követő felfutó élre a RuleDistance számításai és végül a Consequent modul számításai és jelenik meg az eredménye a kimeneten.



57. ábra Késletetés szimulációja

A pipeline működést az 58. ábra szemlélteti. Az órajelenként adott új adat az előzőleg megállapított 4 órajel késleltetéssel jelenik meg a kimeneten. Mivel az utolsó adat után nem történik változás, amely elindítaná a számítást így az utolsó előtti érték marad meg. A folyamatos működéskor ez a jelenség nem áll fenn. A *consequent_0* a kimenetet jelöli, az *observation_x_0* a megfigyelés, bemenő adat portja, A *result_ready_0* a kész állapotot jelzi, az *rst_0* a reset jel és a *clk_0* az órajel: 100 ns periódusidővel.

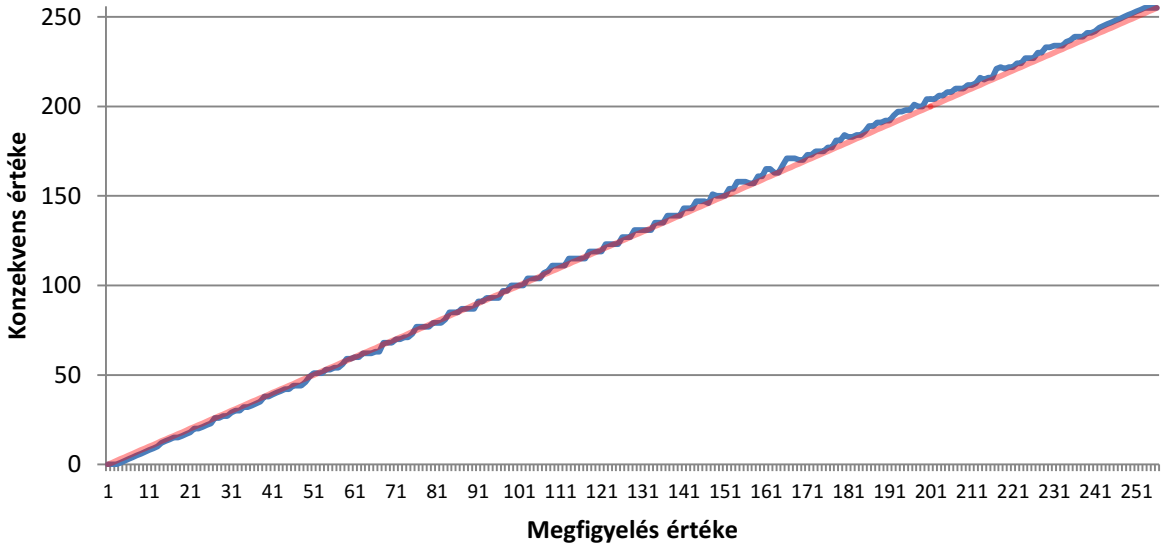
Megvizsgáltam a számítás linearitási hibáját a szoftveres változathoz képest. A

58. ábra Pipeline működést szemléltető szimuláció

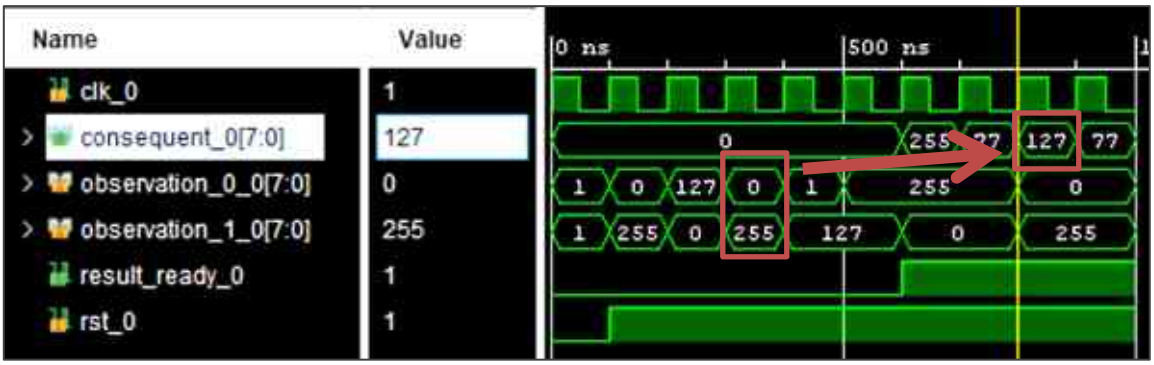
tesztadatok mindkét bemeneten 0-255-ig egész számok. Ekkor az eredmény ennél a szabálybázisnál a bemenetre adott érték. A végeredményt korrigáltam az áramkör késletetésének megfelelően. Ez könnyen ábrázolható eredményt szolgáltat (lásd, 59. ábra).

Az eredmények az egész számos számítás miatt tartalmaznak eltérést az ideálishoz képest, de a rendszer működése szempontjából elhanyagolható mértékben. Az osztás művelet esetén 8 bites eltolást alkalmaztam, kevesebb eltolás esetén jelentős számítási hiba alakult ki. Nagyobb eltolás nem hozott jelentős javulást. A 8 bites eltolás 256-al való szorzásnak felel meg 10-es számrendszerben. Az eredményeket a 59. ábra grafikonjáról lehet leolvasni. A szimuláció alapján a számítási hiba legnagyobb abszolút értéke 6, amely 2,34 %-nak felel meg ezen a tartományon. Az elvárt értékeket a FIVE C nyelvű megvalósítása a µFRI könyvtár számította.

Elvégeztem a teljes értelmezési tartományra a számítást a szimulátorban. A 255, 0 kombinációnál hibás értéket adott az alkatrész, mivel nem érzékelte az egyik bemenet változását. Ez a jelenség csak akkor fordul elő, ha reset után rögtön van 0 értékű megfigyelés. További egyedi tesztek alkalmával megfelelő értéket adott. A teljes értelmezési tartomány vizsgálatakor áll fenn a jelenség. A felületet ettől eltekintve



59. ábra A linearitás vizsgálata. A piros vonal az elvárt értéket, a kék vonal a számított értéket jelöli. Az elvárt értéket a µFRI C nyelvű megvalósítása számolta.

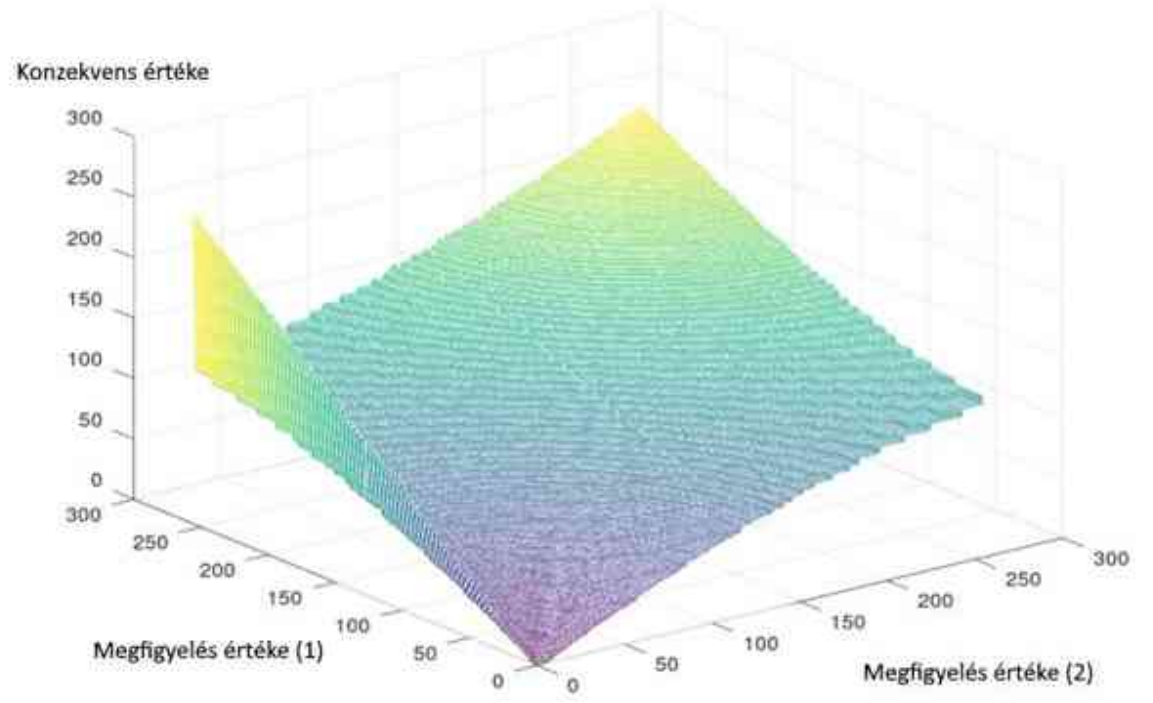


60. ábra A helyes működés szimulációja

megfelelően fedi le. Az egyedi teszt eredményét az 60. ábra tartalmazza, ahol a piros keret jelzi a kérdéses megfigyelést és a hozzá tartozó kimenetet.

A teljes értelmezési tartomány szimulációját az 61. ábra mutatja, ahol a függőleges tengely a szabálybázis kimenetét, a vízszintes tengelyek a megfigyeléseket tartalmazzák.

A szimulációs vizsgálatokkal az adatbázist is tartalmazó elemek (*Universe*, *AntecedentDistance*, *Consequent*) elemeknek az adatbázis frissítéséhez és olvasásához kapcsolódó viselkedéseket is ellenőriztem. Az említett funkciók a hangolás lehetőségét biztosítják.



61. ábra Szimulációs eredmények a teljes értelmezési tartományra

6.8. Szintézis és implementáció

A szintézis és implementáció végrehajtásával azt vizsgáltam, mekkora az erőforrás igénye az FRI_FIVE modulnak illetve mekkora a legnagyobb működési frekvenciája, amivel a rendszer még működhet? A szintézishez és implementációhoz az intézetben is elérhető Xilinx SoC áramköröket választottam:

- Zybo,
- Nexys4 DDR.

A kapott eredményeket a 5. táblázat foglalja össze. A különböző SoC-k implementációi között nincs jelentős különbség. Nagyobb eltérés egyedül a késleltetési időben (WNS, Worst Negative Slack) látható a Zybo SoC javára. A táblázat nem tartalmazz az egységesen 0 értékű oszlopokat. Az jelenlegi szabálybázist megvalósító áramkör teljesítményfelvétele körülbelül 0,49 W.

5. táblázat A szintézis és implementáció eredményét összefoglaló táblázat. WNS – Worst Negative Slack, legrosszabb negatív késleltetés; WHS – Worst Hold Slack, legrosszabb tartási idő; Total Power – teljes teljesítmény; LUT – Look Up Tables – keresőtáblák száma; FF – Flip Flops, flip-floppok darab száma; DSP – Digital Signal Processors; DSP-k száma; IO- bemenetek és kimenetek száma; BUFG, Bufferek száma

Stratégia neve	SoC típusa	WNS (ns)	WHS (ns)	Total Power (W)	LUT	FF	DSP	IO	BUFG
Vivado Synthesis Defaults	Nexys4 DDR				8277	174	6	27	2
Vivado Synthesis Defaults	xc7a100tcsg324-3	27,58	0,148	0,4992	8237	173	6	27	2
Vivado Synthesis Defaults	Zybo				8277	174	6	27	2
Vivado Implementation Defaults	xc7z010clg400-1	120,29	0,152	0,4924	8241	173	6	27	2
Performance_NetDelay_high	xc7z010clg400-1	120,53	0,152	0,4924	8241	173	6	27	2
Power_DefaultOpt	xc7z010clg400-1	120,75	0,155	0,4923	8245	173	6	27	2
Flow_RunPostRoutePhysOpt	xc7z010clg400-1	120,29	0,152	0,4924	8241	173	6	27	2
Flow_AreaOptimized_high	Zybo				8010	172	10	27	2
Vivado Implementation Defaults	xc7z010clg400-1	120,50	0,156	0,4855	7983	171	10	27	2
Performance_NetDelay_high	xc7z010clg400-1	120,45	0,156	0,4855	7983	171	10	27	2
Power_DefaultOpt	xc7z010clg400-1	120,34	0,156	0,4834	7977	171	10	27	2
Flow_RunPostRoutePhysOpt	xc7z010clg400-1	120,50	0,156	0,4855	7983	171	10	27	2
Flow_RuntimeOptimized	Zybo				8331	177	6	27	2
Vivado Implementation Defaults	xc7z010clg400-1	120,47	0,170	0,4894	8257	173	6	27	2
Performance_NetDelay_high	xc7z010clg400-1	120,47	0,170	0,4894	8257	173	6	27	2
Power_DefaultOpt	xc7z010clg400-1	121,08	0,152	0,4896	8253	173	6	27	2
Flow_RunPostRoutePhysOpt	xc7z010clg400-1	120,47	0,170	0,4894	8257	173	6	27	2

A felhasznált erőforrások a szabályszám és az antecedensek számának növekedésével növekszik. A *Universe* részegység darabszáma a megfigyelések számától függ, az elfoglalt méretét befolyásolja, hogy hány nyelvi értéket tartalmaz, mennyi elemmel közelíti a kívánt bemeneti függvényt. Az *AntecedentDistance* egység darabszáma a szabály antecedenseinek a számától függ. A *RuleDistance* komponensek darabszáma a szabálybázisban lévő szabályok számától függ, míg a bemeneteinek a száma a szabályt alkotó antecedensek számától függ. A *Consequent* elemből szabálybázisonként egy darab van, a bemeneteinek a számát a szabálybázist alkotó szabályok határozzák meg. A szabályok száma befolyásolja, hogy mennyi adatot kell tárolnia a konzekvensek értékeinek megfelelően.

A számított biztonságos működési frekvencia 4 MHz. A fenti táblázatban megjelenő WNS érték is erre a frekvenciára vonatkozik. A kiindulási 100 MHz esetén nem volt számottevő különbség a kártyák WNS értékei között, a lassabb órajel esetén jelent meg a nagyobb eltérés.

6.9. Összevetés a Vivado High Level Synthesis megvalósításával

A Vivado High Level Synthesis (HLS) eszköze lehetővé teszi, hogy C/C++/SystemC nyelveken leírt algoritmusok Verilog vagy VHDL megvalósítását. Ez az eszköz segítséget nyújt, amikor bonyolult, a hardverleíró nyelvekre nehezen átültethető algoritmusok FPGA-s gyorsításakor. A kész IP-k tetszőleges csatolási felülettel adott programozható logikai áramkörre specializáltan generálható a fent említett magas szintű nyelvekről [56], [57], [58]és [59].

Az algoritmusok többsége nem generálható egyből a HLS eszközzel. A hardveres megvalósítás miatt a dinamikusan változó részeket statikussá kell alakítani illetve az állapotgépeket nem tudja megfelelően kezelni. A dinamikusan változó részek, a változó méretű tömbök, amelyeknek meg kell adni a felső méretkorlátját illetve azoknak a ciklusoknak, amelyeknek nincs megadva vagy változik az iteráció száma szintén meg kell adni a maximális iterációk számát. Az ilyen módon átalakított programkódot tovább lehet finomítani különféle direktívák alkalmazásával. Ezen korlátok, ajánlások az [60] dokumentációban érhetők el.

A µFRI könyvtárat 8 bites egész számok használatára alakítottam, és megadtam az egyes ciklusok maximális iterációk számát. Az eredmények alapján a HLS 90-110 órajel közötti eltérést ad ugyan azon szabálybázis számítására. Az Verilog megvalósítással összevetett eredményeket az 6. táblázat foglalja össze.

6. táblázat Időzítés és erőforrás összehasonlító táblázat

µFRI	Órajel	LUT	FF	DSP
Saját	4	~8000	~173	~6
HLS	90-110	~8600	~5300	~7

6.10. Tézis

A FIVE IP könyvtár moduljai alkalmasak a FIVE FRI FPGA-n való FPGA erőforrás hatékony implementációjára. A könyvtár felhasználásával a létrehozott rendszer képes a tudásbázis rendszerleállás nélküli dinamikus megváltoztatására, beágyazott rendszerbeli adaptív alkalmazások kialakítására.

Magyarázat:

A dinamikus megváltoztatást az FPGA architektúra teszi lehetővé. Az előre ismert viselkedésekre több szabálybázis alkotható és implementálható. Amennyiben módosul a környezet abban az esetben a hardver módosítása szükséges (Adaptív konfigurálás), amely egyben a szabálybázis módosítását jelenti. A rendszer a környezetnek megfelelő kész szabálybázist választja és konfigurálja az FPGA-n (Parciális rekonfiguráció). Az irányítási algoritmus tartalmazza a rekonfiguráció módosításához szükséges állapotgépet.

6.11. Új eredmények összefoglalása

A FIVE módszer FPGA-n történő használatához elkészítettem egy vázat Verilog hardver leíró nyelven, amely alkalmas arra, hogy a FBDL (**F**uzzy **B**ehavior **D**escroption **L**anguage) alapján generálható egy hardver elem, amely a FIVE módszer számítását gyorsítja. Alkalmazható önállóan vagy CPU-val társítva gyorsító segédáramkörként. A FRI_FIVE modulok összekapcsolásával létrehozható összetett viselkedést megvalósító fuzzy viselkedés automata.

A FIVE IP struktúrája lehetővé teszi a Xilinx Adaptive Platformon történő megvalósítását is, mint egy szoftveres környezet hardveres gyorsító egységeként. Ekkor

számítási egységenként vagy a teljes FIVE IP AXI vagy PCIe sínrendszeren csatlakozik a processzoros rendszerhez és a szoftveres modul az adatgyűjtést és a vezérlési feladatot látja el.

Jelenleg nincs általam ismert FPGA-n megvalósított fuzzy interpolációs módszer vagy adaptív fuzzy interpolációs módszer. Klasszikus fuzzy illetve neurofuzzy módszerek FPGA megvalósításai léteznek.

6.12. Gyakorlati alkalmazás

A FIVE IP segítségével viselkedés alapú irányítás valósítható meg áramkör szinten FPGA segítségével. Az egyszerű csataló felületnek köszönhetően beágyazott mikrovezérlőhöz vagy AXI sínen keresztül alkalmazás processzorhoz is csatolható. Közvetlen felhasználásra is lehetőség van az adatok megfelelő skálázásával.

Az irányítási feladatokon kívül viselkedések hangolását is gyorsíthatja a szoftveres megvalósításhoz képest az egyes iterációk gyorsabb végrehajtásával. Viszont a szabályszám a FIVE IP-ben nem növelhető tetszőlegesen csak a hardver által megadott kereteken/korlátokon belül. A paraméterek (univerzumok értékei, megfigyelések és konzekvensek értékei) korlátlanul átírhatóak, a paraméterek átírása egy órajelet igényel, a következő órajelnél már az új paraméterrel elvégzett eredmény jelenik meg a kimeneten.

6.13. Összegzés

Elkészítettem a FIVE implementációját IP formájában Verilog nyelven, amely adatstruktúrája lehetővé teszi a szabálybázis hangolását valós időben.

Kapcsolódó cikkek: [S1], [S2], [S8] és [S13].

7. A FIVE módszer, mint osztályozó eljárás alkalmazása

A megfigyelésekből származó adatok osztályozása, címkézése fontos azok további feldolgozása, mobil robot vagy mesterséges intelligencia esetén a döntési fa előállításában. Az osztályozó eljárások felügyelt válogatási folyamatok, amelyek statisztikai vagy szakértői adatbázist vesznek alapul a megfigyelések címkézésére. A hagyományos osztályozó eljárások általában crisp halmazokba válogatják a bemenő adatokat. A fuzzy osztályozóval azonban az is megadható, hogy az egyes adatok milyen mértékben tartoznak az egyes osztályokhoz. Az ilyen módon akár szakértői adatbázis alapján működő osztályozó kimenetét akár több fuzzy alapú viselkedés leírás használhatja fel. A gyakorlatban ez finomabb átmenetet képez majd a viselkedések közötti váltás esetén.

A fuzzy osztályozó módszer felhasználható olyan mintafelismerő eljárásként, amely arról is számot ad, hogy a legjobban illeszkedő mintán kívül mely mintázatokra illeszkedik még bemenő adatsor.

7.1. Célkitűzés

A FIVE eljárás szabály alapú osztályozóként történő alkalmazása a μFRI könyvtár használatával. Az elért eredményeket összevetem Naiv Bayes osztályozóval.

Az osztályozó vizsgálatában egy labirintusban közlekedő robot számára számítja ki, hol találhatóak a falak a robot aktuális tartózkodási pontjában.

7.2. Osztályozó eljárások

Többfajta osztályozó eljárás létezik különféle sajátosságokkal, különféle helyzetekhez igazodva. Az osztályozó eljárások alapvető fogalmai a következők:

- **Osztályozó**: Az algoritmus, amely a megfigyeléseket a megfelelő kategóriába rendezi.
- **Osztályozó modell**: Tanító minta alapján létrehozott adathalmaz, amelyet az osztályozó algoritmus használ fel a kimeneti kategóriák előállítására.
- **Jellemző**: A megfigyelés egyedi mérhető tulajdonsága.

- **Bináris osztályozó**: Olyan osztályozó eljárás, amelynek két kimenete van. Például: hideg/meleg, igaz/hamis.
- **Több osztályba soroló osztályozó**: Kettőnél több osztályba sorolja a megfigyeléseket. Egy megfigyelés egy időben csak egy osztályban lehet.
- **Többszörös címkéző osztályozó**: Egy megfigyelés egy időben több osztályba is sorolható.

Az osztályozók működése hasonló lépéssorozattal írhatók le típustól függetlenül. Ezek a lépések az alábbiak:

- Az osztályozó kezdeti értékeinek beállítása.
- Az osztályozó tanítása, ehhez szükséges valamilyen tanító minta, amely alapján elkészítheti az osztályozó modellt.
- Osztályozási folyamat, amikor a betanított osztályozó a felépített modell alapján egy ismeretlen adathalmaz elemeit felcímkézi.
- Az osztályozó modell értékelése, hogy milyen találati aránnyal dolgozott.

Az osztályozó algoritmusoknak 8 fajtája ismert. Az osztályozó algoritmusok fajtái a következők:

- Lineáris osztályozók,
- Support Vector Machine (SVM),
- Kvadratikus osztályozók,
- Kernel becslő,
- Döntési fák,
- Neurális hálózatok,
- Learning vector quantization,
- Szabály alapú osztályozók.

Ezek közül a lineáris Naiv Bayes osztályozót fogom részletesen bemutatni. A szabály alapú fuzzy osztályozóval a Bayes osztályozót hasonlítom össze egy gyakorlati esetben. A fuzzy szabálybázis, a FIVE interpolációs módszer bizonyos megkötésekkel hasonló a Bayes osztályozóhoz. A következő részek a Bayes és a szabály alapú osztályozót mutatják be. [61]

7.2.1. Naiv Bayes osztályozó

A Naiv Bayes osztályozó lineáris osztályozók családjába tartozik, valószínűségi alapú osztályozó, amely a Bayes tételen (35) alapul és függetlenséget feltételez a jellemzők között. A függetlenség azt jelenti, hogy egy adott jellemző értéke nincs semmilyen kapcsolatban más jellemzővel. A Naiv Bayes osztályozó tetszőleges számú független változót képes kezelni. Egy-egy esetet képes véges számú osztály valamelyikébe besorolni. Abba az osztályba, amelybe a legnagyobb valószínűséggel tartozik. A Bayes osztályozó hatékonyan tanítható felügyelt tanulási környezetben. A neurális hálókhoz képest kisebb mennyiségű adattal is hatékonyan tanítható osztályozó.

A Bayes osztályozó a következő (35) összefüggés szerint számítja ki a megfigyelésekre a legvalószínűbb címke értékét:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{35}$$

ahol a P(c|x) a c osztály valószínűsége az x attribútumok szerint. A P(c) a keresett osztály előzetes előfordulási valószínűsége a rendszerben. P(x|c) a likelihood becslés az adatokból a c osztályhoz x attribútum szerint, ahol az x a megfigyelés. P(x) az x megfigyelés valószínűsége. A Naiv Bayes esetén ez elhagyható, így az összefüggés a következő:

$$P(c|\boldsymbol{X}) = P(x_1|c)P(x_2|c) \dots P(x_n|c)P(c) \tag{36}$$

ahol **X** a megfigyelések vektora (például jellemzők, érzékelők mért értéke stb.). P(c|**X**) a c osztály valószínűsége az **X** megfigyelések alapján. $P(x_n|c)$ az egyes megfigyelésekhez tartozó valószínűség a c osztályra nézve. P(c) a c osztály valószínűsége a megfigyelt környezetben.

A Bayes osztályozó alacsony számítási igényű osztályozó. Legfőbb hátránya a szabály alapú osztályozóhoz képest nagy mennyiségű tanító minta összegyűjtése miatt jelentkezik. Kimenete egy valószínűség a [0;1] intervallumban. Az osztályozó minden ismert osztály valószínűségét előállítja, ezek közül általában a legnagyobb valószínűségűt vagy egy küszöbérték feletti legnagyobb lesz a döntés. [62], [61], [63] és [64]

7.2.2. Szabály alapú osztályozók és fuzzy osztályozók

A szabály alapú osztályozónak nevezhető bármilyen osztályozó, amely „HA…AKKOR” sémájú szabályokat használ az osztály kiválasztására. A szabály alapú osztályozók esetén az antecedens elemek logikai ÉS kapcsolatban állnak egymással. A szabálybázisnak a lehetséges bemeneti variációkat a lehető legjobban le kell fednie és ellentmondásmentesnek kell lennie. Mivel a legjobb fedés nem feltétlenül biztosítható, ezért úgy kell kialakítani az osztályozót, hogy egy olyan eset, amely nem sorolható egyértelműen a lehetséges osztályokba se okozzon hibás működést, jelezze a nem osztályozható adatokat. Ezáltal ezek tanulás vagy kézi beavatkozás útján megfelelő osztályba helyezhetők. Tehát a szabály alapú osztályozók esetén hasonló probléma lép fel, mint a klasszikus fuzzy esetén, bizonyos megfigyelés kombinációkra előfordulhat, hogy nem tud döntést produkálni a szabályrendszer.

Szabály alapú osztályozó kialakítható fuzzy logikával is. Tehát megkapjuk, hogy melyik osztály milyen mértékben illeszkedik a megfigyelésre. Mivel előfordulhat, hogy több szabály is illeszkedést mutat, ezért különféle már ismert eljárásokkal lehet szűrni és akár crisp eredményt előállítani egy határérték beállításával vagy a legjobb illeszkedés figyelembe vételével. Azonban ezzel a fuzzy jelleget lehet elveszíteni. A különböző illeszkedési mértékek többlet információkat szolgáltatnak, amelyeket felhasználva finomítható a megfigyelésekre adott válasz reakció.

A klasszikus fuzzy fedési problémáját fuzzy szabály interpolációs módszerekkel lehet orvosolni, például a FIVE módszerrel, amelyet a fejezetben vizsgálni fogok, hogyan használható osztályozó eljárásként. Kihasználva azt a lehetőséget, hogy a FIVE szabálybázisait szakértői adatokkal töltöm fel. [10], [64], [65], [66], [67], [68]

7.3. A fuzzy osztályozó összevetése a Bayes osztályozóval

A FIVE módszert, mint fuzzy osztályozó eljárás egy gyakorlati példán keresztül hasonlítom össze a Bayes osztályozóval. A példában egy labirintusban közlekedő mobil robot fogja a környezetében érzékelni az őt körül vevő falakat. A falak helyzete fontos a robot körül, mert a haladási iránytól és a falak helyzetétől függően kell megválasztania az iránya mellett a sebességét is miközben rögzíti az adott cellában érzékelt mintázatot a térképére.

Az érzékelésre 4 darab infravörös diódapár szolgál. Ezek eltérő kimeneti jele eltérő lehet és a falaktól körülbelül egyenlő távolság tartására szükség van a nyers mért értékekre egy egyszerű a fal jelenlétét jelző bináris jelhez képest. Azonban a térképezésnél már elegendő a bináris jel, amelyet az osztályozó fog előállítani az érzékelők adataiból.

Mivel most a térképezéshez szükséges jel előállítása az elsődleges így a fuzzy osztályozó kimenetét is egy határérték feletti értékeknél veszem figyelembe. A Bayes osztályozó esetén a legnagyobb valószínűségű mintázat érvényes.

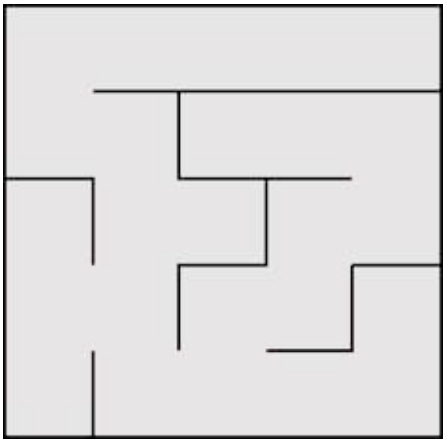
A Naiv-Bayes osztályozó kiválasztásának oka az egyszerű számíthatóság és a statisztikai, tapasztalati úton történő taníthatóság. Utóbbi jellemző a szabály alapú osztályozóra, a szakértői szabályrendszer létrehozásakor.

7.3.1. A mobil robot és a pálya felépítése

A mobil robot egy „Half-size Micromouse" versenyre készült differenciál hajtásos robot 4 infravörös érzékelő párral, a kerekein út jeladóval és egy gyorsulásérzékelővel. A robotot (62. ábra) a Budapesti Műszaki Egyetem, Mechatronika, Optika és Gépészeti Informatika Tanszéke készítette. Részletes leírása az alábbi Tudományos Diákköri Dolgozatban olvasható: [69], ahol a pálya részletes leírása is megtalálható.

Mivel az eredeti pálya túl nagy, egy 5x5 darab cellából álló változatot használtam, amely tartalmazza a legtöbb jellemző mintázatot. A cellák mérete 90x90 mm, a falak magassága 25 mm. A labirintus elrendezését a 63. ábra szemlélteti.

62. ábra A mobil robot vázlata



63. ábra A teszteléshez használt labirintus vázlata

7.3.2. Adatgyűjtés a környezetről, a Bayes osztályozó tanító mintái

A labirintusban 7 fajta alakzat fedezhető fel a falakból. A robot minden mintázatnál előre-hátra mozgással adatot gyűjt, ezzel elég nagy mennyiségű adatot gyűjthet a Bayes-osztályozó tanítására. Az alábbi mintázatoknál gyűjtött a robot információt a környezetéről, ahol feltüntettem az adott mintázat relatív gyakoriság alapján becsült valószínűségét is a labirintusban:

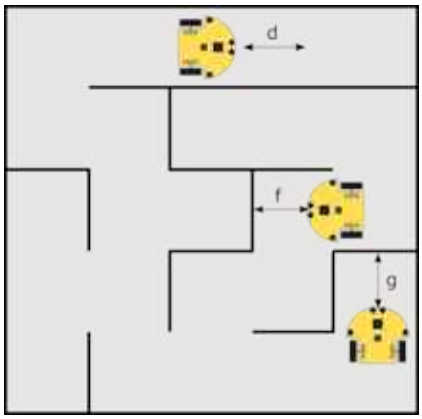
- P_a=0,08, amikor előtte van fal,
- P_b=0,12, amikor bal oldalt van fal,
- P_c=0,12, amikor jobb oldalt van fal,
- P_d=0,24, amikor bal és jobb oldalt van fal,
- P_e=0,36, amikor bal oldalt és előtte van fal,
- P_f=0,36, amikor jobb oldalt és előtte van fal,
- P_g=0,24, amikor balra előtte és jobb oldalt is fal van.

A robot egy-egy cellába több irányból beléphet, ahol szabad az út. A becsült valószínűségek az alábbi összefüggés szerint alakul:

$$P = \frac{lehetséges\ esetek\ száma}{összes\ eset\ száma} \tag{37}$$

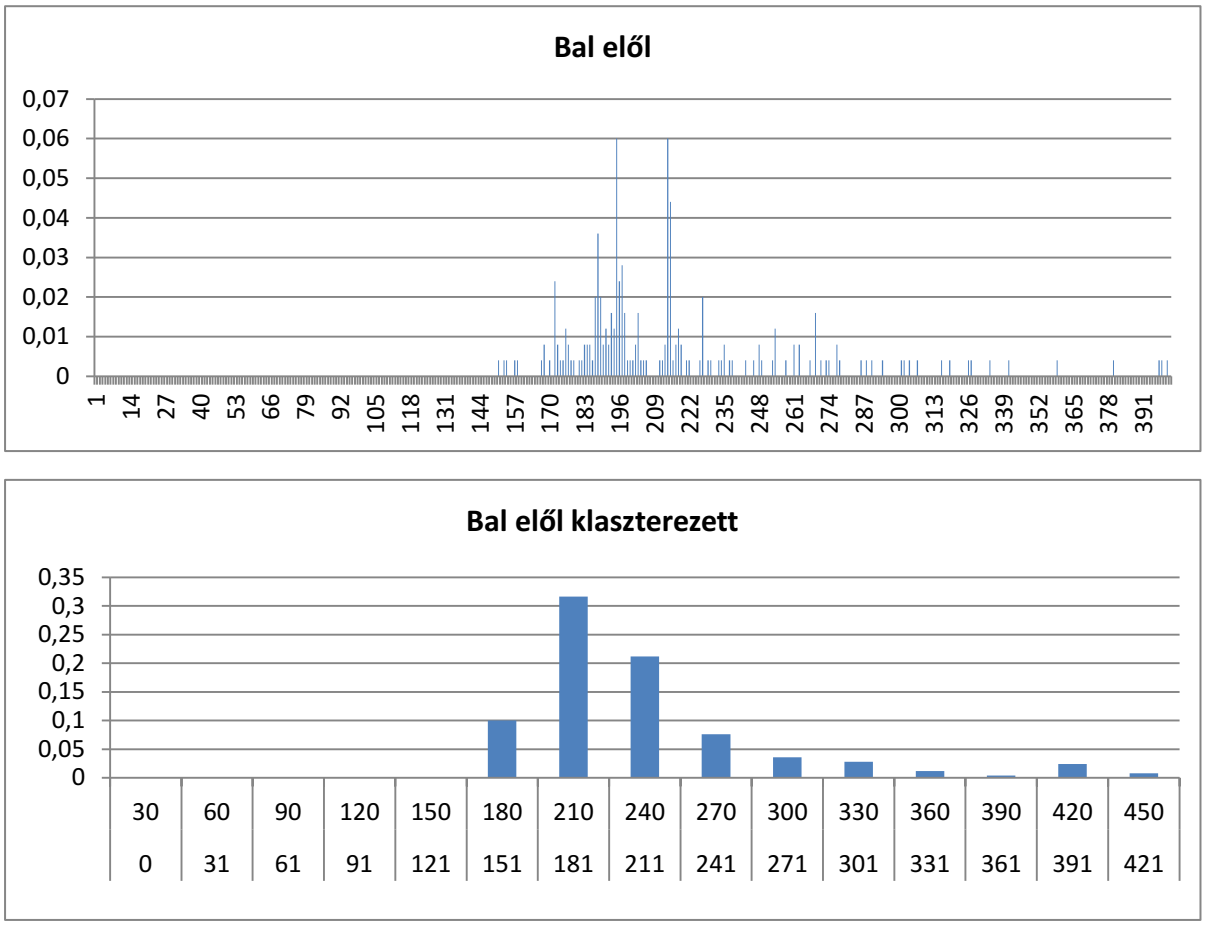
ahol az esetek száma abból adódik, hogy a robot több irányból érkezve milyen mintázatot láthat.

A robot az a-g osztályoknak megfelelő részeken végzett oda-vissza mozgást adatgyűjtés közben a 64. ábra szerint.



64. ábra Néhány osztályhoz tartozó tanító mozgás

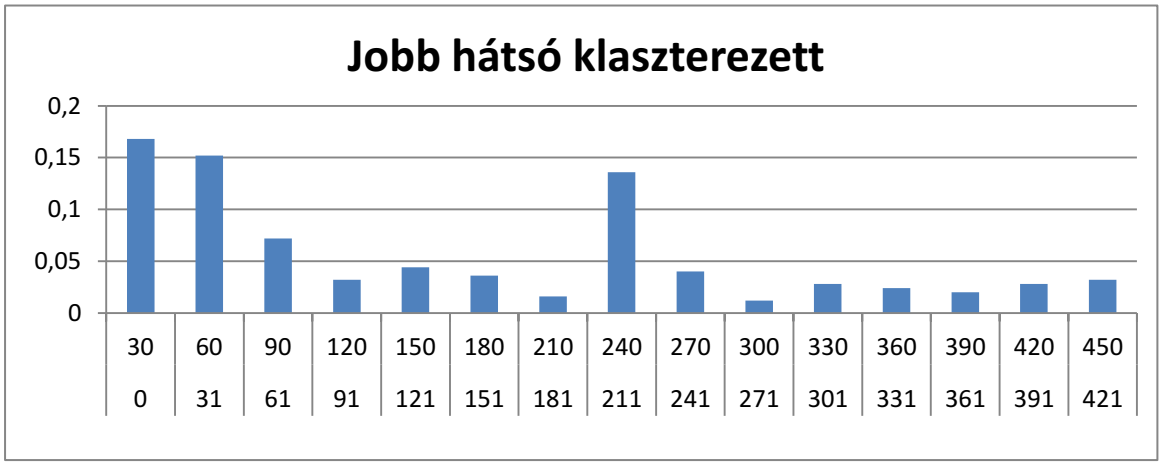
Az érzékelők [0; 400] tartományban szolgáltatják a mért értékeket (analóg-digitál átalakító eredménye). Az értékeket klaszteresztem, 14 egyenlő méretű klaszterre bontottam. Az egyes tartományokban előforduló értékek alapján számítottam ki a tartományok előfordulási



65. ábra Felül a klaszterezés előtti értékek valószínűségei alul pedig a klaszterezés után az egyes tartományok valószínűségei a bal első érzékelő esetén, amikor van előtte fal

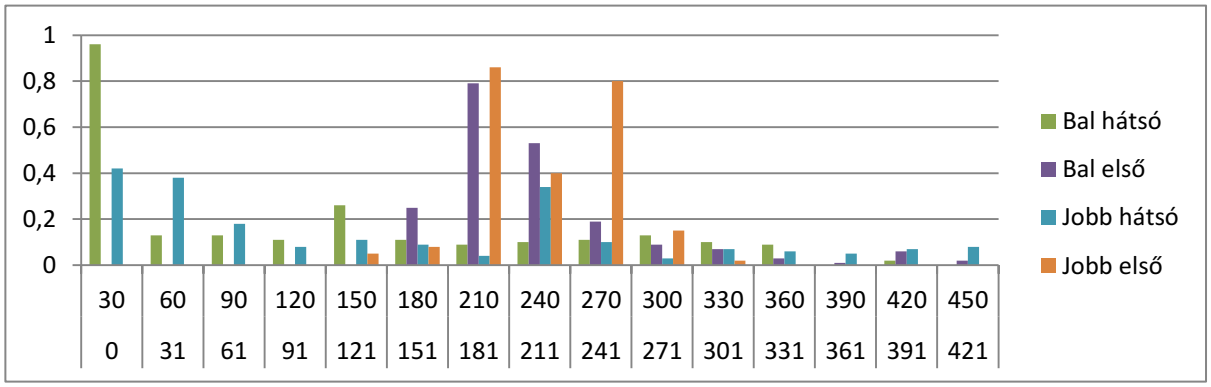
valószínűségét. A tartományok méretét tapasztalati úton választottam, hogy még ne okozzon túl nagy információvesztést. A klaszterezésre az adatbázis méretének és a zaj csökkentése érdekében volt szükség, mivel nem néhány tízezres csak ezres nagyságrendű a tanítóminta. Egy-egy tartomány mérete így 30-ra esett. A következő ábrán látszik, hogyan alakult az adathalmaz formája a klaszterezés és a valószínűségek számítása után (65. ábra). Erre a számítási igény csökkentése miatt volt szükség.

Az érzékelők közül a legtöbb zajt a jobb hátsó jele hordozta, amely ennek ellenére az osztályozók számára értékelhető információt hordozott. A zaj ellenére a változás elég jelentős ahhoz, hogy az osztályozó képes legyen különbséget tenni az egyes mintázatok között. A jobb hátsó érzékelő jelsorozatát, amikor fal van előtte, az alábbi ábra szemlélteti klaszterezés után (67. ábra):



67. ábra A jobb hátsó érzékelő zajos jele, amikor van előtte fal

A g osztály esetén, amikor a mobil robotot körül veszik a falak az alábbiak szerint alakultak a valószínűségek a klaszterezett adatokon (66. ábra):



66. ábra Az összes érzékelő klaszterezett adatainak valószínűségei a g mintázat esetén

A klaszterezett adatok valószínűségeit kiszámítottam a mért értékek alapján mind a 7 osztályra. Amikor egyik osztály sem érvényes, nincsenek falak a robot körül.

A mért értékek alapján készítettem egy szakértői szabálybázist a FIVE módszer számára.

7.3.3. Szabályok kialakítása a FIVE módszerhez

A FIVE módszer számára egy szakértői szabálybázis rendszert alakítottam ki a Bayes osztályozó számára gyűjtött adatok alapján. Mindkét osztályozó estén az elvárás, hogy a fal jelenlétét vagy a szabad utat jelezze a térképezés számára. Ezért a kimeneti univerzum két nyelvi értéket tartalmaz: igen vagy nem. A Bayes osztályozótól eltérően a FIVE esetén nem 7 osztály került definiálásra, hanem három, a három alapvető falhelyzetnek megfelelően:

- Fal balról,
- Fal jobbról,
- Fal elől.

A fuzzy alapvető működése szerint ezek a halmazok egyszerre is produkálhatnak magas tagsági értékeket. Míg a nem fuzzy jellegű osztályozók esetén egyetlen osztályt választ az osztályozó például a legnagyobb valószínűségűt a Bayes osztályozó esetén. Bemeneti értékeknek közeli, közép távolság és távoli és nagyon távoli nyelvi értékeket vezettem be, ahol a közép távolság és közeli értékek jelzik a fal jelenlétét. Egy-egy szabálybázis egy-egy osztályt képvisel a fentebb felsorolt három közül.

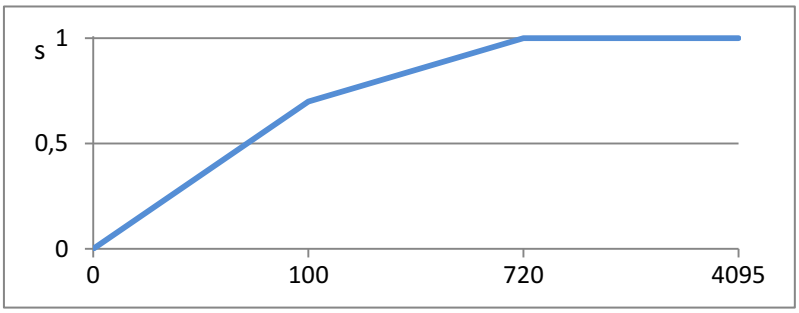
A fuzzy szabálybázisok esetén az egyes szabályok csak az őket érintő érzékelők megfigyeléseit használják fel. A Bayes osztályozó az egyes osztályokat az összes bemenő adat segítségével becsülte meg. A FIVE módszer esetén az összes megfigyelés bizonytalan működést eredményezett.

A nyelvi értékek az univerzumokban a robot faltól való távolsága alapján kerültek meghatározásra. A robot érzékelője [0; 4095] közötti tartományban biztosít értéket, amely fordítottan arányos a távolsággal, mértékegysége nincs. Tehát a 0 érték távoli falakat (nincs visszaverődés), míg a 4095 érték nagyon közeli falat (erős visszaverődés) jelez. A közepes távolság esetén a robot a cella közepén helyezkedik el, egyenlő távolságra minden faltól. Ilyenkor a grafikonon is látható 100 feletti értékeket ad a távolságmérőkre kötött analóg-digitális átalakító, ez lesz a határa a „*middle*”, közepes

címkével ellátott távolságnak. A közeli, „*close*” távolság érték 720, míg a legközelebbi értékek, „*very close*” 4096-ig terjednek. Utóbbi címke csak technikai jellegű, hogy lefedhető legyen a teljes tartománya az érzékelőnek. Az infravörös érzékelő pár, néhány centiméteres távolságról képes a fehér színű falakat érzékelni. A roboton az elhelyezés alapján, ha a robot az egyik falhoz közel halad az érzékelő már a másik oldalon már nem tudja megfelelő biztonsággal érzékelni a falat, csak annyit, hogy távol van. A „*very close*” állapot az érzékelő elhelyezkedése miatt nem fordulhat elő, de az érzékelő által biztosított legnagyobb érték miatt volt szükség, mivel az univerzumoknak mindig le kell fedni a teljes értéktartományt. Az FBDL-ben az alábbiak szerint jelennek meg:

```
universe "leftBack"
      "far" 0 0
      "middle" 100 0.7
      "close" 720 1
      "very close" 4096 1
end
```

Ahol a bal hátsó érzékelő bemeneti megfigyeléseit tartalmazó „*leftBack*” címkéjű univerzum elemei a „*far*”, „*middle*”, „*close*” és „*very close*”, rendre: távoli, középtáv, közeli és nagyon közeli értékeket tárol. A skálafüggvény értéke úgy került kialakításra, hogy a közepes távolságot már falnak érzékelje, abban az esetben is, amikor a robot éppen nagyon közel halad az ellentétes oldalon lévő falhoz. A „far” és „middle” értékek között illetve a „middle” és „close” értékek közötti ferde szakasz biztosítja, hogy az egyre közelebbi fal az egyenes meredekségének megfelelően egyre nagyobb skálaértéket szolgáltasson ez a kimeneten a döntésnél nagyobb változást okoz a közelebbi falak esetén. Még meredekebb szakasz még nagyobb kimenő értéket jelentene. Grafikonon ábrázolva az univerzumot az alábbi képet rajzolja (68. ábra).



68. ábra A „leftBack" univerzum grafikonja (a vízszintes tengely az ADC -Analog-Digital Converter - értékét veszi fel)

A többi érzékelő univerzuma is megegyezik a bal hátsó („*leftBack*") érzékelő univerzumával. A kimenetként szolgáló konzekvens univerzum az FBDL nyelven a következő:

```
universe "isWallOnLeft"
      "no" 0 0
      "yes" 1 1
end
```

Az „*isWallOnLeft*" univerzum a bal oldali falat érzékelő szabálybázis kimenete. A két nyelvi érték – „*yes*", „*no*" – a kimeneti tartomány lefedésére szolgál, ahol a kimeneti érték és a skálaérték is [0;1] intervallumban változik. A tapasztalati értékek alapján beállítottam minden osztálynak egy határértéket a végeredményre 0,7 értékre, amely fölött a rendszer úgy tekinti, hogy van fal az adott irányban. Erre a térképező algoritmus miatt volt szükség, amely csak a fal létezésének a tényét rögzíti nem pedig a mértékét fuzzy elven.

A szabályok antecedensei azokat a nyelvi értékeket tartalmazzák, amelyek a legjellemzőbbek az adott állapot szempontjából. A bal oldali falat érzékelő szabálybázis így a következő alakban néz ki FBDL nyelven:

```
rulebase "isWallOnLeft"
      rule
      "no" when "leftBack" is "far" and "leftFront" is "far"
      end
      rule
      "yes" when "leftBack" is "middle" and "lefFfront" is "middle"
      end
end
```

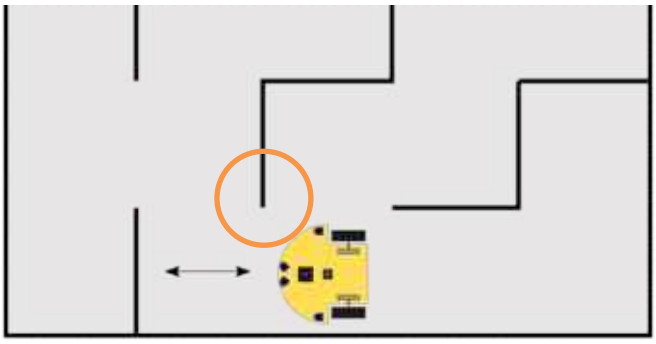
A helyes működéshez legkevesebb kettő szabályra van szükség. A fennmaradó két osztály szabálybázisa is az „*isWallOnLeft*" címkéjű szabálybázis felépítését követi az első illetve a jobb oldali érzékelők figyelembe vételével. A szabálybázisok kimenete a

megfigyelések univerzumait alkotó értékek miatt nemlineárisak. A kimeneti érték 0 és 1 közötti valós szám, amit akkor tekintettem aktív osztálynak, ha 0,7-től nagyobb értékű.

A szabálybázisok kiértékelését a µFRI könyvtárral megvalósított program végezte.

7.3.4. Eredmények

Az előző leírás alapján létrehozott két osztályozó eljárás a teszteléshez használt labirintus, különböző pontjain helyesen ismerték fel a robotot körülvevő falak formáját. Az tesztelés során figyeltem arra, hogy főként olyan helyeken végezzen a robot mérést, ahol adatgyűjtés nem történt a tanítómintához. A két osztályozó eredményei egymásnak megfelelőek és helyesek voltak. Egyetlen eltérés jelentkezett, amely pozíciót az 69. ábra mutat. Ahol a bekarikázott résznél FIVE alapú osztályozó mutatott téves érzékelést folyosó mintázatra.



69. ábra Fals folyosó érzékelés a FIVE alapú osztályozónál

7.4. Tézis

A létrehozott µFRI könyvtár vagy a FIVE IP modulok felhasználásával, a FIVE FRI módszer alkalmazásával hatékony osztályozó algoritmus alakítható ki. Az így kapott osztályozó algoritmus alkalmas beágyazott rendszerekben történő közvetlen használatra.

7.5. Új eredmények összefoglalása

A FIVE fuzzy interpolációs módszerrel kialakítható fuzzy osztályozó. Fuzzy logikán alapuló osztályozó előnye, hogy az osztályozó dimenziószáma csökkenthető, ahogy az ismertetett példában is látható volt. Interpolációs módszerrel kialakított fuzzy

osztályozók az alábbi cikkekben találhatók: [70], [71], [72] és [73], amelyek a KH módszeren alapulnak.

A FIVE módszerrel kialakított osztályozó előnye a Bayes-osztályozóhoz képest a tanító minta méretében látható. A néhány ezres nagyságrendű mintából számolt előfordulási valószínűségek helyett használható az szakértői (emberi) tapasztalat és néhány mért adatsorból létrehozható a működő szabályrendszer. A szabályrendszer esetenként igényel finomítást, de még ez is szakértő által elvégezhető. Az FBDL segítségével a szabályrendszer könnyen létrehozható.

Az µFRI könyvtár és az FIVE IP, FPGA megvalósítás is támogatja a gépi tanulással létrehozott szabálybázist vagy a már meglévő szabálybázis hangolását [31], [35], [36] és [74].

A FIVE módszer esetén elhagyható az adatok klaszterezése, amelyre szükség volt a Bayes-osztályozó esetén az tanítóminta tömörítése céljából.

A µFRI és a FIVE IP segítségével kialakítható beágyazott rendszereken is használható osztályozó eljárás.

7.6. Gyakorlati alkalmazás

A µFRI segítségével mikrovezérlőn szoftveres és a FIVE IP segítségével FPGA-n hardveres környezetben implementálható szabály alapú gyors osztályozó. A szabály alapú megoldás előnye, hogy szakértői tudásbázis alapján is felépíthető az osztályozó. Alkalmazható IoT (Internet of Things) rendszerekben viselkedések, minták felismerésében, korai hibafelismerés vagy karbantartás előrejelzésében. FPGA megvalósítással olyan nagy sebességet igénylő alkalmazás is lehetséges, mint számítógépes hálózat forgalomfigyelése valós időben.

7.7. Összegzés

A µFRI tesztelésére készítettem egy mintapéldát, amely egy mobil robot környezetében ismeri fel a falakat osztályozó eljárás segítségével. A µFRI mikrovezérlőn fut. Az eredményeket összehasonlítottam Naiv Bayes osztályozóval. A két osztályozót azonos adatminta segítésével paramétereztem fel. A különböző fal mintázatokat mindkét osztályozó helyesen ismerte fel. A FIVE módszerrel kialakított osztályozó faltöredék esetén helytelen eredményt adhat.

Kapcsolódó cikkek: [S5], [S7], [S8] és [S10].

Összegezve, a létrehozott szoftveres és hardveres megvalósításokkal hatékony osztályozó eljárás alakítható ki beágyazott rendszereken. A szabály alapú osztályozó paraméterezése kisebb adatmennyiségből szakértői szabályokkal (tapasztalati úton) is létrehozható, amellyel gyorsabban lehet az osztályozót tanítani.

8. Az értekezés tézisei

I. A µFRI könyvtár alkalmas a FIVE FRI mikrovezérlőkön való implementációjára. Lehetőséget biztosít a tudásbázis rendszerleállás nélküli dinamikus paraméter megváltoztatására, beágyazott rendszerbeli adaptív alkalmazások kialakítására.
Kapcsolódó cikkek: [S5], [S7], [S8] és [S10].

II. A µFRI könyvtár szorosan csatolt ARM A72 párhuzamos architektúrán való implementációja esetén a dinamikusan változó elosztott fuzzy tudásbázis összesen 1720 kB memóriaigényig hatékonyan allokálható 1 processzormagon.
Tetszőleges szabálybázis memória igénye kiszámítható a következő képlettel:

$$T = 8N_{UE} + 12N_U + 12N_A + 14N_R + 12N_{RB}$$

ahol az N_{UE} az összes univerzum elem száma, N_U az összes univerzumok száma, N_A az összes antecedens száma a szabályokban, N_R a szabályok száma, N_{RB} a szabálybázisok száma, T pedig a teljes szükséges tárterület mérete byte-ban. Ez alapján a legkisebb elfoglalt memóriaterület 58 byte, amikor $N_{UE}=N_U=N_A=N_R=N_{RB}=1$.
Kapcsolódó cikkek: [S3], [S4] és [S6].

III. A FIVE IP könyvtár moduljai alkalmasak a FIVE FRI FPGA-n való FPGA erőforrás hatékony implementációjára. A könyvtár felhasználásával a létrehozott rendszer képes a tudásbázis rendszerleállás nélküli dinamikus megváltoztatására, beágyazott rendszerbeli adaptív alkalmazások kialakítására.
Kapcsolódó cikkek: [S1], [S2], [S8] és [S13].

IV. A létrehozott µFRI könyvtár vagy a FIVE IP modulok felhasználásával, a FIVE FRI módszer alkalmazásával hatékony osztályozó algoritmus alakítható ki. Az így kapott osztályozó algoritmus alkalmas beágyazott rendszerekben történő közvetlen használatra.
Kapcsolódó cikkek: [S5], [S7], [S8] és [S10].

9. Összefoglalás

A FIVE módszerrel etológiai viselkedésmodellek és viselkedés alapú irányítás valósítható meg úgy, hogy a rendszer működését szabályok halmaza írja le. A FIVE módszer egy fuzzy interpolációs módszer, amely a fuzzy halmazokat az interpolációhoz egy többdimenziós, úgynevezett homályos környezetbe helyezi. A FIVE módszer egy könnyűsúlyú interpoláció, amely alkalmas beágyazott rendszereken történő használatra is. A viselkedések, szabályok leírását az FBDL nyelv segíti.

A dolgozatban a FIVE módszerhez kapcsolódóan elkészítettem a már meglévő megvalósítások mellé a C nyelvű beágyazott rendszerekre optimalizált változata, a µFRI könyvtár. Alacsony memóriaigénye miatt eredményesen alkalmazható mikrovezérlőkön való használatra is. A µFRI könyvtár továbbá alapját képezi a FIVE módszer FPGA megvalósításának. A könyvtár segítségével különféle robotok viselkedése került elkészítésre a kutatási idő alatt ARM mikrovezérlőre és alkalmazás processzoros rendszerekre. A µFRI könyvtár adatszerkezete felhasználja az FBDL nyelv szerkezetét a könnyebb átírás segítésére. A µFRI könyvtár lehetővé teszi a paraméterek működés közbeni dinamikus megváltoztatását, dinamikus memórifoglalást támogató rendszerek esetén a futás közbeni a teljes szabálybázis megváltoztatását.

A µFRI könyvtárat megvizsgáltam hogyan viselkedik többprocesszoros rendszeren. A nem gyorsítótárazott és a gyorsítótárazott adatokkal végzett számítási időket vizsgáltam 1, 2 és 3 processzormagon szorosan és lazán csatolt rendszeren. Bár a µFRI könyvtár lehetővé teszi, hogy az egyes szabálybázisokat a rendszer egymástól függetlenül számítsa, de a rövid számítási idő és az alacsony memóriaigény miatt hatékonyabban működik a vizsgált 1 és 10000 szabálybázisszám között (20000 szabály), ha egy processzormag végzi a számítást. Megmértem, hogy mennyivel gyorsabb a végrehajtás több processzormagon az egy processzormagos számításokhoz képest. A számítási idők hányadosa alapján és Amdahl képletével vizsgáltam meg. A vizsgálatokhoz a kutatás ideje alatt elterjedt ARM A53 és ARM A72 architektúrákat használtam.

A FIVE módszer további gyorsítása érdekében elkészítettem a hardveres megvalósítását FPGA-ra Verilog hardverleíró nyelven. A 4 MHz maximális működési frekvenciájú hardver alapértelmezetten 8 bites, de paraméterezhető módon tetszőleges

bitszélességgel generálható. A hardver előjel nélküli egész számokkal képes műveletet végezni. Az elkészült megvalósítás hardverigényét összevetettem a Xilinx High Level Systhesis nevű eszközével, amely speciálisan átalakított C kódból képes leíró nyelvre fordítani. Sikerült elérni, hogy a saját megvalósításom kevesebb hardveres erőforrást használjon az FPGA-n. Az általam készített hardveres megvalósítás lehetővé teszi a paraméterek működés közbeni dinamikus megváltoztatását.

A FIVE módszert a µFRI könyvtár segítségével egyéb alkalmazások mellett osztályozó eljárásként is felhasználtam. Egy valós, labirintusban közlekedő robot érzékelő adatait feldolgozva határoztam meg szabályokkal, hogy a robot körül milyen falmintázat található az adott négyzet alakú labirintus cellában. A roboton 4 darab egyszerű infravörös érzékelő helyezkedett el és ezek segítségével érzékelte a falakat, akadályokat. Az FIVE alapú osztályozót összevetettem a klasszikus Naiv-Bayes osztályozóval, végeredményben a két megoldás egyforma valószínűséggel találta el a falak helyzetét a robot körül.

A kutatás további lépései közé tartozik a szabályrendszer előállítása a 7. fejezetben szerzett ismeretek alapján, statisztikai módszerekkel, segítve a gépi tanulás kialakítását a már meglévő Q-tanulás mellett. A cél, hogy szakértő által bemutatott viselkedésből generáljon a rendszer szabálybázisokat, amelyeket aztán egy Q-tanulás módszer finomíthat. Továbbá a hardveres megvalósítás dinamikus módosíthatóságának továbbfejlesztése a cél a modern, Xilinx által biztosított megoldásokhoz igazítva.

A kutatás során tehát elkészítettem olyan megoldásokat, amelyek lehetővé teszik a FIVE módszer hatékony használatát beágyazott rendszereken processzoros és hardveres rendszereken egyaránt illetve lehetővé teszik a szabálybázisok dinamikus paraméterezhetőségét, hangolását.

10. Summary

Ethological behaviour models and behaviour based control can be implemented wtih the FIVE method. The operation of behaviour based system is described by a set of rules. The FIVE method is a fuzzy interpolation method that places fuzzy sets in a multidimensional, so-called fuzzy environment for interpolation. The FIVE method is a lightweight interpolation that is also suitable for use on embedded systems. The FBDL language helps to describe the behaviours and rules.

In the dissertation I created a version of FIVE method which is optimized for embedded systems in languge C. Due to its low memory requirements, it can also be used successfully on microcontrollers. The µFRI library is also the basis for the implementation of the FIVE method FPGA. Various robots were developed based on behaviour based control during the research period. The behaviour implementations are based on µFRI library for ARM microcontrollers and application processor systems. The data structure of the µFRI library uses the structure of the FBDL language to help with easier transcription. The µFRI library allows to change parameters dynamically during operation for systems that support dynamic memory allocation. The entire rulebase can be changed at runtime.

I examined how the µFRI library behaves on a multiprocessor system. I investigated the computation times with non-cached and cached data on a tightly and loosely coupled system with 1, 2, and 3 processor cores. Although the µFRI library allows each rule base to be computed independently, it works more efficiently between 1 and 10,000 number of rulebases (20,000 rules) due to the short computation time and low memory requirements when one processor core performs the calculation. I examined that how much faster execution is done on multiple processor cores compared to single-processor calculations. Based on the quotient of the computational times and I used Amdahl's formula. For the studies, I used the popular ARM A53 and ARM A72 architectures during the research.

I created a hardware implementation for FPGA in the Verilog hardware description language to further accelerate the FIVE method. The FIVE IP is operating with 8 bits integers by default at maximum 4 MHz frequency but the bit witdh is paramemterizable. I compared the hardware requirements of my implementation with a

tool called Xilinx High Level Systhesis, which can translate from specially modified C code into a descriptive language. I managed to make my own implementation use less hardware resources on the FPGA. The hardware implementation I created allows me to change the parameters dynamically during operation.

I used the FIVE method – implemented with μFRI library – as a classification procedure. By processing the sensor data of a real robot moving in a maze, I determined by rules what wall pattern is in the given square maze cell around the robot. The robot was equipped with 4 simple infrared sensors and used them to detect walls and obstacles. I compared the FIVE-based classifier with the classic Naiv-Bayes classifier, and in the end, the two solutions hit the position of the walls around the robot with equal probability.

Further steps in the research include the generation of a set of rules based on the knowledge gained in Chapter 7, using statistical methods to help develop machine learning in addition to existing Q-learning. The goal is to generate rule bases from the behavior presented by an expert that can be tuned by a Q-learning method. Furthermore, the goal is to further improve the dynamically reconfigurable hardware implementation by adapting it to modern solutions provided by Xilinx.

During the research, I have created solutions that allow the efficient use of the FIVE method on embedded systems on both processor and hardware systems, as well as the dynamic parameterization and tuning of rulebases.

11.Irodalom jegyzék

[1] Szilveszter Kovács, Dávid Vincze, Márta Gácsi, Ádám Miklósi, and Péter Korondi, "Ethologically inspired robot behavior implementation," in *2011 4th International Conference on Human System Interactions, HSI 2011*, May 2011, pp. 64–69.

[2] Michiel P. De Looze, Tim Bosch, Frank Krause, Konrad S. Stadler, and Leonard W. O'sullivan, "Exoskeletons for industrial application and their potential effects on physical work load," *Ergonomics*, vol. 59, pp. 671–681, 2016.

[3] Universal Robots, "www.universal-robots.com," Tech. rep. 2020.

[4] Stuart Shepherd and Alois Buchstab, "Kuka robots on-site," in *Robotic Fabrication in Architecture, Art and Design 2014*.: Springer, 2014, pp. 373–380.

[5] Simon Blackmore, Bill Stout, Maohua Wang, Boris Runov, and others, "Robotic agriculture–the future of agricultural mechanisation," in *Proceedings of the 5th European conference on precision agriculture*, 2005, pp. 621–628.

[6] Richard Bloss, "Mobile hospital robots cure numerous logistic needs," *Industrial Robot: An International Journal*, 2011.

[7] Bence Kovács et al., "Ethologically Inspired Robot Design," in *The 2nd International Conference on Cognitive Infocommunications (CogInfoCom2011)* , 2011, pp. 7-9.

[8] Péter Korondi, Ádám Miklósi, Szilveszter Kovács, and Antal Dóka, "Ethologically inspired models for human-robot interaction-state of the art," 2009.

[9] Dávid Vincze and Szilveszter Kovács, "Using fuzzy rule interpolation based automata for controlling navigation and collision avoidance behaviour of a robot," in *2008 IEEE International Conference on Computational Cybernetics*, 2008, pp. 79–84.

[10] Hongwei Mo, Qirong Tang, and Longlong Meng, "Behavior-based fuzzy control for mobile robot navigation," *Mathematical problems in engineering*, vol. 2013, 2013.

[11] Paolo Pirjanian, "Multiple objective behavior-based control," *Robotics and Autonomous Systems*, vol. 31, pp. 53–60, 2000.

[12] Michael Dooley, Nikolai Romanov, Paolo Pirjanian, Lihu Chiu, and Enrico Di Bernardo, Robotic game systems and methods, January 2014, US Patent 8,632,376.

[13] Lotfi A. Zadeh, "Information and control," *Fuzzy sets*, vol. 8, pp. 338–353, 1965.

[14] Johanyák Zsolt Csaba and Dr. Tóth Tibor, "Fuzzy szabály-interpolációs módszerek és mintaadatok alapján történő automatikus rendszergenerálás," Ph. D. értekezés, Miskolci Egyetem, Hatvany József Informatikai Tudományok, Ph.D. dissertation 2007.

[15] Kovács Szilveszter, "Fuzzy logic control," Technical University of Budapest, Faculty of Informatics and Electrical Engineering, Master's thesis 1993.

[16] Szilveszter Kovács, "Extending the fuzzy rule interpolation" FIVE" by fuzzy

observation," in *Computational Intelligence, Theory and Applications*.: Springer, 2006, pp. 485–497.

[17] LászlóT Kóczy and Kaoru Hirota, "Approximate reasoning by linear rule interpolation and general approximation," *International Journal of Approximate Reasoning*, vol. 9, pp. 197–225, 1993.

[18] LászlóT Kóczy and Kaoru Hirota, "Interpolative reasoning with insufficient evidence in sparse fuzzy rule bases," *Information Sciences*, vol. 71, pp. 169–201, 1993.

[19] Domonkos Tikk et al., "Stability of interpolative fuzzy KH controllers," *Fuzzy Sets and Systems*, vol. 125, pp. 105–119, 2002.

[20] Gy Vass, L. Kalmár, and L. T. Kóczy, "Extension of the fuzzy rule interpolation method," in *Proc. Int. Conf. Fuzzy Sets Theory Applications*, 1992, pp. 1–6.

[21] Péter Baranyi, László T. Kóczy, and Tamás D. Gedeon, "A generalized concept for fuzzy rule interpolation," *IEEE Transactions on Fuzzy Systems*, vol. 12, pp. 820–837, 2004.

[22] Suman Das, Debjani Chakraborty, and László T. Kóczy, "Linear fuzzy rule base interpolation using fuzzy geometry," *International Journal of Approximate Reasoning*, vol. 112, pp. 105–118, 2019.

[23] Szilveszter Kovács, Márta Gácsi, Dávid Vincze, Péter Korondi, and Ádám Miklósi, "A novel, ethologically inspired HRI model implementation: Simulating dog-human attachment," in *2nd International Conference on Cognitive Infocommunications*, 2011, pp. 1–4.

[24] Szilveszter Kovács and László T. Kóczy, "Application of interpolation-based fuzzy logic reasoning in behaviour-based control structures," in *2004 IEEE international conference on fuzzy systems (IEEE Cat. No. 04CH37542)*, vol. 3, 2004, pp. 1543–1548.

[25] Szilveszter Kovács and László T. Kóczy, "Application of the approximate fuzzy reasoning based on interpolation in the vague environment of the fuzzy rulebase in the fuzzy logic controlled path tracking strategy of differential steered AGVs," in *International Conference on Computational Intelligence*, 1997, pp. 456–467.

[26] Szilveszter Kovacs and Laszlo T. Koczy, "Approximate fuzzy reasoning based on interpolation in the vague environment of the fuzzy rulebase," in *Proceedings of IEEE International Conference on Intelligent Engineering Systems*, 1997, pp. 63–68.

[27] Szilveszter Kovács, "New aspects of interpolative reasoning," in *Proceedings of the 6th. International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Granada, Spain*, 1996, pp. 477–482.

[28] Imre Piller, Dávid Vincze, and Szilveszter Kovács, "Declarative language for behaviour description," in *Emergent trends in robotics and intelligent systems*.: Springer, 2015, pp. 103–112.

[29] Dávid Vincze and Szilveszter Kovács, "Incremental rule base creation with fuzzy rule interpolation-based Q-learning," in *Computational Intelligence in Engineering*.:

Springer, 2010, pp. 191–203.

[30] Dávid Vincze and Szilveszter Kovács, "Fuzzy rule interpolation-based Q-learning," in *2009 5th International Symposium on Applied Computational Intelligence and Informatics*, 2009, pp. 55–60.

[31] Tamás Tompa and Szilveszter Kovács, "Applying expert heuristic as an a priori knowledge for FRIQ-learning," *Acta Polytechnica Hungarica*, vol. 17, pp. 27–45, 2020.

[32] Imre Piller and Szilveszter Kovács, "FBDL: A Declarative Language for Interpolative Fuzzy Behavior Modeling," in *2019 IEEE 23rd International Conference on Intelligent Engineering Systems (INES)*, 2019, pp. 000295–000300.

[33] Balázs Pónya and László Czap, "Gömbrobot stabilizálása viselkedés leíró nyelvvel," in *ENELKO 2017 XVIII. Nemzetközi Energetika-Elektrotechnika Konferencia, SzámOkt 2017 XXVII. Nemzetközi Számítástechnika és Oktatás Konferencia, Kolozsvár 2017. október 12-15.*, Cluj-Napoca, RomĂˇnia, 2017.

[34] Gergő Lengyel, "Egyensúlyozó robot megvalósítása Fuzzy szabály interpolációval (FRI)," Miskolci Egyetem, Master's thesis 2017.

[35] Tamás Tompa and Szilveszter Kovács, "Clustering-based fuzzy knowledgebase reduction in the FRIQ-learning," in *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2017, pp. 000197–000200.

[36] Tamás Tompa, Szilveszter Kovács, Dávid Vincze, and Mihoko Niitsuma, "Demonstration of expert knowledge injection in Fuzzy Rule Interpolation based Q-learning," in *2021 IEEE/SICE International Symposium on System Integration (SII)*, 2021, pp. 843-844.

[37] Gene M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[38] Cypress. (2020) PSoC Products. [Online]. https://www.cypress.com/products/microcontrollers-mcus

[39] Xilinx. (2020) Products. [Online]. https://www.xilinx.com/products/silicon-devices/soc.html

[40] S. T. Microelectronics. (2020) Products. [Online]. https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html

[41] Atul P. Godse and Deepali A. Godse, *Microprocessors & Microcontrollers*.: Technical publications, 2021.

[42] János Végh, Péter Molnár, and József Vásárhelyi, "A figure of merit for describing the performance of scaling of parallelization," *arXiv preprint arXiv:1606.02686*, 2016.

[43] János Végh, József Vásárhelyi, and Dániel Drótos, "The performance wall of large parallel computing systems," in *International Conference on Reliability and Statistics in Transportation and Communication*, 2018, pp. 224–237.

[44] STMicroelectronics, "STM32F4 datasheet," p. 80, Tech. rep.. [Online]. https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[45] Dávid Vincze and Szilveszter Kovács, "Performance Optimization of the Fuzzy Rule Interpolation Method" FIVE".," *J. Adv. Comput. Intell. Intell. Informatics*, vol. 15, pp. 313–320, 2011.

[46] Ulrich Drepper, "What every programmer should know about memory," *Red Hat, Inc*, vol. 11, p. 2007, 2007.

[47] Haohuan Fu and Robert G. Clapp, "Eliminating the memory bottleneck: an FPGA-based solution for 3D reverse time migration," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011, pp. 65–74.

[48] Andreas Nowatzyk, Fong Pong, and Ashley Saulsbury, "Missing the memory wall: The case for processor/memory integration," in *23rd Annual International Symposium on Computer Architecture (ISCA'96)*, 1996, pp. 90–90.

[49] Digilent Inc. (2017) Zybo reference manual. [Online]. https://reference.digilentinc.com/_media/reference/programmable-logic/zybo/zybo_rm.pdf

[50] RaspberryPi.org. (2020) Raspberry Pi 4 B. [Online]. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[51] Broadcom. (2021) BCM5871 adatlap. [Online]. https://docs.broadcom.com/doc/1211168571391

[52] Kernel.org. (2020) Scheduler. [Online]. https://docs.kernel.org/scheduler/

[53] József Vásárhelyi and János Végh, "Clock Around Embedded Systems and Reconfigurable Systems," 2013.

[54] Xilinx. (2022) Xilinx Unified Software Platform. [Online]. https://www.xilinx.com/products/design-tools/vitis.html

[55] Kovács Szilveszter Tompa Tamás, "Heuristically accelerated FRIQ-learning," *IEEE 20th Jubilee International Symposium on Intelligent Systems and Informatics (SISY 2022), Szabadka, Szerbia : IEEE (2022)*, p. 6, 2022.

[56] Daniele Bagni, A. Di Fresco, J. Noguera, and F. M. Vallina, "A zynq accelerator for floating point matrix multiplication designed with vivado hls," *Application note, January*, 2016.

[57] Xilinx. (2013) Xilinx, Introduction to High-Level Synthesis with Vivado HLS. [Online]. www.xilinx.com

[58] Xilinx, "UltraFast High-Level Productivity Desing Methodology Guide, UG1197," , 2016.

[59] Xilinx, "Vivado Design Suite - High Level Synthesis, UG902," , 2012, 07. 25.

[60] Xilinx, "High-Level Synthesis with Vivado HLS, heterogen_xilinx_hls.pdf," , 2012.

[61] Daniel T. Larose, *Data mining and predictive analytics*.: John Wiley & Sons, 2015.

[62] Irina Rish and others, "An empirical study of the naive Bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, 2001, pp. 41–46.

[63] Farzad Zafarani and Chris Clifton, "Differentially private naive bayes classifier using smooth sensitivity," *arXiv preprint arXiv:2003.13955*, 2020.

[64] Manuel Baltieri, "A Bayesian perspective on classical control," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[65] Baosheng He, *New Bayesian methods for quality control applications*.: The University of Iowa, 2018.

[66] Rainer Deventer, Joachim Denzler, and Heinrich Niemann, "Bayesian control of dynamic systems," in *Innovations in Intelligent Systems*.: Springer, 2004, pp. 21–50.

[67] Pedro A. Ortega. Bayesian Control Rule. [Online]. https://www.adaptiveagents.org/lib/exe/fetch.php?media=bayesiancontrolrule.pdf

[68] Pedro Alejandro Ortega, Daniel Alexander Braun, and Simon Godsill, "Reinforcement learning and the Bayesian control rule," in *International Conference on Artificial General Intelligence*, 2011, pp. 281–285.

[69] Hably Alexandra, "Szériában gyártható half-size MicroMouse robot tervezése," in *Konzulens: Szayer Géza, Tajti Ferenc*, BME-MOGI, Budapest TDK dolgozat, 2014, p. 35.

[70] Ferenc Lilik, Szilvia Nagy, and László T. Kóczy, "Wavelet based fuzzy rule bases in pre-qualification of access networks' wire pairs," in *AFRICON 2015*, 2015, pp. 1–5.

[71] Ferenc Lilik, Szilvia Nagy, Melinda Kovács, Szonja Krisztina Szujó, and László T. Kóczy, "Interpolative decisions in the fuzzy signature based image classification for liver CT," in *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2021, pp. 1–6.

[72] Szilvia Nagy, Ferenc Lilik, and Laszló T. Kóczy, "Entropy based fuzzy classification and detection aid for colorectal polyps," in *2017 IEEE AFRICON*, 2017, pp. 78–82.

[73] Szilvia Nagy, Ferenc Lilik, and László T. Kóczy, "Applicability of various wavelet families in fuzzy classification of access networks' telecommunication lines," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017, pp. 1–6.

[74] Tamás Tompa and Szilveszter Kovács, "Determining the minimally allowed rule-distance for the incremental rule-base contruction phase of the FRIQ-learning," in *2018 19th International Carpathian Control Conference (ICCC)*, 2018, pp. 480–483.

[75] Xilinx, "High-Level Systhesis with Vivado HLS," 2012, heterogen_xilinx_hls.pdf.

[76] Paolo Pirjanian, "Behavior coordination mechanisms-state-of-the-art," Technical

report, University of Southern California, Institute for Robotics â€¦, Tech. rep. 1999.

[77] Pedro A. Ortega and Daniel A. Braun, "A bayesian rule for adaptive control based on causal interventions," *arXiv preprint arXiv:0911.5104*, 2009.

[78] Andreas Olofsson, Tomas Nordström, and Zain Ul-Abdin, "Kickstarting high-performance energy-efficient manycore architectures with epiphany," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, 2014, pp. 1719–1726.

[79] Amin Shaka Aunali and D. Venkatesan, "Bayesian Approach in Control Charts Techniques," *Int. J. Sci. Res. in Mathematical and Statistical Sciences Vol*, vol. 6, p. 2, 2019.

[80] Kovács Szilveszter. FIVE jegyzetek.

Hivatkozások ellenőrizve: 2022. 12. 12.

12. Saját cikkek

[S1] Bartók, Roland; Vásárhelyi, József: **Design of a FPGA Accelerator for the FIVE Fuzzy Interpolation Method**, INTERNATIONAL JOURNAL OF COMPUTER APPLICATIONS IN TECHNOLOGY, 68: 4 pp. 321-331. Paper: 11 , 11 p. (2022)

[S2] *Bartók, Roland*: **Fuzzy szabály interpolációs eljárás HLS-el történő hardveres megvalósíthatóságának vizsgálata**: Analysis of hardware implemented fuzzy rule interpolation, implemented with HLS In: Dr. Sebestyén-Pál, György; Dr. Szabó, Loránd; Dr. Biró, Károly-Ágoston (szerk.) ENELKO 2019 SzámOkt 2019 : XX. Nemzetközi Energetika-Elektrotechnika Konferencia, XXIX. Nemzetközi Számítástechnika és Oktatás Konferencia Kolozsvár, Románia : Erdélyi Magyar Műszaki Tudományos Társaság (EMT), (2019) pp. 115-118. , 4 p.

[S3] *Bartók, Roland ; Vásárhelyi, József*: **Examining Cache Handling of the FIVE Method on Multicore Systems** In: Szakál, Anikó (szerk.) SAMI 2019 : IEEE 17th World Symposium on Applied Machine Intelligence and Informatics Herlany, Szlovákia : IEEE, (2019) pp. 141-146. Paper: 24_sami2019 , 5 p.

[S4] *Bartók, Roland ; Vásárhelyi, József*: **A FIVE módszer párhuzamosíthatóságának vizsgálata Parallella mikroszámítógépen**: Examining the parallelism of FIVE method on Parallella microcomputer In: Bíró-Károly, Ágoston; Sebestyén-Pál, György; Szabó, Lóránd (szerk.) ENELKO 2018 XIX. Nemzetközi Energetika-Elektrotechnika Konferencia SzámOkt 2018 XXVIII. Nemzetközi Számítástechnika és Oktatás Konferencia Erdélyi Magyar Műszaki Tudományos Társaság (EMT), (2018) p. 134 , 6 p.

[S5] *Bartók, Roland ; Vásárhelyi, József*: **Fuzzy Rule Interpolation Based Object Tracking and Navigation for Social Robot** Lecture Notes In Mechanical Engineering 2018 pp. 370-375. , 6 p. (2018)

[S6] *Bartók, Roland ; Vásárhelyi, József*: **Parallelization of FIVE method on multicore embedded system** In: Drótos, Dániel; Vásárhelyi, József; Czap, László; Ivo, Petráš (szerk.) Proceedings of the 19th International Carpathian Control Conference (ICCC 2018) Piscataway (NJ), Amerikai Egyesült Államok : IEEE, (2018) pp. 400-403., 4 p.

[S7] *Bartók, Roland ; Korcsok, Erika ; Vásárhelyi, József:* **Fuzzy szabály interpoláció alapú objektumkövetés megvalósítása Robot Operációs Rendszerben**: Implementation of Fuzzy Rule Interpolation Based Object Tracking in Robot Operating System In: Bíró, Károly-Ágoston; Sebestyén-Pál, György (szerk.) ENELKO 2017 : XVIII. Energetika-Elektrotechnika Konferencia : SzámOkt 2017 : XVII. Nemzetközi Számítástechnika és Oktatás Konferencia Kolozsvár, Románia : Erdélyi Magyar Tudományos Társaság, (2017) pp. 123-126. , 4 p.

[S8] *Bartók, Roland ; Vásárhelyi, József*: **Two Methods for Autonomous Robot Obstacle Sensing and Application Programming Interface for Fuzzy Rule Interpolation** In: Dan, Popescu; Dorin, Şendrescu; Monica, Roman; Elvira, Popescu; Lucian, Bărbulescu (szerk.) 2017 18th International Carpathian Control Conference (ICCC) Craiova, Románia: IEEE, (2017) pp. 87-92. Paper: 7970376 , 6 p.

[S9] *Bartók, Roland ; Pintér, Judit Mária*: **Beszéd alapú ember-gép interfész fejlesztése szociális robothoz: Development of Speech-based Human-machine Interface for Social Robot** In: OGÉT 2017: XXV. Nemzetközi Gépészeti Konferencia : 25th International Conference on Mechanical Engineering Kolozsvár, Románia : Erdélyi Magyar Műszaki Tudományos Társaság (EMT), (2017) pp. 67-70., 4 p.

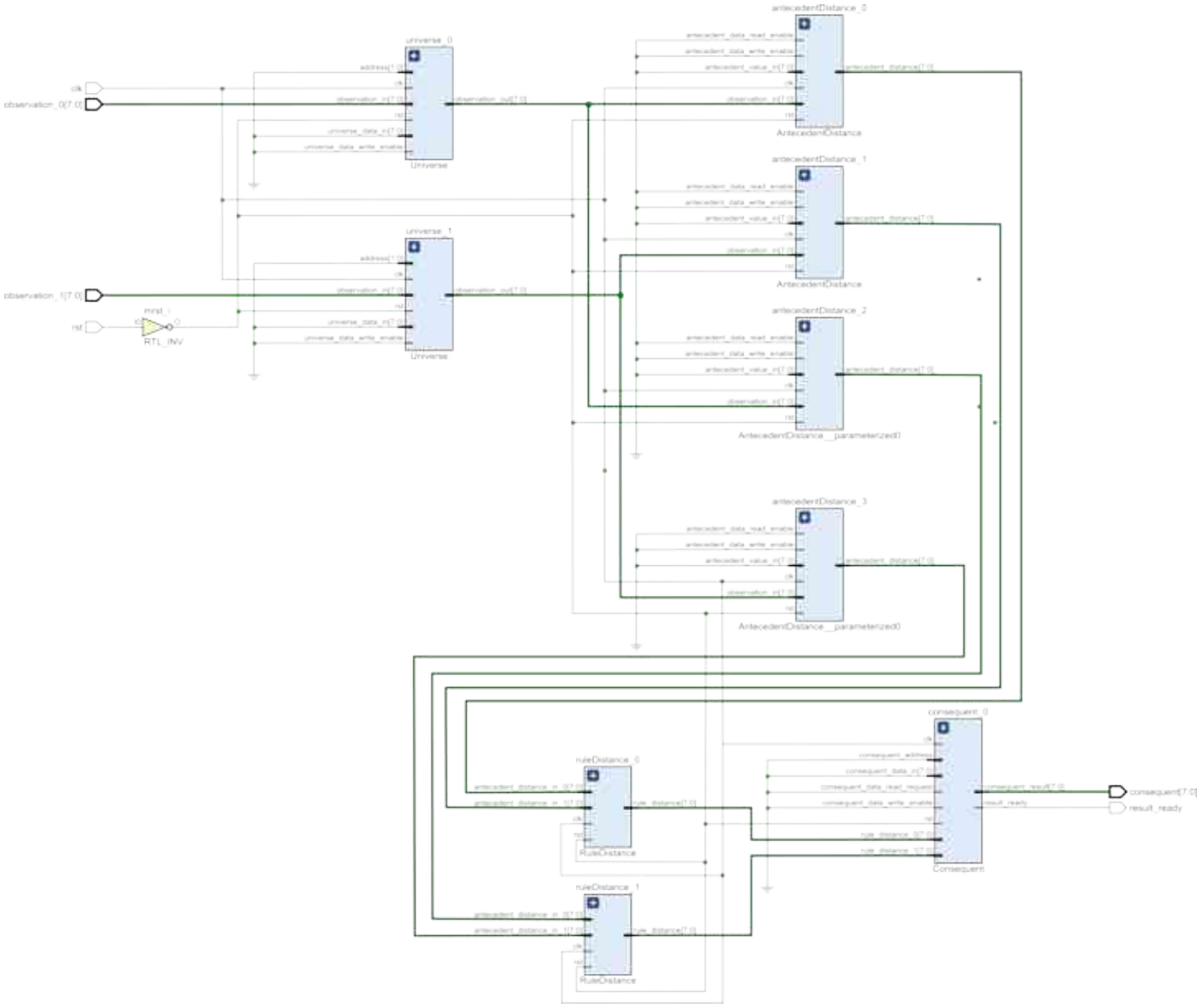
[S10] *Bartók, Roland ; Ahmed, Bouzid ; Vásárhelyi, József ; L Kiss, Márton*: **Wall and Object Detection with FRI and Bayes-Classifier for Autonomous Robot** Lecture Notes In Mechanical Engineering F12 pp. 383-389. , 7 p. (2017)

[S11] *Bartók, Roland ; L, Kiss Márton ; Dr. Vásárhelyi, József ; Dr. Kovács, Szilveszter ; Ahmed, Bouzid*: **Embedded behavioral model implementation** In: Ivo, Petráš; Igor, Podlubny; Ján, Kačur (szerk.) Proceedings of the 2016 17th International Carpathian Control Conference (ICCC) [s. l.] - Nemzetközi, Nemzetközi : IEEE, (2016) pp. 35-40., 6 p.

[S12] *Bartók, Roland ; L, Kiss Márton ; Vásárhelyi, József ; Ferenczi, István*: **Holonomikus hajtású robot vezérlése Fuzzy alapú viselkedés leírással** In: Biró, Károly-Ágoston; Sebestyén-Pál, György (szerk.) ENELKO 2015 XVI Nemzetközi Energetika-Elektrotechnika konferencia; SzámOkt 2015 XXV. Nemzetközi Számítástechnika és Oktatás Konferencia Arad, Románia : Erdélyi Magyar Műszaki Tudományos Társaság (EMT), (2015) pp. 185-188. , 4 p.

[S13] *Bartók, Roland ; Vásárhelyi, József*: ***A fuzzy rule interpolation base algorithm implementation on different platforms*** In: Ivo, Petras; Igor, Podlubny; Jan, Kacur; Vásárhelyi, József (szerk.) Proceedings of the 16th International Carpathian Control Conference Miskolc, Magyarország : IEEE IAS/IES/PELS, (2015) pp. 37-40. , 4 p.

Melléklet


antecedentDistance_0
universe_0
clk
observation_0[7:0]
Universe
AntecedentDistance
antecedentDistance_1
universe_1
observation_1[7:0]
rst
rst_i
RTL_INV
antecedentDistance_2
AntecedentDistance__parameterized0
antecedentDistance_3
consequent_0
ruleDistance_0
RuleDistance
ruleDistance_1
Consequent
consequent[7:0]
result_ready