Robotágens fejlesztése a hardvertől a szoftverig

Készítette: Bolla Kálmán Milán
Konzulens: Dr. Várterész Magda

2009. Május 4.

Tartalomjegyzék

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Várterész Magdának diplomamunkám elkészítésében nyújtott segítségéért és útmutató tanácsaiért. Továbbá Dr. Fazekas Gábornak robotika témakörében nyújtott segítségért és a költséges hardverelemek beszerzéséért.

Köszönettel tartozom Kovács Sándornak a mikrovezérlővel és elektronikával kapcsolatos összes segítségéért. Nélküle nem valósulhatott volna meg diplomamunkám hardveres része.

Valamint Dr. Fazekas Attilának képfeldolgozásban nyújtott segítségéért.

1. fejezet

Bevezetés

Diplomamunkám elkészítése során egy demonstrációs célú robotágenst terveztem meg. A feladat megoldása során körbejártam a mesterséges intelligencia ágensek, robotika, robotok vizuális látása és robotirányítás témaköreit. Bemutatom továbbá az általam készített demonstrációs eszközt, ami szemlélteti a témakörökkel kapcsolatos elméleti és gyakorlati tudásomat mind a hardver elkészítésén, mind pedig a hardvert működtető szoftver implementálásán keresztül.

Mesterséges intelligencia a 20. század közepétől kezdve nőtte ki magát önálló tudományterületté. Célkitűzése az emberi gondolkodás megértése, hogyan tudjuk a nagy és bonyolult világot észlelni, megérteni, annak alakulását megjósolni és manipulálni. Ezen kívül nem csak az intelligens entitások megértésével, hanem azok építésével is próbálkozik. Napjainkban már szinte az élet minden területén találkozhatunk olyan eszközökkel, amik felhasználják a mesterséges intelligencia által elért eredményeket. Ez a tudományterület nagyon szerteágazó, sok más tudományterületből tevődik össze, többekközt felhasználja a matematikai logikát és matematikai statisztikát.

A robotika két évtizeddel később indult fejlődésnek, mint a mesterséges intelligencia. Az igény, ami inspirálta az első robotok megalkotóit azaz ősi törekvés, hogy az ember minél jobban szeretné a munka alól tehermentesíteni magát. A robotok első generációja nem mondható túl intelligensnek, de a robotika fejlődése során egyre többet vett át a mesterséges intelligencia témakörében elért eredményekből. Mára pedig a robotika olyan szorosan kapcsolódik hozzá, hogy a kettő szinte elválaszthatatlan egymástól.

2. fejezet

A konkrét feladat és a vázlatos megoldási terv

A konkrét feladat egy olyan intelligens ágens tervezése, amely egy környezetétől eltérő színű, kör alakú tárgyat képes követni. Ezt úgy valósítja meg, hogy a képalkotó középpontját az alatta levő motor mozgatásával a köralakzat középpontjára pozicionálja. Mindezt úgy hajtja végre, hogy az algoritmus futási ideje megfelelő, az ágens működése látványos legyen. (Tehát másodpercenként legalább 3-4 képet legyen képes feldolgozni, és a motor pozicionálását véghezvinni.) Mivel ez az ágens fizikailag is megvalósul, ezért joggal nevezhető robotágensnek.

A megoldáshoz először meg kell nevezni az ágens érzékelő és beavatkozó szerveit. Egyetlen érzékelő szerve egy kamera, amivel a külvilágot érzékeli. Beavatkozó szerve pedig egy motor, feladata a kamera mozgatása. A kamera közvetlenül kapcsolódik egy központi egységhez, ami a szükséges számításokat végzi és futtatja az ágensprogramot. Az így készített kameraképet fel kell dolgozni és ez alapján kell szabályozni a motort a kívánt irányba. A motor vezérléséhez egy jelgenerátorra is szükség lesz, ezt a kapcsolásban a központi számító egység és a motor közé kell elhelyezni.

Szoftver oldalról első lépés a kamerakép mentése. Ezután a kapott képen el kell különíteni a kitüntetett, környezetétől eltérő színű objektumot az ágensnek haszontalan információktól. Majd az így kapott információhalmaz segítségével meg kell határozni az objektum középpontját. A kamera középpontja és a körközéppont közti különbség és a kamera látószöge alapján már kiszámolható az eltérés szöge. Az így kapott szöggel be lehet állítani a kiszámolt pozícióba a motort közvetett módon a jelgenerátor segítségével. Kezdetben a motor középállásban, vagyis 90 fokban található. Ebből vonódik vagy ehhez adódik hozzá a szögkülönbség annak függvényében, hogy az aktuális állástól balra, illetve jobbra kell mozognia a motornak. Végül ez a lépéssorozat addig ismétlődik, amíg csak a tárgy látható a kamera képén.

3. fejezet

Elméleti háttér

Ebben a fejezetben minden, a megoldáshoz szükséges elméleti anyagrész szerepel. Absztrakt módon meghatározzuk a szükséges fogalmakat és algoritmusokat, ezek birtokában a feladat könnyebben behatárolhatóvá válik és nagyobb rálátást kapunk a problémakörre is.

Robotágensek már a 20. század közepétől fogva léteznek és azóta dinamikusan fejlődnek. Nem csak fizikai megvalósításuk fejlődött, hanem egyre ügyesebbek, hibatűrőbbek; egyszóval intelligensekké váltak az évek során.

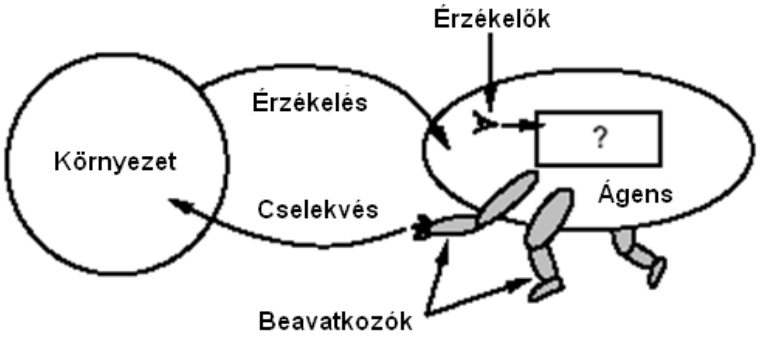
A robotágensek mindig rendelkeznek beavatkozókkal és érzékelőkkel. Érzékelőket a környezetük felfedezésére használják, beavatkozóikkal pedig befolyásolják azt. Így szerves részét képezik környezetüknek, képesek azt módosítani. Az érzékelők lehetnek külsők, illetve belsők egyaránt. Külső érzékelőkkel az ágens képes elhelyezni magát a világban, ebből információt nyerni, és felhasználni ezt a további cselekvésekhez. Belső érzékelők pedig megmutatják az ágens belső állapotát, amivel a vezérlés által okozott hibák csökkenthetők.

Ágensek különböző környezetekben tevékenykedhetnek. Ezek a környezetek nagyon sokfélék lehetnek. Általában egy ágenst egyfajta környezetre készítenek fel, ami nem mondható túlságosan intelligensnek, de ettől még a saját világában sikeresen működhet. Ha egy ilyen entitást áthelyezünk egy számára ismeretlen világba, nem képes relevánsan működni, mert csak a tervezője által beépített előzetes tudására hagyatkozik. Az ilyen „sérülékeny'' ágenseknél sokkal hibatűrőbbek és intelligensebbek azok, amelyek a beépített tudásukon túl képesek tanulásra is. Tanulás révén tapasztalatokat gyűjtenek és úgy módosítják tudásbázisukat, hogy az adott környezetben legjobb eredményeket érjék el.

3.1. Intelligens ágensek

A mesterséges intelligencia kutatások meghatározó részét képzik az intelligens ágensek. Definíció szerint egy ágens bármi lehet, amit úgy tekinthetünk, mint ami az érzékelői segítségével érzékeli a környezetét, és beavatkozói segítségével megváltoztatja azt. Ezt a koncepciót széles körben alkalmazhatjuk akár szoftveres akár hardveres ágensként minden elképzelhető környezetben. Amennyiben ágensünk jól végzi feladatát, tehát az ágens sikeres, joggal lehet őt nevezni intelligens ágensnek.

Egy robotágens például kamerákat és infravörös távolságérzékelőket használhat érzékelőként, és különféle motorokat beavatkozóként. Egy szoftverágens billentyűleütéseket, háttértáron tárolt adatokat, hálózati adatcsomagokat fogad érzékelőinek bemeneteként, és képernyőre kijelzéssel, fájlok írásá-



3.1. ábra. Az ágens felépítése ([2] 65. oldal).

val, hálózati csomagok küldésével avatkozik be a környezetébe.

Érzékelés az ágens érzékelő bemeneteinek leírása egy tetszőleges pillanatban. Egy ágens érzékelési sorozata az ágens érzékeléseinek teljes története, minden, amit az ágens valaha is érzékelt. Ha az összes lehetséges érzékélési sorozatához meg tudjuk határozni az ágens lehetséges cselekvéseit, akkor mindent tudunk az ágensről. Ezt matematikailag az ágensfüggvény felírásával tudjuk megfogalmazni, ami az adott érzékelési sorozatot egy cselekvéssorozatra képzi le. Egyszerűbb ágensek esetében táblázatos formában is megadható az ágensfüggvény, de legtöbb esetben ez nem célszerű, mivel nagyobb feladatokat megoldó ágensek táblázatos formában megadott ágensfüggvénye túlságosan nagy lenne.

Az ágensfüggvény számítógépes megvalósítását ágensprogramnak nevezzük. Az ágensfüggvény egy absztrakt matematikai leírás és ennek a konkrét implementációja az ágensprogram.

3.1.1. Racionalitás

Ágensünk az intelligensségen kívül még rendelkezhet más jellemzőkkel is. Ilyen a racionalitás. Egy olyan ágenst nevezünk racionális ágensnek, amely helyesen cselekszik. Ez táblázatos formában megadott ágensfüggvény esetén azt jelenti, hogy a táblázatban minden bejegyzés helyesen van kitöltve. A helyes cselekedet az, amely az ágenst legsikeresebbé teszi. Tehát az az ágens, amely racionális, egyben intelligens is. A sikeresség mérésére szükség van egy olyan módszerre, amivel meghatározható az ágens sikeressége. Ehhez szükség van az ágens érzékelőinek, beavatkozóinak, valamint a környezetének leírására, amivel már kiszámolható, hogy milyen mértékben specifikálja feladatát.

3.1.2. A teljesítmény mérése

A teljesítménymérték valósítja meg egy ágens sikerességének kritériumát. Amennyiben egy ágenst elhelyezünk egy környezetben, akkor cselekvések sorozatával válaszol az érzékeléseire. Ezen cselekvések sorozatának hatására környezete állapotainak sorozatán halad végig. Ha a kapott értékek megfelelőek számunkra, akkor az ágens jól teljesített. Ezt a teljesítménymértéket az ágens tervezője határozza meg. Ökölszabályként a teljesítménymértéket legjobb aszerint megállapítani, hogy mit akarunk elérni a környezetben, mint aszerint, hogy miképp kellene az ágensnek viselkednie.

A következő kritériumokon múlik, hogy az ágensünk az adott pillanatban racionális-e:

- siker fokát meghatározó teljesítménymérték,
- ágens eddigi tudása a környezetről,
- azok a cselekvések, amiket az ágens képes végrehajtani,

- az ágens adott pillanatig tartó érzékelési sorozata.

Ezek alapján már megadható a racionális ágens definíciója: az ideális racionális ágens minden egyes észlelési sorozathoz a benne található tények és a beépített tudása alapján minden elvárható dolgot megtesz a teljesítménymérték maximalizálásáért.

3.1.3. Mindentudás, tanulás, autonómia

Abban az esetben, ha az ágensünk racionálisan viselkedik, még nem feltétlenül rendelkezik mindentudással. Így a kettőt külön kell választani. Egy mindentudó ágens tudja cselekvései valódi kimenetét, és ennek megfelelően cselekedhet. Ezt a gyakorlatban lehetetlenség megvalósítani. Egy példán keresztül ez könnyen belátható. Vegyünk azt az esetet, hogy át szeretnénk menni az utca egyik oldaláról a másikra. Ilyenkor szokás szerint először balra, utána jobbra tekintünk, hogy meggyőződjünk arról, jön-e valamilyen jármű az egyik, illetve másik oldalról. Ha nem látunk senkit akkor átkelünk, de egy váratlan pillanatban ránk zuhan egy repülőgép. Ebben az esetben racionálisan viselkedtünk, hiszen egyik irányból sem jött jármű, tehát szabad volt az átkelés. De nem rendelkeztünk mindentudással, ezért mégsem volt helyes a cselekvésünk. Így könnyen beláthatjuk, hogy a racionalitás az elvárt teljesítményt maximalizálja, míg a tökéletesség a tényleges teljesítményt. A racionalitás nem követeli meg a mindentudást, hiszen az csak az adott pillanatig felépített érzékelési sorozattól függ.

Az ágensnek a racionalitáson túl információt kell gyűjtenie a környezetéről, valamint ezekből amennyit csak lehet, tanulnia is kell. Előzetesen rendelkeznie kell valamilyen beépített tudással, de ahogy tapasztalatot szerez, ez a tudás átértékelődhet, módosulhat. Szélsőséges esetben ez annyira pontos tudás, hogy működés közben nem is kell változtatni rajta. Viszont az ilyen ágensek nagyon sérülékenyek.

Legtöbbször egy sikeres ágens esetén három részre szokás bontani az ágensfüggvény kiszámításának feladatát. Első eset, amikor az ágenst tervezik és a tervezője kiszámolja. Másodszor, amikor megfontolja a következő cselekvést. Végül, amikor tanul a tapasztalataiból, még további számításokat végez annak érdekében, hogyan módosítja a viselkedését.

Az ágenst abban az esetben nevezzük nem autonómnak, ha csak a beépített tudására támaszkodik. Viszont ha racionális ágenst szeretnénk készíteni, akkor annak autonómnak kell lennie. Tehát mindent, amit lehet, meg kell tanulnia annak érdekében, hogy a hibás cselekvéseinek számát minimalizálja. Autonóm ágenseknek viszont ugyanúgy kell előzetes, beépített tudás. Amennyiben ezzel nem rendelkezik, véletlenszerűen kell cselekednie, és hosszabb ideig tart a környezet kiismerése. Ezért biztosít a tervező előzetes tudást. Ha már elegendő tapasztalatot szerzett az ágens a környezetről, viselkedése gyakorlatilag függetlenné válik a beépített előzetes tudástól. Így a tanulás révén olyan ágens tervezhető, ami sokféle környezetben sikeres működést mutat.

3.1.4. Környezetek típusai

Környezetek lényegében problémák, amelyekre a racionális ágensek jelentik a megoldást. Ezen környezetek típusa közvetlenül befolyásolja az ágensprogram tervezését. A lehetséges környezetek száma hatalmas, de ezeket be lehet kategorizálni, amelyek az ágens tervezésekor nagy segítséget jelenthet annak tervezőjének. Ezek a kategóriák a következők:

Teljesen megfigyelhető vagy részlegesen megfigyelhető. Ha az ágens szenzorai minden pillanatban hozzáférést nyújtanak a környezet teljes állapotához. Ekkor azt mondjuk, hogy a környezet

teljesen megfigyelhető. Viszont ha a szenzorok adatai pontatlanok, zajosak, a környezetet részlegesen megfigyelhetőnek nevezzük.

Determinisztikus vagy sztochasztikus. Amennyiben a környezet következő állapotát jelenlegi állapota és az ágens által végrehajtott cselekvések teljesen meghatározzák, akkor a környezet determinisztikus. Ellenkező esetben pedig sztochasztikus.

Epizódszerű vagy sorozatszerű. Egy epizódszerű környezetben az ágens tapasztalata epizódokra, elemi részekre bontható. Ilyen epizódok egy érzékelést és a hozzátartozó egy cselekvést tartalmazzák. Ezek az epizódok viszont nem függnek az előzőekben végrehajtott cselekvésektől, tehát csak a pillanatnyi állapot a fontos. Sorozatszerű környezetnél pedig az aktuális cselekvés befolyásolhatja a további döntéseket.

Statikus és dinamikus. Ha a környezet megváltozik mialatt az ágens gondolkodik, akkor azt a környezetet dinamikusnak nevezzük. Ellenkező esetben statikus környezetről beszélünk. Egy statikus környezettel könnyű bánni, mivel az ágensnek nem kell folyamatosan a világot figyelnie, miközben kiválasztja az adott érzékeléshez a megfelelő cselekvést. Dinamikus esetben az ágens dolga jóval nehezebb.

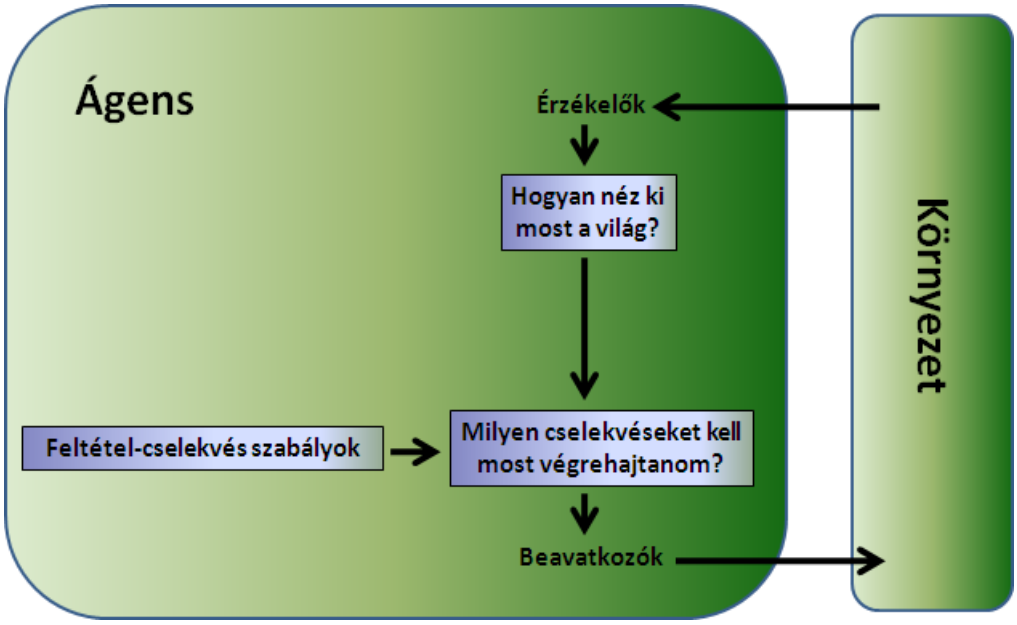
Diszkrét vagy folytonos. Ez a felosztás alkalmazható a környezet állapotára, az időkezelés módjára, az ágens észleléseire és a cselekvéseire. Például a diszkrét eset lehet egy sakkjáték, ami véges számú állapottal rendelkezik. Egy autonóm városban közlekedő jármű esetében folytonos esetről beszélhetünk.

Egyágenses vagy többágenses. Ennek eldöntése bizonyos esetekben könnyű, viszont léteznek olyan környezetek amikor nehéz eldönteni, hogy egy vagy többágenses környezetben kell ágensünknek teljesítenie. Egyszerű eset lehet például egy sakkozó ágens. Ilyenkor tudjuk, hogy a környezet kétágenses, két szereplős játékról lévén szó. Amennyiben viszont az előbb említett városban közlekedő ágensünkről van szó, nehéz eldönteni, hogy például az akadály, amit ki akar kerülni, egy tereptárgy vagy egy másik autó, tehát egy másik ágens. Többágenses esetben felmerülhetnek más fogalmak is. Ezek a versengés (kettő vagy több ágens egymás ellen dolgoznak), kooperativitás (egymással együtt dolgozva érhető el minden egyes ágens maximális teljesítménymértéke) valamint a kommunikáció (az együtt működéshez szükséges, a többi ágenssel való kapcsolattartás).

3.1.5. Az ágensek típusai

A mesterséges intelligencia egyik feladata olyan ágensprogramok megvalósítása, amelyek nagy táblabejegyzések helyett kisméretű programkóddal is képesek megvalósítani a racionális viselkedést. Következőkben felsorolt ágenstípusok ugyanazzal a vázzal rendelkeznek: a szenzorokból kapott aktuális észlelések alapján visszaküldenek egy cselekvést a beavatkozókhoz.

Egyszerű reflexszerű ágensek. Ágensfajták közül ez a legegyszerűbb. Az aktuális észlelés alapján választják ki a cselekvéseket, figyelmen kívül hagyva az észlelési történet többi részét. Ha az egyik feltétel teljesül, akkor a hozzá tartozó szabályt végrehajtjuk. Ez a fajta ágens csak akkor fog helyesen működni, ha kizárólag az aktuális észlelés alapján hozható meg a helyes döntés, azaz, ha a környezet teljesen megfigyelhető.



3.2. ábra. Egyszerű reflexszerű ágens felépítése ([2] 74. oldal).

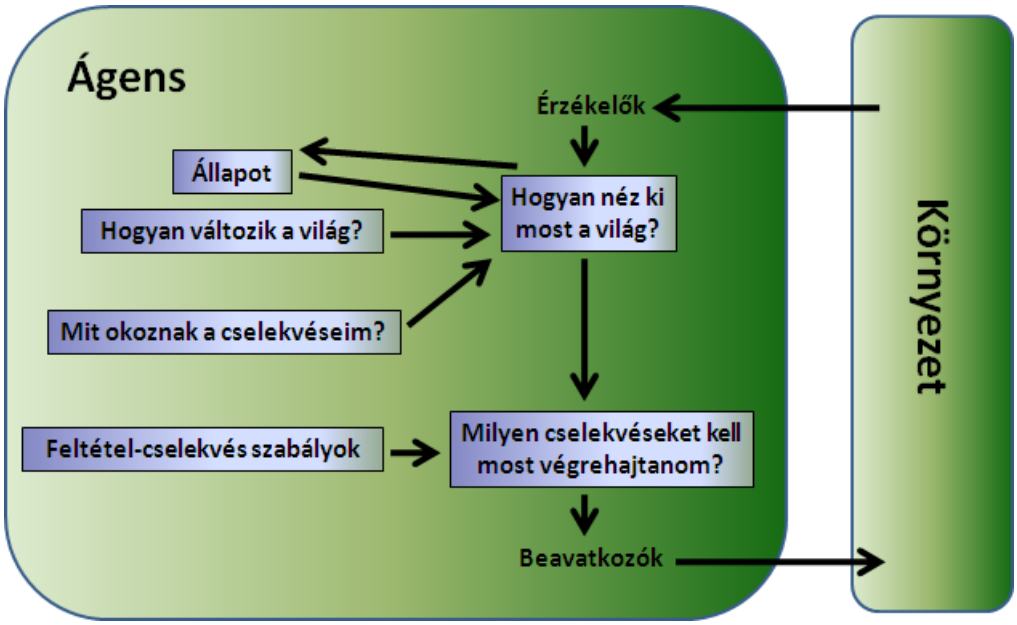
Modellalapú reflexszerű ágensek. A minél részletesebb megfigyelhetőség egyik leghatékonyabb módja, ha az ágens nyomon követi a világ jelenleg nem látható részét. Ez azt jelenti, hogy rendelkezik valamiféle belső állapottal, amely az eddig észleltek alapján jön létre. Így a jelenlegi állapot nem megfigyelt részeinek legalább egy részét tartalmazza az aktuális állapot. Ezt az állapotot folyamatosan frissíteni kell és ehhez két dologra van szükség. Először is kellenek olyan információk, hogy hogyan változik a világ az ágenstől függetlenül. Másodszor pedig szükség van olyan információkra is, amik megmondják, hogy az ágens cselekvései hogyan változtatják meg a környezetét. A világ működésének leírását a tervező határozza meg, ezt a világ modelljének hívjuk. Az ilyen modellt használó ágenst pedig modellalapú ágensnek nevezzük.

Célorientált ágensek. A környezet jelenlegi állapotának ismerete nem mindig elegendő annak eldöntéséhez, hogy mit kell tennie az ágensnek. A jelenlegi állapot mellett az ágensnek kell egy célinformáció is, ami a kívánatos helyzeteket írja le. Az ágensprogram ezt összeveti a lehetséges cselekvésekkel annak érdekében, hogy a célhoz megfelelő cselekvést megválassza.

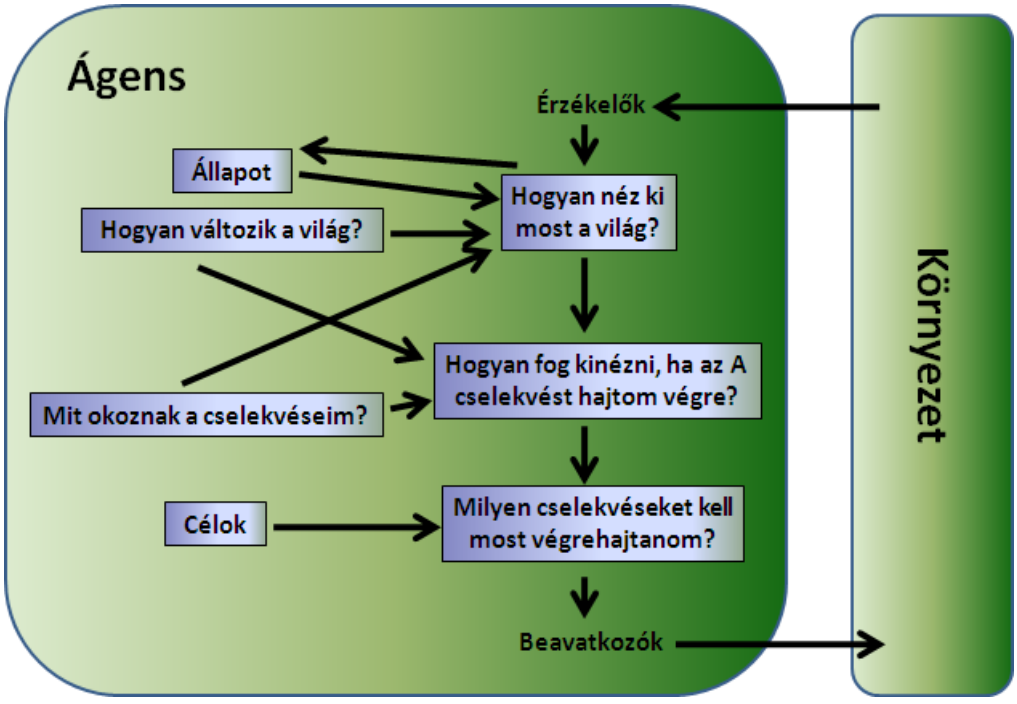
Vannak olyan egyszerű esetek, amikor a cselekvésválasztás egyszerű, amikor a cél eléréséhez elegendő egyetlen cselekvés. Viszont lehetségesek olyan esetek is, amikor zsákutcába jut az ágens és onnan kell visszafelé haladva célját elérni. Ehhez már valamiféle keresési algoritmusra vagy tervkészítésre van szüksége az ágensnek.

Célorientált ágensek sokkal rugalmasabbak a reflexszerű ágensekhez képest, mert új környezetbe helyezés esetén nem kell újraírni a feltétel-cselekvés szabályokat.

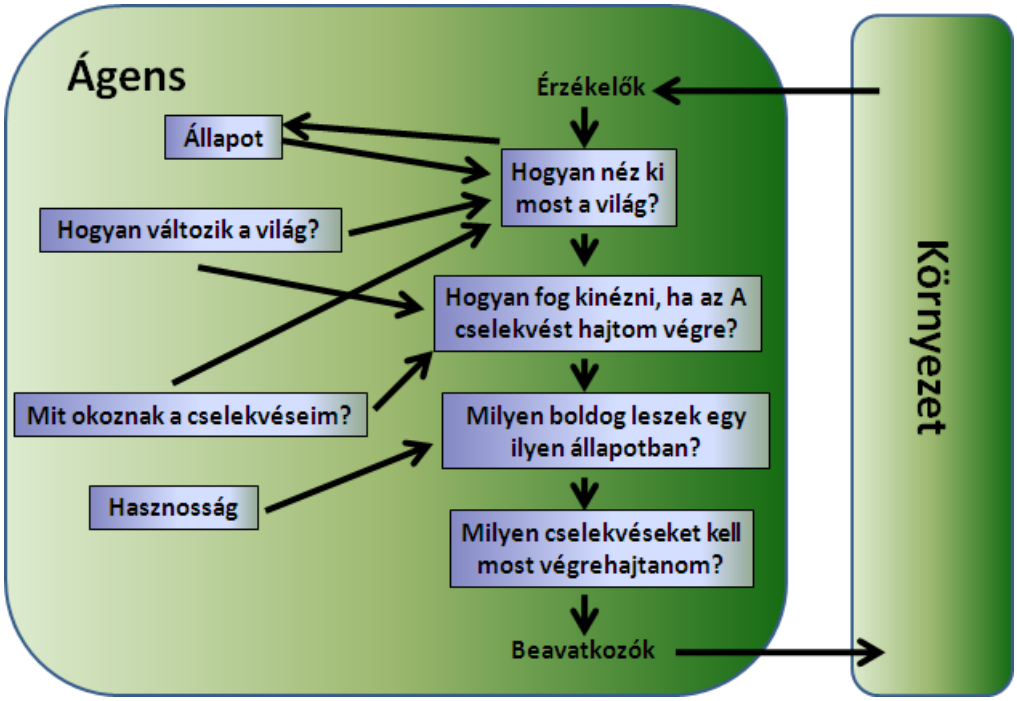
Hasznosságorientált ágensek. Ágenskörnyezetek túlnyomó többségében a célok magukban nem elegendőek a jó minőségű viselkedések létrehozásához. A világ egyik állapota akkor előnyösebb egy másikhoz képest, ha nagyobb a hasznossága az ágens számára. Ennek megállapítására először meg kell határozni a hasznosságfüggvényt, ami az adott állapothoz leírja a hasznosság fokát. Hasznosságfüggvénnyel kétféle helyzetben is képes lesz az ágensünk racionális döntésekre. Egyrészt, amikor egymásnak ellentmondó célok vannak (például a gyorsaság vagy a biztonság a fontosabb),



3.3. ábra. Modellalapú reflexszerű ágens felépítése ([2] 76. oldal).



3.4. ábra. Célorientált ágens felépítése ([2] 77. oldal).



3.5. ábra. Hasznosságorientált ágense felépítése ([2] 79. oldal).

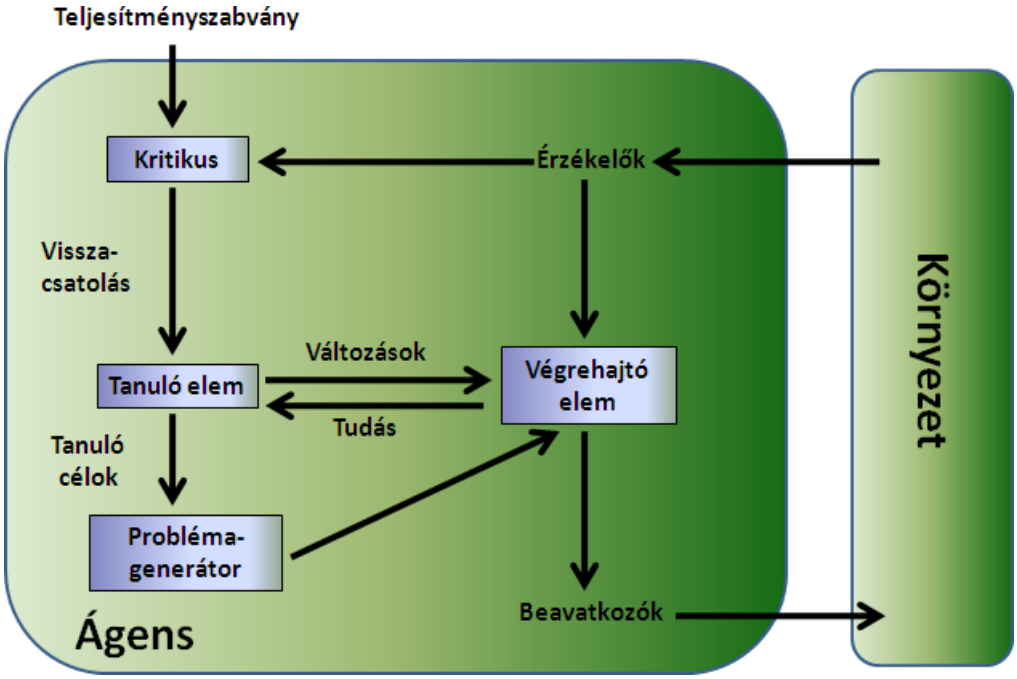
képes kompromisszumos döntések létrehozására. Másrészt, amikor több cél van, és egyikük sem érhető el teljes biztonsággal.

Tanuló ágensek. Egy tanuló ágens négy egymástól elkülöníthető részre bontható. Rendelkezik egy tanuló elemmel, ami a javításokért felelős, valamint a külső cselekvések kiválasztásáért felelős végrehajtó elemmel. A végrehajtó elem az, amit eddig a teljes ágensnek tekintettünk (végzi az érzékelés és ez alapján a cselekvést). A tanuló elem a kritikustól kapott, az ágens működéséről szóló visszajelzéseket használja fel arra, hogy a végrehajtó elemet hogyan kell módosítani, hogy a jövőben sikeresebben működjön.

A tanuló elem tervezése nagy részben függ a végrehajtó elem tervezésétől. Olyan ágens esetében, ami képes megtanulni bizonyos képességeket, azt kell meghatározni milyen teljesítményelemre van szüksége az ágensnek a végrehajtáshoz, ha már azt megtanulta. Tanuló eljárások az ágens minden részéhez készíthetők.

A kritikus azt mondja meg a tanuló elemnek, hogy az ágens milyen jól működik egy rögzített teljesítményszabályhoz képest. Erre szükség van, mert az észlelések maguktól nem jelzik az ágens sikerességét. A teljesítménymértéknek rögzítettnek kell lennie és ez mindig az ágensen kívül helyezkedik el, hiszen ezt nem módosíthatja saját viselkedésének megjavítása érdekében.

Problémagenerátorra is szükség lesz, aminek a feladata, hogy olyan cselekvéseket javasoljon, amelyek új és hasznos tapasztalatokhoz vezetnek. Ha a végrehajtó elem csak az adott tudására támaszkodna, mindig a legjobb cselekvéseket választaná ki. Viszont ha az ágens hajlandó egy kis felfedezésre rövid távon nem biztos, hogy mindig jól döntene, de hosszútávon sokkal jobb cselekvéseket fedezhet fel. A problémagenerátor feladata ilyen felfedező cselekvések javaslása.



3.6. ábra. Tanuló ágens felépítése ([2] 88. oldal).

3.2. Robotika

Az ágensek absztrakt világából most térjünk át egy szűkebb területre, a robotikára. A robotika fizikai ágensekkel foglalkozik, a csak tisztán szoftveres ágensekkel nem, ezért ez a terület az ágensek csak részhalmazát jelenti. Mivel az általam készített ágens hardveresen is megvalósult, azért joggal nevezhető robotágensnek. Az ilyen ágensek ágensprogramból és architektúrából állnak. Ágensprogramról már tudjuk, hogy az ágensfüggvény implementációja, az architektúra pedig az a fizikai érzékelőkkel és beavatkozókkal ellátott számítóeszköz, amin ez a program fut. Ebben a fejezetben a robotikáról, robotágensekről és e témában elért eredményekről lesz szó.

3.2.1. Robotika múltja, jelene és jövője

Robotok fejlesztése abból az ősi igényből adódik, hogy az ember minél jobban szeretné saját erőfeszítéseit csökkenteni. Ehhez először eszközöket készített a munka megkönnyítése érdekében, napjainkban pedig az ipari robotok fejlesztésével próbáljuk a legnehezebb és legköltségesebb munkafolyamatokat gépekre áthárítani. A mai értelemben vett robotok ipari alkalmazásának története az 1960-as években vette kezdetét. Azóta a munkafolyamatok automatizálása és a gépek intelligenciája jelentősen és egymással párhuzamosan nőttek.

A gépiesítés első lépése az emberi munka részfeladatai elvégzésének kiváltása volt. Ehhez célgépeket, primitív automatákat készítettek. Ilyenek például az automatizált gyártósorok, aminek a következménye a költségcsökkenés és a termelékenység megugrása. Mind több és több ilyen gyártósor létrehozása után felmerült az igény, hogy ezek a célgépek legyenek intelligensebbek, így a konkurenciánál nagyobb termelékenységet és a profitot lehetne elérni. Ezen igényekre választ adva jelent meg a robottechnika tudománya, aminek a feladata a kezdeti primitív gépektől eltérően a fejlett, intelligens robotok létrehozása. A mai ipari gyártórendszerek legintelligensebb gépei az ipari robotok.

A mai ipari robotok története Joseph Engelberger és George C. Devol amerikai irányítástechnikai és

elektronikai szakértők nevéhez fűződik. Az első ipari robot az 1960-as években Ultimate néven került forgalomba. Ez idő tájt a robotokat a nehéz fizikai munkafolyamatok emberi kiváltására, főleg anyagmozgatásra használták. Második nagy fejlődési fázis a 70-es években valósult meg. Ekkor már a japán és európai cégek is beléptek az iparágba. Felhasználási terület is bővült ezáltal: fúrásra, ívhegesztésre, marásra, illesztési feladatok végrehajtására használták a robotokat. Harmadik fázisként a robotok intelligenciáját is elkezdték fejleszteni. Valamint az egyenáramú és léptető motorok felhasználásával a fizikai megvalósítás is egyszerűsödött. Megjelentek az első robotprogramozási nyelvek. Így a felhasználási kör kibővült a laboratóriumi, mélytengeri, űrkutatásbeli, mezőgazdasági és szerelési folyamatokkal. Napjainkban pedig tipikusan minden olyan helyen fellelhetők, ahol az emberi élet fenntartása extrém magas költségbe kerülne (űrkutatás, mélytengeri kutatások, szennyező anyagok kezelése, stb.). Jövőben egyre jobban fognak szélesedni felhasználási területeik, az emberi élet és munka legkülönbözőbb részein jelennek meg. Ilyennek tekinthetők azok a próbálkozások, amik arra irányulnak, hogy önműködő, sofőr nélküli járművet hozzanak létre, amely képes a városi forgalomban helytállni. Ezen a Grand Challange nevű versenyen a csapatok évről évre egyre kifinomultabb és megbízhatóbb járművekkel állnak elő. A közeljövőben pedig bizonyára sikerül olyan autonóm járművet fejleszteni, ami közel hibamentesen, relevánsan képes ezt a komplex feladatot végrehajtani és akár tömeggyártásba is kerülhet majd ez a megoldás.

3.2.2. Robotok irányítása

A robotok irányításához először meg kell ismerkedni a robotikában felhasznált szakterületeket:

- *Gépészet*: szerkezeti elemek, motorok tervezése, alkalmazása.
- *Informatika*: vezérlések megvalósítása, kommunikáció, stb.
- *Szenzortechnológia*:
  - Belső érzékelők: a robot belső állapotát mutatják meg (például: optikai közelségérzékelők)
  - Külső érzékelők: környezet állapotát figyelik (például: ultrahangos távolságérzékelő)
- *Irányítástechnika*: robot, mint fizikai rendszer modellezése.

A robotirányítás célja, minél robusztusabb irányítás létrehozása, ami kompenzálja az esetleges modellezési hibákat és a külső zavarokat. A robusztusság mint tulajdonság valamilyen minőségi tulajdonság, ami megmarad akkor is, ha a mennyiségi tulajdonságok folyamatosan változnak. Az irányítás minősége sokféleképpen megfogalmazható, de itt a pontos matematikai leírás nem előnyös, ugyanis ez csak komplikáltabbá tenné a vezérlést.

Az irányítás három fő rendszerből áll:

- A rendszer egészére előírt feladatnak a részrendszerekre kiróható feladatokra bontása.
- A kapott részfeladatok átadása a hozzájuk tartozó részrendszereknek.
- A részfeladatok végrehajtásának ellenőrzése (egyszerűbb rendszerek esetében ez elmaradhat).

A robotikában többféle vezérlési módszert is kifejlesztettek a motorok irányítására.

***Pontvezérlés (Pont To Point)*:** A robot számára csak a kívánt kezdeti, illetve végpont van megadva. A feladat, hogy a kezdőpontból a végpontba jussunk, a robotvezérlés tetszés szerint hajthatja végre. A felhasználó által a robot mozgásának pályagörbéje ellenőrizhetetlen a két pont között. Itt csak arra van lehetőség, hogy hosszabb szakaszokat rövidebbekre osszunk fel.

***Pályavezérlés (Continuous Path)*:** A vezérlőegység nem csak a végpont koordinátáit adja meg, hanem kiszámolja a pályagörbe egyenletes időközönkénti felosztását és a hozzá tartozó koordinátaállásokat is.

***Sebességszabályozás (Resolved Motion Rate Control)*:** A pályavezérlés tovább finomítható, ha nem csak azt határozzuk meg, hogy egy időpillanatban hol kell tartózkodnia a robotnak, hanem azt is, e pont környékén milyen sebességgel kell haladnia.

***Gyorsulásszabályozás (Resolved Acceleration Control)*:** Továbbfejlesztve a sebességszabályozást arra is kell ügyelni, hogy a pályagörbe a lehető legsimább legyen.

Látható, hogy a gyorsulásszabályozás valósítja meg a legmegbízhatóbb és legkifinomultabb irányítást. Viszont a többletinformációk miatt a sávszélesség növelésével, valamint a központi számítóegység nagyobb leterheltségével is számolnunk kell. Bármelyiket is választjuk a négy vezérlés körül, tudnunk kell, milyen feladatra szeretnénk használni. Egyszerűbb, hibát jobban megengedő rendszerek esetén elegendő a pontvezérlés, viszont precíziós feladatoknál már nem megengedhető a hibák elhanyagolása. A feladat, számítási képesség és sávszélesség alapján kell mérlegelni, milyen vezérlési technikát választunk egy adott problémához.

3.3. A robotok vizuális érzékelési rendszere

A robotágensek vizuális érzékelőkkel is rendelkezhetnek, amikkel megvalósítják vagy kiegészítik érzékelési sorozatukat. Ezekkel az érzékelőkkel részletesebb képet kapnak környezetükről, ezáltal sikeresebbé válnak. Természetesen az így kapott többletinformációt értelmezni kell, ami esetenként költséges számításokat von maga után. Ehhez nyújt segítséget a képfeldolgozás, gépi látás és az alakfelismerés témakörei. A fejezet célja bemutatni a vizuális érzékelés alapjait.

3.3.1. A vizuális érzékelés folyamata

A mesterséges látás információ feldolgozás, ahol a bemenet egy digitális kép, kimenet pedig a képen található objektumok egy leírása. Ez a bementi és kimeneti kapcsolat jól látható a 3.7-es ábrán.

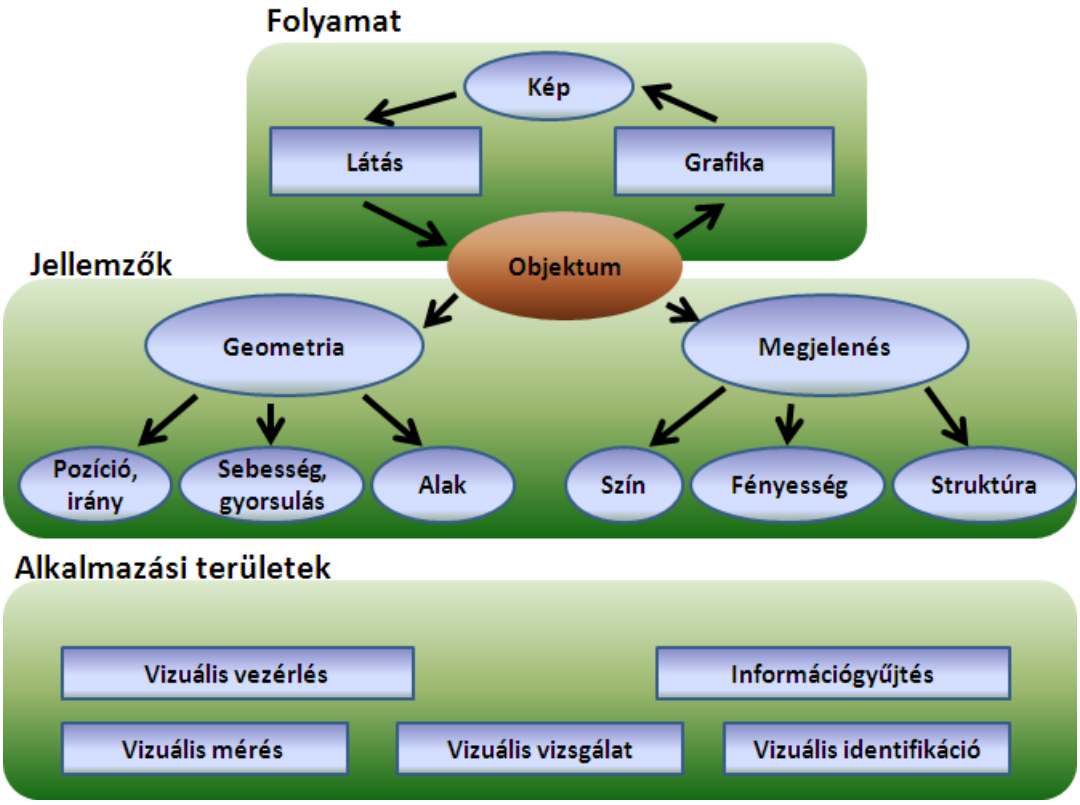
A 3.7-es ábra legfelső részében két, egymással inverz művelet helyezkedik el: a látás és a grafika. Grafika esetében pontosan definiált objektuminformációk alapján keletkezik a pontos képi megjelenítés. Látás esetében pedig a képi adatok segítségével következtetünk az objektumokat leíró információkra. Az így kapott objektumok rendelkeznek geometriai és megjelenési jellemzőkkel, amikkel már elegendő információt tudunk biztosítani a vizuális érzékelésen alapuló rendszerek többi egységének.

3.3.2. A vizuális érzékelés jellemzői

Az általunk érzékelt objektumok a következő jellemzőkkel rendelkezhetnek:

- *Fizikai jellemzők*: szag, szín, merevség, elektromos vezetőképesség, hő-vezetőképesség, stb.
- *Geometriai jellemzők*: alak, pozíció és irány, sebesség és gyorsulás

Egy objektum megjelenése tartalmazza a szín, fényesség és textúra (szín és/vagy fényesség valamilyen eloszlása) tulajdonságokat. Ezek az információk kombinálva vannak geometriai jellemzőkkel, melyek igen hasznosak a vizuális érzékelésben, mert egy objektum legfontosabb jellemzői e tulajdonságok.



3.7. ábra. Vizuális érzékelés folyamata, jellemzői és alkalmazási területei ([1] 446. oldal).

Egy kép mindig a tér egy projekciója a kamera képének síkján, tehát a háromdimenziós teret rögzítjük egy kétdimenziós képen. Viszont így, a térbeli koordináta elhagyásával a geometriai tulajdonságok becslése is nehézkessé válik, a kétdimenziós kép nem hordoz minden szükséges információt. Mivel a rögzített kép csak megjelenési tulajdonságokat hordoz, ezekből az információkból kell következtetnünk a geometriai jellemzőkre. Ezért a vizuális érzékelés első lépése mindig egy képfeldolgozás és a jellemzők becslése. Ezek segítségével számításokat végzünk a szín, fényesség és textúra információhalmazon azon célból, hogy minél több hasznos geometriai tulajdonságot kapjunk.

3.3.3. A vizuális érzékelés felhasználási területei

Az előző részben láthattunk, hogy a vizuális érzékelés képes információt biztosítani a digitális képen rögzített térről. Ezt a képességet rengeteg helyen lehet hasznosítani, akár humanoid robotok fejlesztésénél, amivel nagyobb autonómiára tehetnek szert és biztonságosan működhetnek egy dinamikusan változó környezetben. Felhasználási területei a vizuális érzékelésnek:

Vizuális vezérlés. A járás, manipulálás és fogás, a három leggyakoribb emberi tevékenység. A vizuális érzékelésünk biztosítja, hogy megfelelően végezzük ezeket a cselekvéseket. Egy humanoid robotnál ilyen mozgások esetén elengedhetetlen, hogy a vizuális érzékelés szorosan kapcsolódjon a végrehajtott műveletekkel. Ezért az irányítás szempontjából az a legjobb, ha a robot minél többször hasznosítja vizuális érzékelési képességét, így a végrehajtandó mozdulatsort is precízebben meg lehet határozni.

Információgyűjtés. Mentális és fizikai képességeinket fejleszteni tudjuk a dinamikusan változó környezetünkkel folyó folyamatos interakcióval. Egy fontos szempontja a mentális fejlődésnek az információgyűjtés, ami alapvetően a természetes nyelvek értelmezéséből ered. Vizuális érzékelésünk segítségével képesek vagyunk írott szövegek értelmezésére és ebből kifolyólag rögzített szimbólumok által tudunk asszociálni a jelentésre is. Egy humanoid robot ugyanígy képes lehet fejleszteni saját tudását nyelvek értelmezésével, vizuális érzékelésének köszönhetően.

Vizuális vizsgálat. Az iparban fellelhetők olyan vizuális vizsgálatot végrehajtó alkalmazások, melyek alapvető feladata a minőség-ellenőrzés. Előre specifikált tudás alapján kell eldönteniük egy termékről, hogy az megfelel-e a követelményeknek. Az ilyen termékeknek lényeges részét képzik a megjelenési és geometriai információk. Így az automatikus vizuális érzékelés felhasználásával ezen eszközök működését képesek vagyunk tökéletesíteni.

Vizuális identifikáció. A geometriai és megjelenítési jellemzők hasznos információt nyújtanak az azonosítás folyamatához, mint például személyazonosság, emóció és nyelvi szimbólumok felismerése. Két ilyen tipikus azonosítási feladat az ujjlenyomat- és az arcfelismerés.

Vizuális mérés. A megjelenés és színek érzékelése az emberi látásban fiziológiai és pszichológiai értelmezései a fénynek. Továbbá a szín érzékelése függ a megvilágítási körülményektől is. Másrészről az emberi látás képtelen pontosan megadni egy tárgy méreteit, csak becsléseink lehetnek a tényleges méretekről. Ez egy mesterséges látást alkalmazó robot esetén másképp van, képes pontos képet kapni mind a színmértékekről, mind pedig a pontos alaki jellemzőkről. Így az ilyen robotok ezen a téren előnyben vannak a hagyományos emberi vizuális érzékeléssel szemben.

3.3.4. Információfeldolgozás a vizuális érzékelésben

A világról rögzített kép a robot vizuális szenzorában három darab, kétdimenziós mátrixban tárolódik a következőképpen:

$$I_R = \{r(u,v),\ 1 \leqq v \leqq r_y \ és\ 1 \leqq u \leqq r_x\}$$

$$I_G = \{g(u,v),\ 1 \leqq v \leqq r_y \ és\ 1 \leqq u \leqq r_x\}$$

$$I_B = \{b(u,v),\ 1 \leqq v \leqq r_y \ és\ 1 \leqq u \leqq r_x\}$$

ahol r_x a kép szélessége és r_y a kép magassága

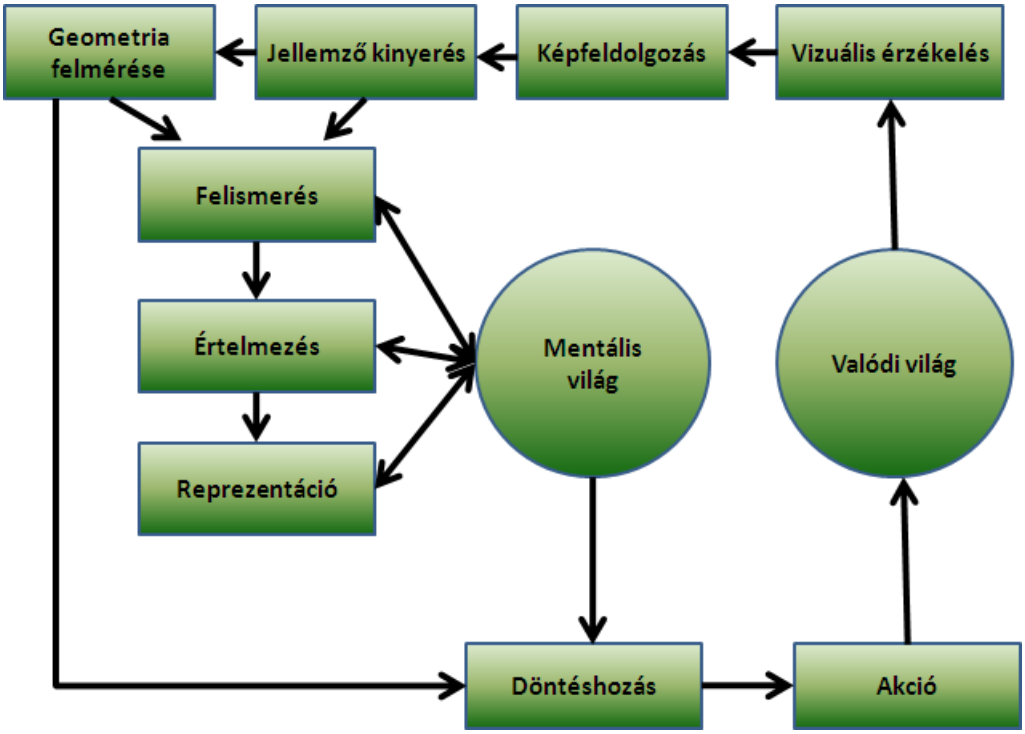
Ezek az összetevők reprezentálják a piros – zöld – kék komponenseket a színes képeken. Továbbá ezeken megtalálható a szín, fényerősség információk is. Utóbbi a súlyozott RGB[1] komponensből számolható:

$$I(v,u) = 0.3 * r(v,u) + 0.59 * g(v,u) + 0.11 * b(v,u)$$

ahol $1 \leqq v \leqq r_y$ és $1 \leqq u \leqq r_x$

Láthattuk, hogy a rögzített pillanatkép hordozza a megjelenési információkat, de ezek még nem adnak elegendő információt az objektumokról. További geometriai információkra van szükségünk a képen található objektumokról, amikre a szín és fényesség komponensekből következtetünk. Az objektumok geometriájának feltérképezésére először képfeldolgozást használunk, majd az objektum jellemzők kinyerése, végül a geometria felmérése következik.

[1] Red - Green - Blue



3.8. ábra. Információfolyam a vizuális érzékelési rendszerben ([1] 451. oldal).

Képfeldolgozás. A digitális képen explicit módon vannak meghatározva a színre, fényességre és szerkezetre vonatkozó információk. Viszont rejtett információkat is tartalmaznak: uniformitás (egységesség), kontinuitás (folyamatosság) és diszkontinuitás (megszakadás). A képfeldolgozás feladata e rejtett jellemzők feltérképezése.

Objektum jellemzők kinyerése. Objektumhoz tartozó jellemzők szorosan kapcsolódnak az uniformitáshoz, a kontinuitáshoz és a diszkontinuitáshoz. Ezek a tulajdonságok az objektumok jellemzőitől és szín információktól függenek. Az objektumok jellemzőinek meghatározása a további lépésekhez elengedhetetlen.

A geometria felmérése. A geometria felmérése során a térbeli információra szeretnénk következtetni, ezért fel kell derítenünk a kapcsolatot a kétdimenziós digitális kép és a háromdimenziós világ között. Viszont a z irányú koordináták nem rögzülnek a képen. Ezen információ megbízható meghatározása a mai napig nem tisztázott.

Objektum felismerése. Objektumok felismerése azok azonosítását jelentik a geometriai és megjelenési információk alapján. Ez amennyire könnyű egy ember számára, annyira nehéz egy robotnak. Mivel az azonosítás fontos részét képezi a vizuális érzékelésnek, ezért a folyamat során elfogadható becsléseket próbálunk tenni a kameraképen rögzített objektumokra.

A kép értelmezése. A sikeres képértelmezés függ a robot képességeitől, objektumokat hogyan tudja felismerni és leírni azokat szemantikai szinten.

Tudásreprezentáció. A természetben, az ismeretreprezentáció alapjait a nyelvi leírások képezik. Ennek következtében a vizuális érzékelőrendszer fontos a tudásreprezentáció számára. Vizuális érzékelő rendszer segítségével folyamatos kapcsolatot tud biztosítani a robot a környezetével. Eredményként nyelvi leírásokat vagy másképpen interpretációkat kapunk a külső világról, ami nagyban hozzájárul a robot mentális és virtuális világának kialakításához.

4. fejezet

Ágensarchitektúra

A fejezet tartalmaz minden a robottal kapcsolatos hardvereszközt, amik a robotágens architektúráját képzik. Az architektúra részei a robotágens érzékelője és beavatkozója, valamint az ágensprogramot futtató számítógép és a motorvezérléshez szükséges jelgenerátor.

4.1. Kamera (Érzékelő)

Az ágensarchitektúra egy Logitech gyártmányú Quickcam E3500-as webkamerával van felszerelve, ami közvetlenül a motor felett helyezkedik el. Ez a vizuális érzékelő fogja segíteni a robotágenst a környezetének kiismerésében, a vele rögzített pillanatkép alapján fog történni a motor szabályozása.

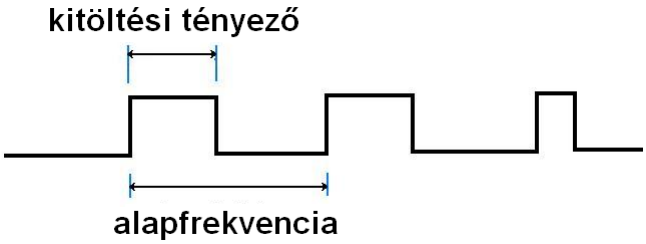
4.1. ábra. Ágens érzékelője.

4.2. Szervomotor (Beavatkozó)

A robotágens beavatkozója egy Modelcraft gyártmányú RS-2 szervomotor. A szervomotorok alapvetően pozicionálási feladatokra lettek kitalálva, két irányba képesek mozogni és négyszögjelek kitöltési tényezőjével állítható be a kívánt pozíció. Minden pozíció, amit képes felvenni, hozzá van rendelve egy kitöltési tényezőhöz. Így a szervomotor szabályzásához csak a kitöltési tényezőn kell változtatni.

A felhasznált motor 0-tól 180 fokig képes egy adott szögbe beállni. Működtetéséhez 5V egyenáramú tápfeszültség szükséges, szabályzáshoz pedig 40Hz-es négyszögjeleket vár.

4.2. ábra. Ágens beavatkozója.



4.3. ábra. Négyszögjel és kitöltési tényező.

4.3. Mikrovezérlő (MCU[1])

A mikrovezérlő, egyetlen lapkára integrált, vezérlési feladatokra optimalizált számítógép. Felhasználásával költséghatékonyan képes a rendszer ellátni egyszerű, kis számítási teljesítményt és operatív tárat igénylő műveleteket. A felhasznált mikrovezérlő egy Atmel gyártmányú ATMEGA8-as, ami egy mikrovezérlőhöz készített nyáklapon található, ahol további a működéshez szükséges IC[2]-k és más áramköri elemek helyezkednek el. Feladata a beágyazott rendszertől kapott utasítások értelmezése és ezen utasítások alapján a szervomotor szabályozása.

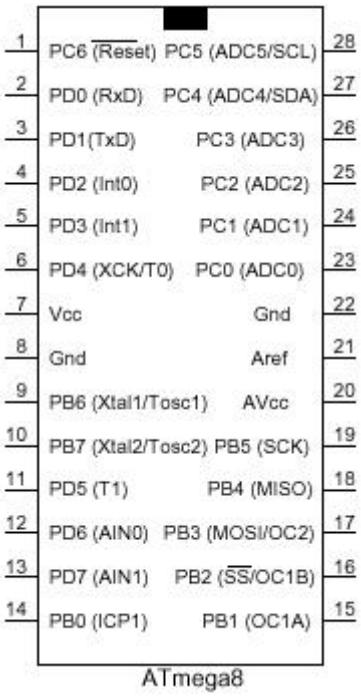
Az ATMEGA8 egy rendkívül kiforrt architektúra, amit számtalan helyen alkalmaznak. Gyártó által kibocsájtott mikrovezérlő leírás nagy segítséget nyújt mind a felhasználási területek megválasztásához, mind pedig ezek megvalósításhoz. Jelen esetben a mikrovezérlőnek két feladata van: jelgenerálás és soros portos kommunikáció. A jelgenerálás ebben az esetben a szervomotor szabályzásához szükséges négyszögjelek előállítását jelenti. Bár a mikrovezérlő rendelkezik hardveres jelgeneráló PWM[3]-el, a generált négyszögjelek egy másik lábon (PD2(INT0)) fognak megjelenni a mikrovezérlő szoftverből generálva. Soros porti kommunikációhoz pedig soros interfész elhelyezése szükséges a nyáklapon.

4.4. ábra. ATMEGA 8.

[1] MicroController Unit
[2] Integrated Circuit
[3] Pulse With Modulation

Mikrovezérlő jellemzői.

- 8KB FLASH memória
- 256B EEPROM[4]
- 512B SRAM[5]
- 1-4MHz belső oszcillátor
- AD[6] átalakító két
- 8 bites időzítő egy
- 16 bites időzítő
- PWM (Pulse With Modulation)
- SDI soros kommunikáció (Teljes duplex)
- párhuzamos kommunikáció



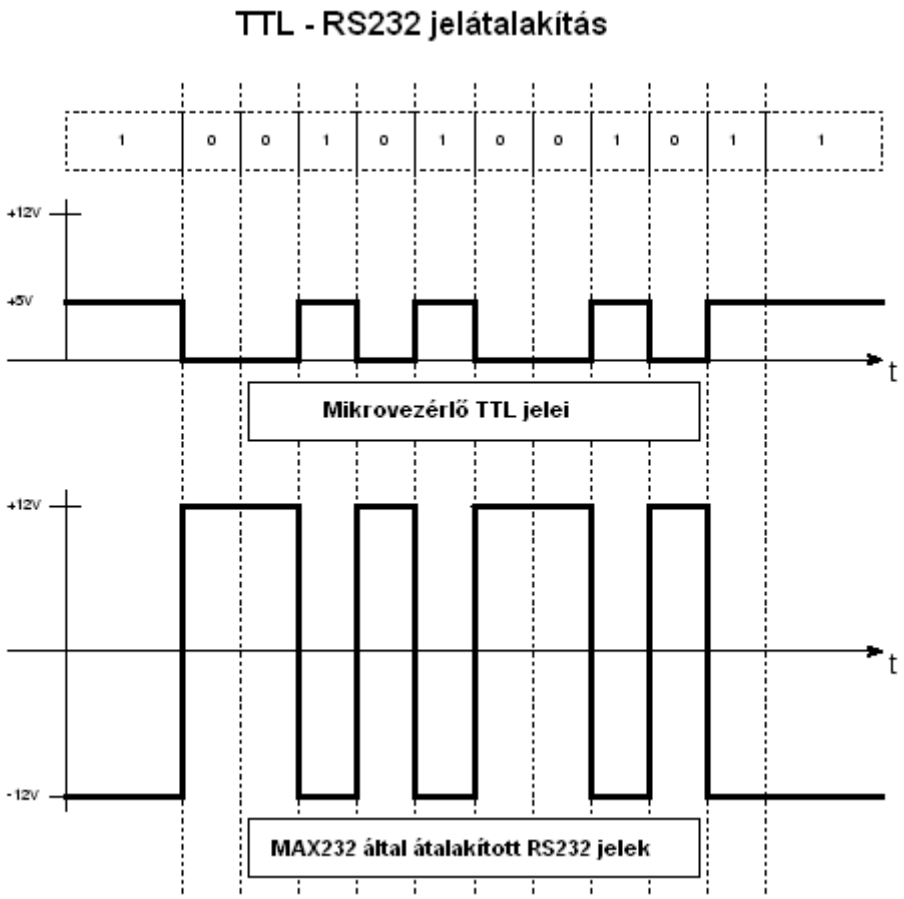
4.5. ábra. ATMEGA8 26 lábas DIP tokozással.

[4] Electrically Erasable Programmable Read-Only Memory
[5] Static Random Access Memory
[6] Analog to Digital

4.4. A kommunikáció a mikrovezérlő és a beágyazott rendszer között

Kommunikáció mikrovezérlő és beágyazott rendszer között soros porton (RS232[7]) keresztül történik. A mikrovezérlő csak TTL[8] jelekkel tud kommunikálni, ezért egy RS232 – TTL jelátalakítóra van szükség. Ezt a feladatot látja el a MAX232 IC, ami ugyancsak megtalálható a mikrovezérlős nyákon. Ezt az átalakító IC-t a mikrovezérlő gyártója ajánlja saját honlapján, arra az esetre, ha a rendszerünket soros portos kommunikációval szeretnénk ellátni.

4.6. ábra. MAX232.



4.7. ábra. TTL – RS232 jelátalakítás.

[7] Recommended Standard 232
[8] Transistor-Transistor Logic

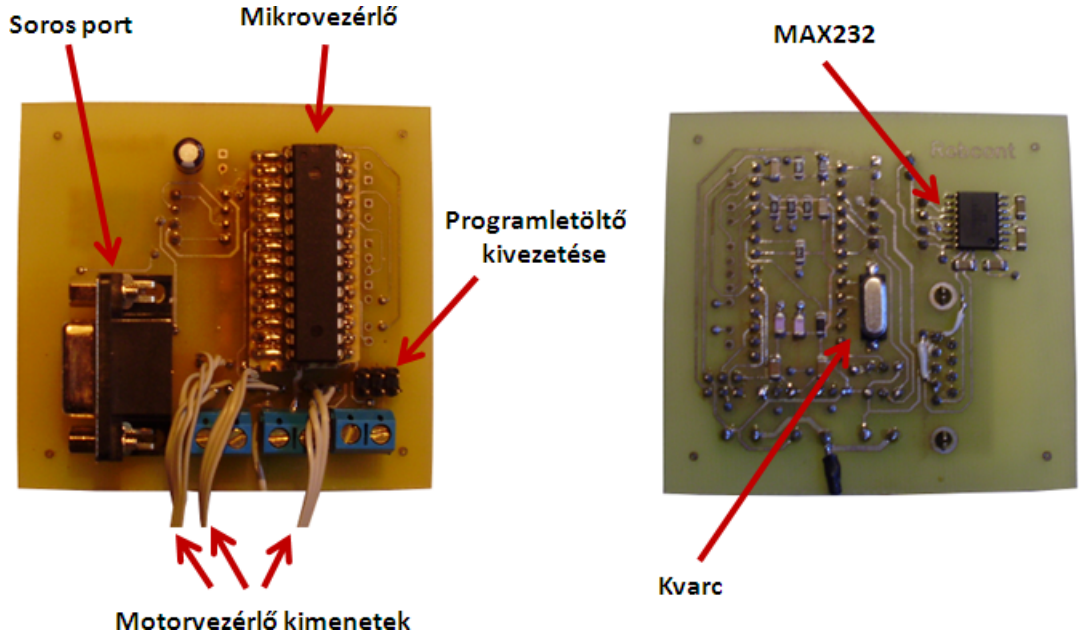
4.5. Mikrovezérlős nyák

Az elkészített mikrovezérlős nyáklapon minden megtalálható, ami a motorvezérléshez kell. Maga a mikrokontroller, a MAX232 átalakító, kvarc, motorok kivezetései, stb.

A mikrovezérlő beépített kvarca helyett egy külső, gyorsabb darab került beszerelésre. Minden ilyen eszköz két órajel alatt végez el egy utasítást. Az elsőben beolvassa a programmemóriából, hogy mit kell végrehajtania, a másodikban pedig egy írás következik, de eközben már megtörténik a következő olvasási művelet. A külső gyorsabb kvarc segítségével ezen utasítások megvalósításához szükséges idő jelentősen csökken.

Soros portos kapcsolathoz egy soros interface van ráforrasztva. Ehhez közvetlen módon lehet rácsatlakoztatni a beágyazott rendszer soros kimenetét. Lábairól az RS232 jelek a MAX232-be futnak be, és innen küldi tovább a már átalakított TLL jeleket a mikrovezérlő felé.

Végül megtalálható rajta a szervomotorhoz szükséges tápfeszültség és négyszögjelet adó kimenetek is.



4.8. ábra. A mikrovezérlős nyák felépítése.

4.6. Beágyazott rendszer (EPIA[9])

A mikrovezérlőhöz hasonlóan a beágyazott rendszer is egy egyetlen lapra integrált, célfeladatra kialakított számítógép. Viszont hardvere sokkal erősebb, operációs rendszerrel működtethető, gyakorlatilag egyenértékű egy hagyományos PC-vel. Az általános célú számítógépekkel szemben egy beágyazott rendszer csupán néhány előre meghatározott feladatot lát el, és sokszor tartalmazhat olyan feladat-specifikus mechanikus és elektronikus alkatrészeket, melyek nem találhatók meg egy általános célú számítógépben. Méretei lehetővé teszik kis helyen való elhelyezését, a rendszer hordozhatóságát, így gyakorlatilag rengeteg helyen alkalmazható, mint jeleket feldolgozó, adatokat továbbküldő rendszer.

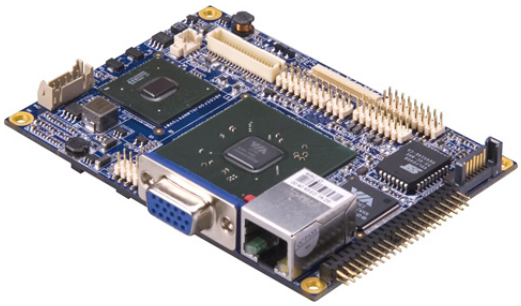
Az ágens beágyazott rendszere VIA gyártmányú EPIA PX. Interfészei között megtalálható minden, ami egy mai modern számítógépben. Ezeket a hardvereket nyílt forráskódú operációs rendszererrel

[9] Embedded Platform Innovative Architechture

célszerű működtetni és valamilyen hardver közeli vagy real-time magas szintű programozási nyelven a célfeladatot megoldani. Ezáltal a lehető legtöbbet lehet kihozni a hardverből, és az adott vezérlést megfelelő sebességgel megoldani.

Beágyazott rendszer jellemzői.

- VIA C7 1.0GHz NanoBGA2 CPU
- VIA VX700 chipset
- 512MB DDR2 memória
- 4GB CF FLASH háttértár
- Integrated VIA UniChrome monitorvezérlő
- VIA VT1708A HD audio
- 1db UltraDMA 133/100 IDE csatlakozó
- 1db SATA csatlakozó
- 1db VIA VT1708A HD audiokimenet
- 4db USB csatlakozó
- 1db Soros port
- 1db RJ45 csatlakozó
- 1db VGA csatlakozó



4.9. ábra. EPIA PX.

Robotágensem esetében az EPIA feladata feldolgozni a videóeszköz képét és ez alapján a szervomotor vezérlését megvalósítani közvetett módon a mikrovezélő segítségével. A kamera USB-n csatlakozik, motorvezérléshez szükséges üzeneteket pedig soros porton keresztül küldi. Az általam kitűzött feladat megoldásához csak erre e két csatlakozási pontra van szükség.

5. fejezet

Az ágens szoftvere

A szoftver az ágensfüggvényt implementálja, ezért ágensprogramnak nevezhetjük. E program feladata egy kitüntetett, a környezetétől különböző színű kör alakú tárgy követése. Ehhez fel kell ismerni a kamera előtt lévő kör alakú, környezetétől eltérő színű tárgyat, meg kell határozni középpontját, és ez alapján a szervomotort a megfelelő szögbe pozicionálni. Ezt a feladatsorozatot ciklikusan addig végzi, amíg manuálisan le nem állítjuk a program futását. A folyamat során nem csak a tárgy középpontja, hanem a kör sugara is meghatározásra kerül. Így az ágensprogram meg tudja határozni, hogy az előtte levő tárgy a kamerától távolodik (csökken a sugár), illetve ahhoz közeledik (nő a sugár).

Ezen művelet meghatározásához először egy olyan programra van szükség, ami kamerából képes képet felvenni. Ezután a felvett képen az RGB[1] színkódokból egy HSV[2] színábrázolássá való áttranszformálás során a megadott színárnyalat értékét kell elszeparálni. Szeparálás után a kapott képen a kört felismerő Hough-transzformáció meghatározza a képen található kör középpontját, és ezen információk alapján az ágensprogram meghatározza az elmozdulás irányát fokban. A számított értéket továbbküldi a mikrovezérlőnek, ami beállítja a szervomotort a megadott szögre. Végül ez a folyamat újra és újra megismétlődik a program leállításáig.

5.1. Operációs rendszer

Az ágensprogram alatt futó operációs rendszer egy Linux rendszer. Ennek előnye, hogy nyílt forráskódú és ingyenes. Tehát bárki módosíthat rajta és felhasználhatja saját belátása szerint. Testreszabhatósága miatt bármilyen igénynek megfelelő, robusztus, a hardvert jól kihasználó operációs rendszer állítható össze, a megfelelő csomagok telepítésével. Mivel a forráskód is rendelkezésre áll, így akár forráskódból fordítva létrehozható egy hardverspecifikus rendszer.

Napjaink Linux kernelje már elég fejlett ahhoz, hogy megbirkózzon a különböző video információt közvetítő eszközök működtetésével. Ezekhez az eszközökhöz hozzáférést tud biztosítani egy programozói felülettel, aminek a segítségével felhasználói programból a hardver kezelhetővé válik. Linux rendszereken a programokat hagyományosan (ugyanúgy, mint a Linux kernelt is) C magasszintű programozási nyelven szokás elkészíteni, ezért az ágensprogramot is C nyelven készítettem el. Az ágensfüggvény implementálása során a cél az, hogy az ágensprogram minél gyorsabban és megbízhatóbban végezze a feladatát. E követelményeknek tökéletesen megfelel a C eljárás orientált programozási nyelv. Használatával a hardvert jól kihasználó, real-time program implementálható.

[1] Red – Green – Blue
[2] Hue – Saturation – Value

5.2. Videó driver

Linux operációs rendszerben a kernel részeit képezik bizonyos videoeszközöket meghajtó driverek. Ilyenek például a webkamerákat kezelő gspca és uvc driverek (természetesen léteznek még más kameradriverek is). Ezek a driverek nem minden videoeszközt támogatnak, így olyan hardvert kell kiválasztani linuxos környezethez, amihez meghajtóprogram is található (Ilyen információkhoz könnyen hozzájuthatunk, ha felkeressük az egyes driverek készítőinek weblapját és azon tájékozódunk, hogy eddig milyen videoeszközöket próbáltak ki és melyek működtek az adott meghajtóprogrammal). Amennyiben a kernelben levő driverrel működőképes hardverünk van, csatlakoztatás után a `/dev` könyvtárban fog megjelenni a videoeszköz (pl.: `/dev/video1`). Az eszköz felismerése után megvizsgáljuk, hogy az adott meghajtóprogram támogatja-e a V4L[3] programozási felületet. A V4L API[4] felhasználásával az azt támogató videoeszközök könnyen programozhatóvá válnak és képesek leszünk képeket felvenni az adott hardverből.

5.3. Video for Linux 2 API (V4L2)

A Video4Linux (V4L) általános programozási felületet ad számos gspca és uvc driverekkel meghajtott új és régi TV-, videokártyához, továbbá párhuzamos porton és USB-n csatlakozó kamerához. Ezeken túl képes rádió, teletext elérésére is. Ennek az API-nak továbbfejlesztett, újragondolt változata a V4L2, ami már a Linux kernel 2.2 verziójával is képes volt együttműködni (2.5 és 2.6 verziók óta már alapértelmezetten a kernel része). Számos ismert program (mplayer, ffmpeg) használja ezt a programozási felületet videoeszközök képének megjelenítésére, képi információk mentésére.

5.4. A V4L2 lehetőségei

5.4.1. Eszközök megnyitása, zárása

Minden egyes a driverek által felismert videoeszköz-csatlakoztatás után a `/dev` könyvtárba fog létrejönni egy `video` nevű fájlként (a többi videó eszköz függvényében adott sorszámmal, például: `/dev/video0`). Mivel a Linux rendszerek egyik sajátossága, hogy szinte mindent fájlként kezelnek, ezért például C nyelvű programokból egyszerűen egy `open` függvénnyel ezek az eszközök megnyithatók írásra, olvasásra. Ez a V4L2 esetében sincs másképpen, a kiválasztott hardver megnyitása után az eszköz már használatra kész. Természetesen bizonyos paramétereket még be kell állítani az eszköz tényleges működtetése előtt.

Az adatcsere befejezése után `close` függvénnyel a hardver használata megszüntethető. A V4L2 egyik sajátossága a korábbi verzióhoz képest, hogy támogatja egy eszközt többszöri megnyitását. Ezáltal több különböző programból is használható ugyanaz a hardver azonos időben.

5.4.2. A hardverek képességei

A kiválasztott videoeszközök használata előtt mindig jó tudni, azok mire is képesek, mennyire támogatottak a hozzá tartozó driver és a V4L2 által. Ezért érdemes a megnyitás után lekérdezni ezek az információkat és ezek tudatában kiválasztani a legeffektívebb video be- és kimeneti folyam megvalósítását.

[3] Video for Linux
[4] Application Programming Interface

Az ilyen információk lekérdezésére és beállítására minden esetben az `ioctl` függvényt kell használni. Feladata eszközök paramétereinek manipulálása. Használata a következő: első paramétere egy `FILE` pointer, második egy eszköz specifikus beállítás, harmadik egy pointer, ami írás esetén tartalmazza a beállítani kívánt értékeket, olvasás esetén pedig azt, hogy hova kerüljenek az eszközspecifikus tulajdonságok. Ez a harmadik paraméter valamilyen videó tulajdonságokat leíró struktúrák egyike, amelyek előre definiáltak a V4L2-ben. A `videodev2.h` fejlécállomány beillesztésével ezeket mind megkapjuk és használhatjuk. Második paramétere szintén V4L2 specifikus. A fejezet további részében nagyrészt ezekről lesz szó, illetve a hozzájuk tartozó struktúrákról. Végül a metódus sikeres végrehajtás esetén 0-át ad vissza, -1-et hiba esetén.

Tulajdonságok megszerzéséhez először egy `v4l2_capability` nevű struktúrát kell definiálni. Ebben fog megjelenni minden a hardverrel kapcsolatos információ. Lényeges adattagjai következők:

- `driver`: hardver meghajtóprogramja
- `card`: eszköz neve
- `capabilities`: videó eszköz képességeit tárolja hexadecimálisan

A `capabilities` értékei konstansként előre definiáltak a `videodev2.h` fejlécállományban. Ebből kiderül, az adott hardver milyen lehetőségekkel bír. Például ha a kapott tulajdonság 0x01020011, az alábbi értékekből adódott össze:

- 0x00000001: a hardver egy felvevő eszköz
- 0x00000010: támogatja a tömörítésmentes felvételt
- 0x00020000: van audio támogatása
- 0x01000000: read/write függvénnyel kezelhető

Többi lehetőség megtalálható az előbb említett header fájlban.

Struktúrán kívül szükséges tudni az `ioctl` második paraméterét is, amik az struktúrákhoz szorosan kapcsolódnak. Minden ilyen V4L2 struktúrához tartozik egy vagy kettő (olvasási és írási lehetőségek együttléte esetén) makro, ami tartalmazza az „ioctl kódokat„ ami ebben az esetben `VIDIOC_QUERYCAP` megnevezésű. Ezek tudatában már le tudjunk kérdezni a videoeszköz képességeit:

```
int fd = open("/dev/video0", O_RDWR, 0);
if(fd < 0) exit(1);
struct v4l2_capability cap;
if(-1 == ioctl(fd, VIDIOC_QUERYCAP, &cap))
{
        perror("VIDIOC_QUERYCAP");
        close(fd);
        exit(1);
}
```

Sikeres végrehajtás esetén a `cap` változó fogja tartalmazni a hardverspecifikus információkat.

További tulajdonságok kiderítésére is lehetőség van a `VIDIOC_ENUMINPUT` ioctl és a hozzá tartozó `v4l2_input` struktúra felhasználásával. Segítségével felderíthető, hogy az adott eszköz analóg vagy digitális, kamera vagy TV tuner, be van-e kapcsolva, stb.

5.4.3. Videó be- és kimenetek

Videó be- és kimenetek fizikai interface-ek a rájuk kapcsolt eszközökhöz. Ezek lehetnek RF csatlakozók (antennák, koax kábel, stb.), Composite Video, S-Video vagy RGB csatlakozók. Bemeneti

képességgel (programból írható) csak a video- és VBI felvevő eszközök rendelkeznek, a többi csak kimeneti lehetőséggel.

Ezek a tulajdonságok az előző részben említett `VIDIOC_ENUMINPUT` és `VIDIOC_ENUMOUTPUT` ioctl-ok (attól függően, hogy be- vagy kimeneti egységről van szó) felhasználásával feltérképezhetők. Továbbá beállítható (`VIDIOC_S_INPUT` és `VIDIOC_S_OUTPUT`), illetve lekérdezhető (`VIDIOC_G_INPUT` és `VIDIOC_G_OUTPUT`) milyen eszköz van éppen használatban a megfelelő ioctl-okkal.

Következő kódrészlet bemutatja, hogyan kell váltani bemeneti eszközök között futásidőben:

```
struct v4l2_input input;
int index;
//Melyik bemeneti eszköz van használatban
if(-1 == ioctl(fd, VIDIOC_G_INPUT, &index))
{
        perror("VIDIOC_G_INPUT");
        exit(1);
}
memset(&input, 0, sizeof(input));
input.index = index;
//Tulajdonságok lekérdezése
if(-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input))
{
        perror("VIDIOC_ENUMINPUT");
        exit(1);
}

index = 1;
//Használt videoeszköz váltása
if(-1 == ioctl(fd, VIDIOC_S_INPUT, &index))
{
        perror("VIDIOC_S_INPUT");
        exit(1);
}
```

5.4.4. Képformátumok

A V4L2 API elsősorban arra lett kitalálva, hogy képi információ cseréjét valósítsa meg a hardver és felhasználói alkalmazás között. Ehhez a `v4l2_pix_format` struktúra nyújt segítséget, amivel meghatározható milyen formátumban tárolódjon a kép a memóriában. Struktúra tagjai:

- `width`: képszélesség
- `height`: képmagasság
- `pixelformat`: a kódolás formátuma, ezt az alkalmazásban kell beállítani
- `field`: a képi információk mindig összefüggnek, de az alkalmazások kérhetik, hogy a felvevő (vagy kimeneti) eszközök a kép alsó illetve felső területét külön buffer-ben tárolják
- `bytesperline`: távolság a szomszédos sorok legbaloldalibb pixeli között bájtban
- `sizeimage`: a kép tárolásához szükséges buffer mérete, ezt a hardver meghajtóprogramja állítja be
- `colorspace`: pixelformat egy kiegészítése, ezt is a driver határozza meg
- `priv`: további információk a kódolásról

Képek sikeres küldéséhez, fogadásához mind a videoeszköz, mind a felhasználó által futtatott program oldalán elengedhetetlen, hogy a kapott, illetve küldött információhalmazt ugyanúgy értelmezzék. Erre a problémára is megoldással szolgál a V4L2 számos előre definiált formátumával. Természetesen nem csak ezekre korlátozódnak a V4L2 driverek képességei, lehetőség van hardverspecifikus formátumok alkalmazására. Ebben az esetben viszont egy dekódolóval is rendelkeznünk kell, hogy szükség esetén

visszaalakítható legyen a képi információ szabványos formátumba. Ettől még a kép a hardver által támogatott formátumban fog tárolódni. Ilyen eset lehet amikor egy hardver támogat egy szabadalmaztatott tömörítési formátumot. Mind hardver, mind szoftver oldalon a kép ebben a tömörített formátumban készül, átadódik az alkalmazásnak és végül mentésre kerül a háttértárra. Viszont ahhoz, hogy a felhasználó meg tudja jeleníteni a mentett képet, szüksége lesz a dekódolóra.

V4L2 által nyújtott formátumok túlnyomó részben tömörítés mentesek. Ezeknél a pixelek mindig a bal felső saroktól kezdődően a jobb alsó sarokig tárolódnak. Tehát az első bájt mindig a felső sor legbaloldalibb pixelét tartalmazza, jobbra haladva tovább végül elérjük az utolsó pixelt is. Az előre definiált képi formátumok három csoportra bonthatók:

- RGB formátumok
- YUV formátumok (Y – világosság, U = B - Y, V = R - Y)
- Tömörített formátumok

RGB formátumok tipikusan a számítógépek által használt buffer méretekhez lettek tervezve. Ez azt jelenti, hogy pixelenként 8, 16, 24, 32 bitet foglalhatnak le. Három komponens – piros, zöld, kék – értékeinek összeadásával adódik a szín. Az egyes színelemek leggyakrabban 0-tól 255-ig terjedhetnek.

YUV formátum egy natív TV- és kompozit videojel. Ebben a kódolásban a fényerősséghez (Y) és a színhez (U és V) tartozó információk el vannak választva. A színadatot piros és kék színek tárolják, a zöld szín, pedig a fényerősség komponens használatával állítható elő. Ez a formátum azért került bele a V4L2 API-ba, mert a korai televíziózás korában is csak a fényerősség volt felhasználva a képek továbbítására. Másik fontos szempont, hogy az U és V részek átviteléhez kevesebb frekvencia szükséges, mint az Y komponenshez. Ezért széles területen használják ezt az analóg átviteli technikát, ami jelentősen függ a fényerősség mértékétől.

5.4.5. Be- és kimeneti adatfolyamok

Adatfolyamok megvalósításra számos lehetőség közül lehet választani a V4L2 API-ban. Ehhez a hardvert meghajtó illesztőprogramnak legalább az egyik technikát támogatnia kell.

Klasszikus módon használhatók ebben az esetben is a `read` és `write` függvények. Használat előtt viszont meg kell győződni, hogy támogatott-e ez a művelet a driver által. Ez az előzőekben tárgyalt módon, a hardver képességeinek lekérdezésekor deríthető ki. Ha a `v4l2_capability` stuktúra `capabilities` mezője `V4L2_CAP_READWRITE` értékkel tér vissza, a szabványos olvasás-írás metódusok támogatottak.

Hagyományos olvasás esetén a hardverből a memóriába másolódnak a képi információk (írás esetén természetesen fordítva). Ehhez a CPU közreműködése szükséges, amit minden esetben támogatott, ha a driver kezelni tudja a `read` és `write` függvényeket. Viszont a CPU közbeiktatásával a másolási, írási folyamatok lelassulnak, így a felhasználói program teljesítménye is romlik. Erre megoldást nyújt a DMA[5] vezérlők használata. Legtöbb meghajtóprogram `read`, `write` használata esetén ezt támogatja. A DMA, azaz közvetlen memória hozzáférés arra ad lehetőséget, hogy ne a processzor, hanem egy külön áramkör, a DMA-vezérlő irányítsa az I/O műveleteket, feltéve, ha nem igényel processzor műveletet. Ez azzal az előnnyel jár, hogy a DMA művelettel párhuzamosan a CPU folytathatja a programvégrehajtást, ezzel gyorsítva a feldolgozás folyamatát. A perifériák (ez esetben a videoeszköz) és a memória között alkalmazzuk, ugyanis, ha a processzor mozgatja az információt, akkor ez kiesett

[5] Direct Memory Access

idő, hiszen ez alatt a CPU nem áll rendelkezésére. A processzor tehermentesítésére és az adatátvitel egyszerűsítésére alkalmazzák a közvetlen memória-hozzáférést.

Egy másik megoldás a Memory Mapping. A legtöbb eszköz ezt is támogatja. Meggyőződhetünk róla ha lekérdezzük a `v4l2_capability` struktúra `capabilities` mezőjét. Használatához `V4L2_CAP_STREAMING` értéket kell tartalmaznia. Továbbá szükséges a `VIDIOC_REQBUFS` ioctl meghívása is, ez megadja, hogy lefoglalható-e buffer terület a hardveren belül.

A módszer erőssége, hogy az adatfolyam működés közben csak mutatókat küld a felhasználói program felé, így nem kerül semmilyen adat másolásra, a hardver saját memóriájába foglal le tárterületet a képnek. Meghajtóprogramok nagy része több ilyen tárterület lefoglalását is támogatja. Az egyes buffer-ek egymástól függetlenek, és azokba különböző adatok kerülhetnek. Tárterületek mennyiségének felderítéséhez `v4l2_requestbuffers` struktúra és `VIDIOC_REQBUFS` ioctl szükséges. Struktúra `count` mezője tartalmazza a kérdéses információt. Ez az ioctl használható továbbá a buffer méret beállítására, tárterület foglalására, illetve felszabadítására.

Mielőtt a felhasználói program hozzáférne a lefoglalt memóriához, le kell futtatni az `mmap` függvényt. Ehhez előbb meg kell határozni, az `VIDIOC_QUERYBUF` ioctl-al, hogy hol található a lefoglalt buffer a hardverben. Ennek egy `v4l2_buffer` struktrúrát kell átadni, aminek a `length` tagja az `mmap` második, `m.offset` tagja pedig a hatodik paramétere lesz. Ezek értéke nem változtatható meg. Miután az adatfolyam lezárult, a felhasznált memória a `munmap` metódussal szabadítható fel. Következő kódrészlet az Memory Mapping megvalósítását mutatja be:

```
struct v4l2_requestbuffers reqbuf;
struct
{
      void *start;
      size_t length;
} *buffers;
unsigned int i;
memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;
if(-1 == ioctl(fd, VIDIOC_REQBUFS, &reqbuf))
{
      printf("Nem támogatja a mmap-ot!\n");
      exit(1);
}

//Legalább 5 memóriát szeretnénk
if(reqbuf.count < 5)
{
      printf("Nincs elég buffer memória!\n");
      exit(1);
}
buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);
//Tárterületek beállítása
for(i = 0; i < reqbuf.count; i++)
{
      struct v4l2_buffer buffer;
      memset(&buffer, 0, sizeof(buffer));
      buffer.type = reqbuf.type;
      buffer.memory = V4L2_MEMORY_MMAP;
      buffer.index = i;
      if(-1 == ioctl(fd, VIDIOC_QUERYBUF, &buffer))
      {
            perror("VIDIOC_QUERYBUF");
            exit(1);
      }
      buffers[i].length = buffer.length;
      buffers[i].start = mmap(NULL, buffer.length,
                              PROT_READ | PROT_WRITE,
                              MAP_SHARED,
                              fd, buffer.m.offset);
      if(MAP_FAILED == buffers[i].start)
      {
            perror("mmap");
            exit(1);
      }
}
//Végül a lefoglalt memóra felszabadítása
for(i = 0; i < reqbuf.count; i++)
      munmap(buffers[i].start, buffers[i].length);
```

Legtöbb esetben a driverek két sort tartanak fent a kimenő és bejövő adatfolyamok számára. Ezek FIFO-ként vannak kialakítva, tehát az elemek tárolási sorrendben olvashatók ki belőle. Tárhely lefoglalásakor alapértelmezetten kiürített állapotban vannak, a felhasználó számára még elérhetetlenek. Inicializálásukhoz először fel kell tölteni a sor adatszerkezetet képi információkkal annak függvényében mennyi tárhelyet foglaltunk le. Ezután kezdődhet el az írási vagy olvasási művelet. Miután ez megtörtént jelezni kell a programnak, hogy a buffer-ek rendelkezésre állnak az `VIDIOC_STREAMON` ioctl-al. Végül íráskor, olvasáskor folyamatosan ki kell üríteni, illetve fel kell tölteni a sort. Adatcsere lezárásakor pedig `VIDIOC_STREAMOFF` ioctl-t (jelezve, hogy az adatcsere befejeződött) és a `munmap` függvényt kell használni.

5.4.6. Az ágensprogram kamerát kezelő modulja

A kameraképet olvasó modul a Memory Mapping eljárást alkalmazza az egyes frame-ek megszerzése érdekében. Ezt a `webcam_grab.c` és `webcam_grab.h` forrásfájlok tartalmazzák. Az egyes videóhoz kapcsolódó információk tárolására egy struktúra lett kialakítva (`video_device`), ez tartalmaz mindent, ami az inicializáláshoz és a képfelvételhez szükséges.

```
struct video_device
{
      int fd;
      struct v4l2_capability cap;
      struct v4l2_format fmt;
      struct v4l2_requestbuffers reqbuf;
      int buffers_num;
      enum v4l2_buf_type type;
      struct v4l2_buffer outbuf;

      struct
      {
                  void *start;
                  size_t length;
      } *buffers;
};
```

Inicializáló függvénye (`v4l2_device_init`) lekérdezi a hardver képességeit és beállítja a felvétel formátumát. Egyetlen buffer tárat foglal le, ennyi elegendő az ágens feladatának megoldásához. Végezetül jelzi, hogy felvételre kész a program.

A `v4l2_read_frame` metódus először feltölti a sort az aktuális információval, majd kiolvassa ezt. A kapott képet bájt tömbként adja vissza a függvényt meghívónak. Adatáramlás lezárására a `v4l2_device_close` szolgál, valamint felszabadítja a lefoglalt memóriát. Továbbá a forrásfájl tartalmaz egy tömörítetlen képet háttértárra mentő függvényt (`webcam_write_raw_picture`) a program jobb futás közbeni ellenőrzéséhez.

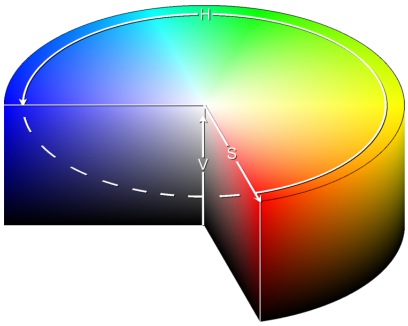
5.5. RGB és HSV színkódolások

Az RGB színrendszerben a színek a három alapszín a vörös (R - red), a zöld (G - green), és a kék (B - blue) egymásra vetítésével - összeadásával - állíthatók elő, ez tulajdonképpen additív színkeverés. Ezek az értékek 0-tól 255-ig terjedhetnek, tehát egy színkomponens legtöbb esetben egy bájton tárolódik. Legegyszerűbb, ha tömörítés mentesen, 24 biten ábrázoljuk az egyes pixelek színeit, így minden komponensre 8 bit jut.

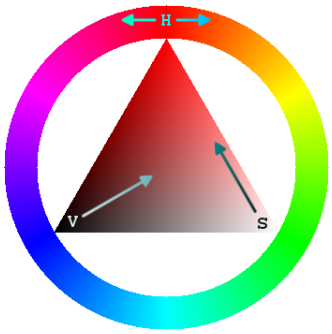
A HSV egy másik színábrázolás, ami sokkal jobban megjeleníti, érzékelteti az egyes színek közötti kapcsolatokat. Három részből áll: színárnyalat (H – hue), telítettség (S – saturation), érték (V – value). E színábrázolást legegyszerűbben egy háromdimenziós, henger alakú alakzattal lehet bemutatni. A színárnyalat körgyűrűje hat színen alapul (Piros – Sárga – Zöld – Világoskék – Kék – Magenta). Ezek között továbbá megtalálható az egyes alapszínek közötti átmenet. A középpont felé haladva – ahol

a fehér szín található – az y tengelyen megadható a telítettség érték. Harmadik paraméter, az érték pedig a henger z irányába adható meg, ahol a fekete a henger alján található.

Természetesen ennek a színábrázolásnak létezik kétdimenziós megjelenítése, amit előszeretettel használnak fényképészek, designerek photoshop-ban, gimp-ben és a többi hasonló képmanipuláló programban.



5.1. ábra. HSV háromdimenziós megjelenítése.



5.2. ábra. HSV kétdimenziós megjelenítése.

5.5.1. RGB-ből áttérés HSV színkódolásba

Kamera által rögzített kép RGB színkódolásban készül és tárolódik a buffer-ben. Ahhoz, hogy az ágens folytatni tudja feladatát át kell térni HSV színkódolásba. Ez egyszerűen megoldható, a képen található összes pixelre ki kell számolni a színárnyalat – telítettség – érték hármast. Következő algoritmus mutatja be ezt a folyamatot:

Az algoritmus lefutása után a három érték ezekbe az intervallumokba fog esni:

$$0 \leqq Színárnyalat \leqq 360$$

$$0 \leqq Telítettség \leqq 100$$

$$0 \leqq Érték \leqq 255$$

1. lépés, H – S – V értékek kiszámítása:

   (a) Színárnyalat: ha szín1 a maximális akkor $H = \frac{szín2 - szín3}{maximális\,értékű\,szín - minimális\,értékű\,szín}$

   (b) Telítettség: $S = \frac{maximális\,értékű\,szín - minimális\,értékű\,szín}{maximális\,értékű\,szín}$

   (c) Érték: $V = maximális\,értékű\,szín$

2. lépés, kapott értékek normalizálása a fent említett tartományokba:

   (a) piros a domináns szín (az a Maximális értékű szín): H *= 60

   (b) zöld a domináns szín: H += 2 ; H *=60

   (c) kék a domináns szín: H += 4 ; H *= 60

   (d) telítettség beállítása: S *= 100

Ugyanez az algoritmus C nyelven megfogalmazva:

```
//Kék a domináns szín
if((B > G) && (B > R))
{
      V = B;
      if(V != 0)
      {
            double min;
            if(R > G) min = G;
            else      min = R;
            double delta = V - min;
            if(delta != 0)
                  { S = (delta/V); H = 4 + (R - G) / delta; }
            else
                  { S = 0;         H = 4 + (R - G); }
            H *=   60; if(H < 0) H += 360;
            S *= 100;
      }
      else
      { S = 0; H = 0;}
}

//Zöld a domináns szín
else if(G > R)
{
      V = G;
      if(V != 0)
      {
            double min;
            if(R > B) min = B;
            else      min = R;
            double delta = V - min;
            if(delta != 0)
            { S = (delta/V); H = 2 + (B - R) / delta; }
            else
            { S = 0;         H = 2 + (B - R); }
            H *=   60; if(H < 0) H += 360;
            S *= 100;
      }
      else
      { S = 0; H = 0;}
}

//Piros a domináns szín
else
{
      V = R;
      if(V != 0)
      {
            double min;
            if(G > B) min = B;
            else      min = G;
            double delta = V - min;
            if(delta != 0)
            { S = (delta/V); H = (G - B) / delta; }
            else
            { S = 0;         H = (G - B); }
            H *=   60; if(H < 0) H += 360;
            S *= 100;
      }
      else
      { S = 0; H = 0;}
}
```

5.5.2. Ágensprogram RGB átalakító és színeket szeparáló modulja

A programnak a színek elszeparálásához a színárnyalat – telítettség – érték hármas közül csak a színárnyalatra van szüksége. Az ágensprogram feltételezi, hogy a kamera előtt található egy meghatározott, környezetétől eltérő színű, gömb alakú tárgy. E kitüntetett tárgy színét kell elszeparálni a többi

képen levő színtől. Ehhez elegendő a színárnyalatot reprezentáló körgyűrű egy intervallumát kijelölni (ami az adott színt határozza meg). A beleeső értékek lesznek a jó értékek, a többivel pedig nem foglalkozik a program.

Szeparálást megvalósító kódot a `hue_sep.c` és `hue_sep.h` forráskódok tartalmazzák. Először a videó eszközből RGB színábrázolással kódolt képet kell átalakítani. Minden egyes pixelre a fent említett eljárást ki kell számolni. A futás meggyorsítása érdekében egy makro (`PIX_RGB_TO_HSV_HUE_ONLY`) végzi ezt a folyamatot.

A folyamat tényleges lebonyolítását a `hue_sep` függvény végzi. Argumentumainak a következőket kell megadni:

- képet tartalmazó tárterület mutatója
- kép szélessége
- kép magassága
- színárnyalat intervallum alsó határa
- színárnyalat intervallum felső határa
- szeparálás utáni pontok x koordinátáit tartalmazó tömb mutatója
- szeparálás utáni pontok y koordinátáit tartalmazó tömb mutatója
- szeparált pontok száma

Függvény a lefutás után feltölti az x és y koordinátákat tartalmazó tömböket és meghatározza a szeparált pontok számát. A szeparált pontok száma azért is fontos, mert egy bizonyos érték alatt az ágensprogram nem fut tovább (nem keresi a középpontot). Ennek okai lehetnek, hogy nincs a kitüntetett tárgy a kamera előtt, vagy az előre megadott távolságon túl van. Valamint képes a program e modulja megjeleníteni a kapott eredményt, úgy hogy azt kiírja egy képfájlba. Ezáltal nyomon követhető a program futása. A képen az intervallumba eső pontokat fekete színnel, a kívül esőket pedig fehérrel jelöli.

5.6. A kört felismerő transzformáció

A feladatmegoldáshoz elengedhetetlen megismerkedni egy kört felismerő transzformációval, ami nélkül a felvetett probléma nem oldható meg. Célunk az, hogy meghatározzuk a kör alakú tárgy középpontjának koordinátáit. Ebben a részben egy ilyen eljárás kerül tárgyalásra.

A Hough-transzformáció a digitális képfeldolgozásban általánosan használt módszer, amely egy adott kép pontjainak segítségével meghatározza az objektum hollétét. Egy adott képen a módszer alkalmazható vonalak, körök és más egyéb alakzatok felismerésére. A transzformáció feltételezi, hogy a felismerni kívánt objektum megtalálható a képen.

Kört felismerő Hough-transzformáció (CHT[6]) esetében a kör középpontját vagyunk képesek meghatározni. Ennek felderítéséhez előbb matematikailag kell leírni a felismerni kívánt objektumot. Kör esetében legegyszerűbb a paraméteres köregyenlet használata:

[6] Cirle Hough Transform

$$a = x + r * cos\Theta$$

$$b = y + r * sin\Theta$$

Transzformáció alapötlete, hogy xy sík vizsgálata helyett ab síkban számolunk az erre átalakított egyenletekkel:

$$x = a - r * cos\Theta$$

$$y = b - r * sin\Theta$$

Így minden xy pontpárhoz kapunk egy ab síkbeli kört, amelyek alapján már behatárolható a kör középpontja. Az eredmények tárolására egy képpel azonos méretű akkumulátor változóra lesz szükség. Az egyenlet kiértékelése közben kapott értékek a kép valamelyik pixelébe fognak esni. Ahányszor ez megtörténik az adott pixelre vonatkozó értéket inkrementálni kell. Miután az algoritmus lefutott, a maximális érték fogja jelölni a keresendő kör középpontját.

5.6.1. A kört felismerő ágensmodul

A szeparálás utáni képen a kör alakú objektumot felismerő Circle Hough-transzformációval folytatódik a programfutás. Feladata az előbb tárgyaltak alapján a kör középpontjának lokalizálása a kapott digitális képen.

A kör közepét meghatározó modult a `hough.c` és `hough.h` forrásfájlok tartalmazzák. A `hough` függvény feladata a kör közepének koordinátáit a képen megtalálni. Viszont az előbb tárgyalt paraméteres köregyenletek közül csak az x koordinátára van szükség, mert a kamerát mozgató szervomotor egy szabadsági fokú. Így az y koordináta meghatározása felesleges számítás, e részhez szükséges futási idő fele spórolható meg, ha csak az x koordinátát találja meg az algoritmus.

A program továbbá képes nemcsak a képen látható körközép x koordinátájának megtalálására, hanem annak sugarát is be tudja határolni. Ehhez a függvény hívásakor meg kell adni egy minimális és maximális sugárértéket, a program pedig kiválasztja a legnagyobb értékhez tartozó sugarat is. Ezen információk alapján akár az is kiszámolható, milyen messze található a kitüntetett tárgy a kamerához képest.

A `hough` metódus következő paramétereket várja:

- szeparálás utáni pontok x koordinátáit tartalmazó tömb mutatója
- szeparálás utáni pontok y koordinátáit tartalmazó tömb mutatója
- szeparált pontok száma
- kép szélessége
- kép magassága
- minimális sugár
- maximális sugár
- megtalált kör középpont x koordinátája
- megtalált kör középpont y koordinátája
- megtalált sugár

5.7. Soros kommunikáció az ágensprogram és a mikrovezérlő között

Kommunikáció soros porton történik a jelgenerátor (mikrovezérlő) és az ágensprogram között. Az adatátvitel megvalósításához egy előre meghatározott protokollt kell használni adó és vevő között, amit mind a két oldali egység tud értelmezni és a feladatát ennek alapján végrehajtani.

5.7.1. STX ETX protokoll

A két egység közötti kommunikáció alapvetően az adatkapcsolati rétegben valósul meg. Ennek a rétegnek az egyik legfontosabb feladata a keretek összeállítása. A keret az adat valamilyen határoló jelek közé foglalt része, amely rendszerint a tényleges adaton kívül egyéb információkat is tartalmaz. Mivel szükséges a biztonságos és hibamentes átvitel biztosítása, ezért nagyon fontos, hogy olyan ellenőrzést is kell a keretbe foglalni, amely alapján felfedezhetők a hibák. Ilyen esetben a rossz keretet újra el lehet kérni a küldőtől. Hibavédelem valamilyen ellenőrző kód, amely matematikai eljárással kerül meghatározásra. A keret vételekor a vevő ismét kiszámolja ezt az összeget, és ha a tárolttal nem egyezik, akkor a keret megsérült, a vevő eldobja és újrakéri a küldőtől.

Átvitel során bináris információt, vagyis biteket kell továbbítani az adó és a vevő között. A hatékonyabb működés érdekében egyszerre a lehető legtöbb bitet kell továbbítani. A számítógépek kialakulásakor nagyon sokszor kellett szöveges információt, vagyis karaktersorozatokat küldeni. A karakterek mindig azonos módon, az ASCII kódnak megfelelően tárolódtak. Az ilyen információtovábbítást karakterorientált átvitelnek nevezzük.

Kommunikációs protokollnak egy ilyen karakterorientált protokollt választottam, a DLE STX DLE ETX protokollt. A DLE rövidítés a Data Link Exchange (adatkapcsolat átkapcsolás), az STX a Start of Text (szöveg kezdete) az ETX pedig az End of Text (szöveg vége) kifejezésekből kialakított betűszavak. Leggyakrabban ezt szokták alkalmazni soros átvitel esetén. A keretek kezdetét és végét speciális karakterekkel jelöljük, ez a STX és az ETX. Ezek speciális, előre definiált konstansok, és keret adatrészében nem fordulnak elő. Ezen csomagok kis méretüknél fogva gyors, ellenőrző összege miatt biztonságos adatátvitelt tesznek lehetővé a két egység között. A keret pontos felépítése a következő:

STX | LEN | ADDR | COM | DATA | ADD | XOR | ETX

Minden egyes rész – kivéve az adatrészt, mert az több bájtból is állhat – 1 bájtos adatmennyiséget jelent. Tehát ha az összes rész 1 bájtos, összesen 8 bájtot küldünk a mikrovezérlő felé. Az egyes részek jelentése:

- STX – ezzel kezdődik a keret, konstans értékű: 0x02
- LEN – ADDR, COM, DATA részek hosszát tartalmazza
- ADDR – mikrovezérlő címe
- COM – utasítás kódja
- DATA – információt tároló adatrész
- ADD – ellenőrző művelet, ami összegzi a cím, utasítás és az adatrészben található bájtokat
- XOR – cím, utasítás és az adatrész bájtjain végzett kizáró vagy művelet, a hibás keretek elkerülésére kell

- ETX – végül valahogy jelezni kell az adatfolyam végét, erre szolgál az ETX, értéke: 0x03

A kamera forgatásáért felelős eljárás adat része a következőket tartalmazza: szervomotor és a szög ahova pozicionálni kell magát. Például ha 90 fokba szeretnénk beállítani a kamerát, a csomagot a 5.3-as ábrán jelöltek alapján kell kialakítani.

STX	LEN	ADDR	COM	DATA		ADD	XOR	ETX
0x02	0x04	0x00	0x01	0x44	0x5A	0x9E	0x1E	0x03
Keret kezdő karaktere	ADDR, COM, DATA hossza	Cím	Utasítás	Szervomotor kiválasztása	90 fokos pozíció	ADDR, COM, DATA összegző ellenőrzése	ADDR, COM, DATA kizáró vagy ellenőrzése	Keret utolsó karaktere

5.3. ábra. Példa STX ETX protokollra.

5.7.2. Soros kommunikációt megvalósító modul

Ugyanúgy mint a videoeszköz kezelésekor, a Linux rendszer a soros portot is fájlként kezeli. A soros kommunikációt megvalósító kódokat a `serial_com.c` és `serial_com.h` fájlok tartalmazzák. Először a soros port megnyitására és annak tulajdonságainak beállítására van szükség. Ezt a `open_serial_port` függvényt végzi, ami a következőket tartalmazza:

```
int fd;
fd = open(SERIAL_PORT, O_RDWR | O_NOCTTY | O_NDELAY);
if(fd == -1)
{
      fprintf(stderr, "/nopen_serial_port():/nCould not open serial device:
                       %s/n", SERIAL_PORT);
      return -1;
}
fcntl(fd, F_SETFL, 0);
//Átviteli sebesség beállítása
struct termios options;
tcgetattr(fd, &options); //Pillanatnyi tulajdonságai a portnak
//Atviteli sebesség: 115200
cfsetispeed(&options, B115200);
cfsetospeed(&options, B115200);
options.c_cflag |= (CLOCAL | CREAD);
tcsetattr(fd, TCSANOW, &options); //Beállítások mentése
```

Inicializálás után a `servo_move` függvénnyel küldhető információ a mikrovezérlő felé. Argumentumainak a következőket várja:

- mikrovezérlő címe (konstans, 0-s azonosítójú mikrovezérlő)
- parancs (konstans, értéke 1, azt jelöli hogy a szervomotort szeretnénk megmozdítani)
- melyik szervomotor (esetünkben csak egy van)
- pozíció megadása szögben

Először a metódus ellenőrzi a kapott adatokat, ha rossz paraméterek lettek megadva, az írás nem fog megtörténni. Ellenkező esetben felépíti a csomagot és az `add` és `xor` segédfüggvényekkel kiszámolja az ellenőrző összegeket. Végül kiírja a soros portra az információt.

5.8. Kamera beállítása a kiszámított pozícióba

Miután minden egyes modul tárgyalásra került már, végezetül a Hough-transzformációval kiszámított kör középpont és a kamera középpontja alapján meg kell határozni az elmozdulás szögét. A kamera az induláskor 90 fokos szögben található, ehhez adódnak, illetve vonódnak ki az egyes elmozdulás szög értékek annak függvényében, hogy jobbra vagy balra kell elmozdítani a kamerát.

A kiszámolt körközéppont és kamera középpontja közötti pixelmennyiséghez tartozó szöget tapasztalati úton számoltam ki. A kamera 30 cm távolságba megközelítőleg 20 cm-t lát be. Ebből kiszámítható, hogy a kamera látószöge hozzávetőlegesen 37^0 (pontosabban $36{,}869897646^0$). Amennyiben e középponttól jobbra található a kör középpontja, a kiszámított szöget ki kell vonni, ha balra van akkor hozzá kell adni az aktuális szögálláshoz. Kameraközépponttól való eltérést szög elmozdulásba a következő képlettel számolható ki:

$$EF = {}^{FOK}/_{KSZ} * EK$$

ahol

EF: hány fok az eltérés

FOK: $36{,}86987646^0$

KSZ: kép szélessége pixelben

EK: eltérés a kameraközépponttól pixelben

Ha a kapott eltérés kisebb, mint 1^0 az ágensprogram nem küld üzenetet a mikrovezérlőnek, így nem is történik meg a szervomotor elmozdítása. 1^0 alatti érték elfogadható hibahatárba tartozik. Ez pixelben kifejezve annyit jelent, hogy minimum 9 pixel eltérés kell, hogy a szervomotor új pozícióba kerüljön beállításra.

Ez a vezérlés megegyezik az elméleti részből ismert pontvezérléssel, mert csak a kezdő és végpont ismert, azt hogyan valósul meg a két pont között a mozgás nincs részletezve.

5.9. Mikrovezérlő programja

A mikrovezérlő programja C nyelven készült, CodeVision fejlesztőkörnyezet segítségével. Ez a fejlesztőkörnyezet nagy segítséget nyújt egy ilyen egy chip-es számítógép programozásában. Az elkészült programot a CodeVision, a kiválasztott vezérlő gépi kódjává alakítja át, amit egy AVRISP mk2 programletöltővel lehet rátölteni közvetlenül a számítógépről a mikrovezérlőre. Ez az eszköz ISP[7] protokollt használ a mikrovezérlővel való kommunikáció során. Ehhez a nyákon egy ilyen csatlakozási pont lett kialakítva. A programletöltő felhasználásával a mikrovezérlő programja könnyen végrehajthatóvá és tesztelhetővé válik.

[7] In-System Programmer

A mikrovezérlőt működtető C nyelvű program feladata a soros port olvasása, és az olvasott szögállás információ alapján a szervomotor pozicionálása. Működés során folyamatosan küldi a négyszögjelet és figyeli, hogy a soros porton megjelenik-e új adat, ami alapján változtatni kell a kitöltési tényezőn. A motor vezérlő négyszögjelek a mikrovezérlő PD2 lábán jelennek meg. Ehhez először a PD2-t kimenetként kell inicializálni. Az így beállított portot pedig a `PORTD.2` változón keresztül érhetjük el. Mivel a felhasznált szervomotor 40Hz-es jeleket vár, ezért egy periódus 25 milliszekundumig tart. Négyszögjelek előállításához a `PORTD.2` változót először 1-es értékre kell állítani meghatározott időtartamig attól függően, hogy mekkora kitöltési tényezőt (melyik pozíciót) szeretnénk küldeni a szervomotor felé. Végül az alapperiódusból fennmaradó részt 0-val töltjük ki. A három alapvető pozícióhoz ($0^0 - 90^0 - 180^0$) tartozó kitöltési tényező a következő:

- 0^0: 550us
- 90^0: 1350us
- 180^0: 2150us

Soros port olvasásához a `gets` függvény szolgál. A beolvasott keret adatrésze tartalmazza a pozíciót fokban, amiből a program kiszámolja a hozzá tartozó kitöltési tényezőt és ezeket a jeleket továbbítja a szervomotornak.

Következő kódrészlet a mikrovezérlő főfüggvényének soros portos információt olvasó és négyszögjeleket generáló részletét mutatja. A megvalósításban a PD2 láb elérését a `SERVO` (`PORTD.2` helyett) változó valósítja meg.

```
gets(keret, 9); //Soros port olvasása
...

kitoltes=550+((160*(int)keret[5])/18); //Kitöltési tényező számolása
if(kitoltes%2==1) kitoltes++;

for(i = 0; i<10; i++)
{
    SERVO = 1;
    my_delay_us(kitoltes);
    SERVO = 0;
    my_delay_us(20000-kitoltes);
}
```

5.10. Az ágensprogram használata

Az általam készített ágensprogram egy konzolos alkalmazás, működéséhez a felhasználónak opciókat kell megadni és a hozzájuk tartozó értékeket beállítani. Ezek az opciók a következők:

Kötelezően megadott argumentumok.

- -d/–device: videoeszköz azonosítójának meghatározása. Például: -d /dev/video0.
- -p/–port: soros port azonosítója. Például: -p /dev/ttyS0.
- -c/–color: színárnyalat intervalluma a HSV színtartományban, ez 0-tól 360-ig terjedhet. Például: -c 90-130.
- -r/–radius: kör sugarának intervalluma. Például: -r 60-61.

Választható opciók.

- -v/–verbose: beszédes mód bekapcsolása. A program futásának részeredményeit képfájlként jeleníti meg (kamerakép, szín szeparálás, körközéppont meghatározása). Minden kamerapozicionálás után billentyűlenyomással folytatódik a program futása.

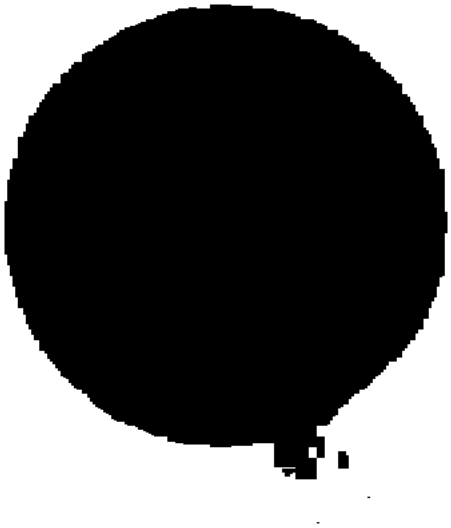
Segítség a paraméterek megadásához.

- -h/–help

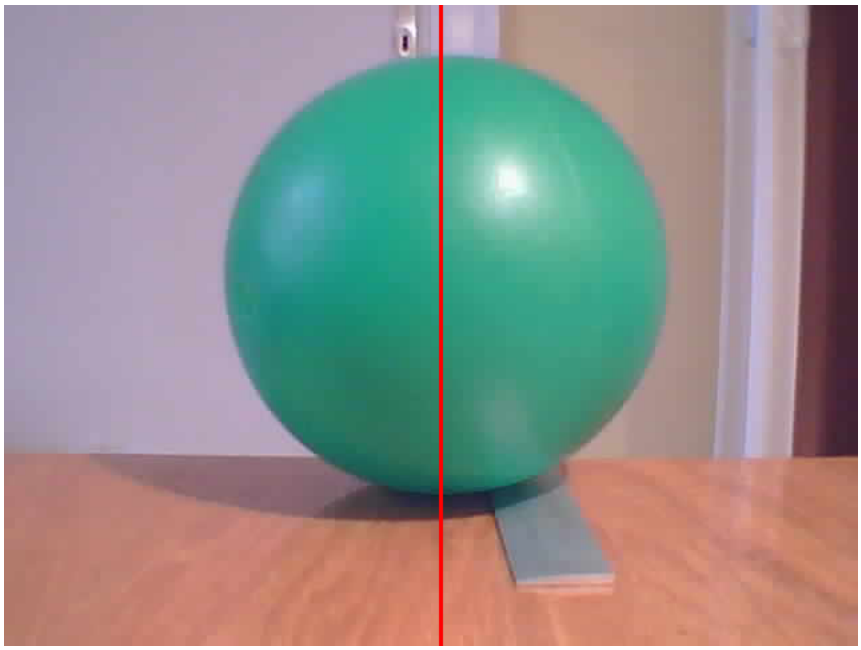
Helyes paraméterezés után kilistázza a videoeszközhöz tartozó fontosabb információkat, illetve a soros port beállításait. Billentyűlenyomás után megkezdődik az ágensprogram tényleges működése.

5.11. Ágensprogram működés közben

5.4. ábra. Webkamera által rögzített pillantkép.



5.5. ábra. Objektum színének elszeparálása.



5.6. ábra. Középpont x koordinátájának megjelölése.

6. fejezet

Összegzés, továbblépési lehetőségek

Diplomamunkám célja egy olyan hardveres intelligens ágens tervezése és létrehozása volt, ami egy kitüntetett köralakzatot képes felismerni érzékelőjével, és azt követni a beavatkozójával. Érzékelőnek egy hagyományos webkamerát használ fel, ami előtt helyezkedik el a környezetétől eltérő színű, kör alakú kitüntetett tárgy. Az ágensprogram meghatározza ennek a tárgynak a középpontját. Majd a kiszámított koordinátákat továbbítja a beavatkozó számára, ami egy szervomotor. Ezen a motoron helyezkedik el a képalkotó érzékelő. Az ágens feladata a kameraközéppont beállítása a köralakzat középpontjába.

Fizikai rész két legfontosabb eleme a számításokat végző egység, a beágyazott rendszer, valamint a szervomotort vezérlő jelgenerátor, a mikrovezérlő. A beágyazott rendszer tartalmazza az ágensprogramot, ami feldolgozza a külvilágból érkező jeleket, kiszámolja a körközéppontot és végül közvetett módon vezérli a motort. Mikrovezérlő feladata a szervomotor pozíciójának beállítása, amihez megfelelő kitöltési tényezőjű négyszögjeleket generál. A két egység közötti kommunikáció soros porton, STX-ETX protokoll felhasználásával valósul meg.

A beágyazott rendszeren futó Linux operációs rendszer segítségével mind a soros kommunikáció, mind a kamera kezelése kernelszinten támogatott. Használatukhoz szükséges technológia C programozási nyelven a legkiforrottabb, ezért az beágyazott rendszeren futó ágensprogram C nyelven készült. A szoftver a kamerából felvett pillanatképeket V4L2 API felhasználásával éri el, majd a rögzített, RGB színábrázolású képet HSV színábrázolásba alakítja át. Az így kapott képen színárnyalat alapján elszeparálja a köralakzathoz tartozó pontokat, és átadja a kört felismerő Circle Hough-transzformációnak, ami meghatározza a középpont koordinátáit. A koordináták alapján már behatárolható az elmozdulás szöge, amit továbbküld a jelgenerátornak. A mikrovezérlő szintén C nyelvű programja a kapott szögállás alapján kiszámolja a szervomotorhoz továbbítandó, az adott pozícióhoz tartozó kitöltési tényezőjű jelet. Végül a kamera középpontja beáll a körközéppontba, és ezt addig folytatja, amíg a tárgy a kamera látószögében helyezkedik el.

Az elkészített robotágens könnyen továbbfejleszthető. Egy lehetséges továbblépési lehetőség, ha a robot beavatkozója nem csak x irányba lenne képes mozogni, hanem az y irányú mozgásra is fel lenne készítve. Így 2D-s mozgást tudna végrehajtani. Ehhez szükséges egy újabb motor, valamint a két szabadsági fokú mozgást megvalósító mechanika. Egy másik fejlesztési lehetőség, ha a pontvezérlést egy fejlettebb robotirányítási módszerrel helyettesítjük. Ezáltal a szervomotor mozgása a pontosabb szabályzásnak köszönhetően egyenletesebbé válna.

Irodalomjegyzék

[1] Xie, Ming. *Fundamentals of Robotics Linking Perception to Action.* World Scientific Publishing Co. Pte. Ltd. Singapure. 2003.

[2] Norvig, Peter – Russell, Stuart J.. *Mesterséges Intellgencia Modern Megközelítésben.* Második kiadás. Panem Könyvkiadó. 2005.

[3] Tar József. *Mesterséges Mesterek 1 - Robotmúlttal fényes jövő?.* Új Alaplap. 1994. 2. szám: 37-39.

[4] Tar József. *Mesterséges Mesterek 2 - A „művezető„ megválasztása.* Új Alaplap. 1994. 3. szám: 35-37.

[5] Fazekas Attila, Kormos János. *Digitális képfeldolgozás matematikai alapjai (egyetemi jegyzet).* Debreceni Egyetem. 2001.

[6] Linux Tv. *Video for Linux Two API Specification.* http://linuxtv.org/downloads/video4linux/API/V4L2_API/v4l2spec/v4l2.pdf. Utolsó látogatás: 2009.04.29.

[7] Linux Tv. *V4L Wiki.* http://www.linuxtv.org/v4lwiki/index.php/Main_Page. Utolsó látogatás: 2009.04.29.

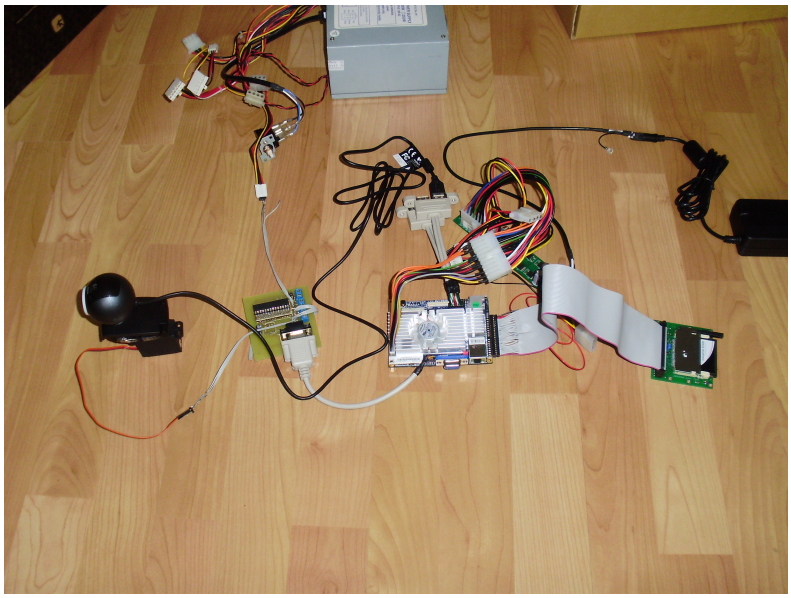
[8] Wikipedia. *HSL and HSV.* http://en.wikipedia.org/wiki/HSL_color_space. Utolsó látogatás: 2009.04.29.

[9] iLab. *HSV And H2SV Color Space.* http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space. Utolsó látogatás: 2009.04.29.

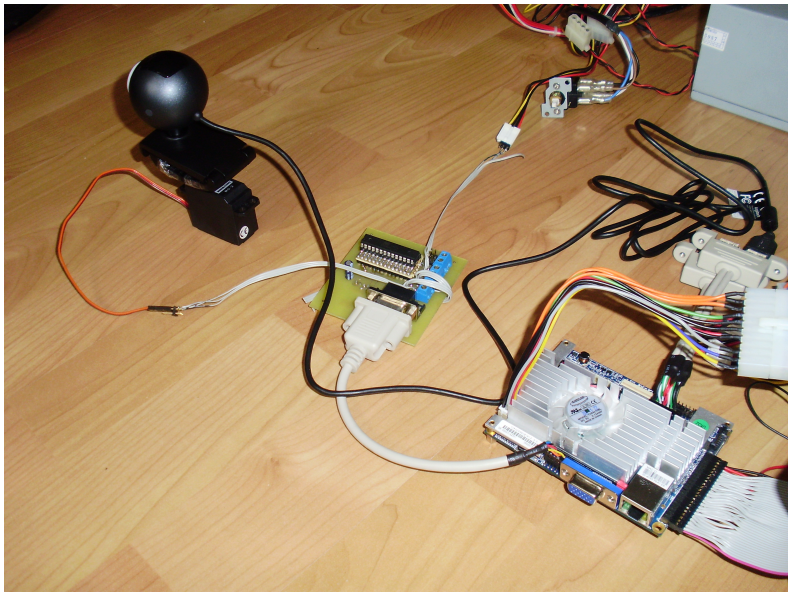
[10] Via. *Beágyazott rendszer.* http://www.via.com.tw/en/products/mainboards/. Utolsó látogatás: 2009.04.29.

[11] Atmel. *ATMEGA8 datasheets.* http://www.atmel.com/dyn/Products/product_card.asp?part_id=2004. Utolsó látogatás: 2009.04.29.

Melléklet



6.1. ábra. Ágensarchitektúra 1.



6.2. ábra. Ágensarchitektúra 2.