

OTDK DOLGOZAT

360 fokban látó omnidirekcionális mobil robot

Írta:

Németh Ferenc

(Villamosmérnöki Bsc szakos hallgató)

Konzulens:

Ballagi Áron

(tanszékvezető-helyettes, egyetemi adjunktus)

Széchenyi István Egyetem

2014

Győr

Tartalomjegyzék

Tartalomjegyzék.....	- 2 -
1. Bevezetés.....	- 4 -
2. Alapvető mobil robot hajtások	- 5 -
2.1. A differenciális meghajtás.....	- 5 -
2.2. A tengelykormányzású meghajtás.....	- 6 -
2.3. A mecanum meghajtás	- 6 -
2.4. Az omni meghajtás.....	- 8 -
3. Az omni platform	- 10 -
3.1. Háromkerekű omnidirekcionális robot vezérlése.....	- 10 -
3.2. Négykerekű omnidirekcionális robot vezérlése	- 12 -
3.3. A három- és a négykerekű változatok összehasonlítása.....	- 13 -
4. Körbelátó kamerarendszerek, kamera típusok	- 14 -
4.1. Forgatható tengelyre erősített kamera	- 14 -
4.2. Több kamera.....	- 15 -
4.3. Halszemoptika	- 16 -
4.4. Parabolikus, szférikus tükörrendszer.....	- 17 -
4.5. PAL rendszer.....	- 18 -
5. A robot hardvere.....	- 20 -
5.1. Az irányszámító egység	- 20 -
5.1.1. Az áramkör készítése	- 25 -
5.2. A képfeldolgozó egység	- 27 -
5.2.1. Központi egység.....	- 28 -
5.2.2. A kamera	- 29 -
5.2.3. Az optika	- 30 -
5.3. Energiaellátás	- 30 -
5.4. Kommunikáció	- 32 -
5.5. A robot szerkezeti felépítése	- 34 -
5.5.1. Az első koncepció	- 34 -
5.5.2. A végső koncepció	- 35 -
6. A robot szoftvere	- 38 -
6.1. Az irányszámító egység szoftvere.....	- 38 -
6.1.1. USART kommunikáció.....	- 38 -

6.1.2. PWM generálása	- 42 -
6.1.3. A főprogram	- 46 -
6.2. A képfeldolgozó egység szoftvere	- 47 -
6.2.1. A kamera képének közvetítése	- 48 -
6.2.2. Utasítások fogadása, kommunikáció a külvilággal	- 49 -
7. A távvezérlés	- 52 -
7.1. A kezelőfelület felépítése	- 52 -
7.2. A kezelőfelület belső működése.....	- 53 -
8. Jövőbeli tervek	- 56 -
9. Összefoglalás.....	- 57 -
10. Irodalomjegyzék.....	- 58 -
11. Melléklet.....	- 60 -
11.1. Az Arduino kapcsolási rajza	- 60 -
11.2. Az Adafruit motorvezérlő kapcsolási rajza	- 61 -
11.3. Az irányszámító egység kapcsolási rajza	- 62 -
11.4. Az irányszámító egység NYÁK terve	- 63 -
11.5. A Raspberry PI kapcsolási rajza.....	- 64 -
11.6. Az irányszámító egység kódja.....	- 69 -
11.7. A képfeldolgozó egység kódja	- 72 -
11.8. A kezelőfelület kódja	- 73 -

1. Bevezetés

Németh Ferenc vagyok, villamosmérnök hallgató a Széchenyi István Egyetemen. Érdeklődési körömbbe tartoznak az automatizálási rendszerek, robottechnika, gépi tanulás és látás, valamint a programozás témakörök.

Napjainkban különösképpen népszerűek a különböző önműködő járművek, mobil robotok. Ezek a szerkezetek mind más-más megvalósítást használnak a mozgásra. Van, amelyik lánctalpakon gurul, van, amelyik kerekeket használ a helyzete változtatására, de még olyanok is vannak, amelyek az emberhez, illetve állatokhoz, rovarokhoz hasonlóan kettő, négy vagy akár hat lábon mozognak. E dolgozat is egy sajátos platform tervezését, valamint vizsgálatának a folyamatát mutatja be.

A dolgozat célja egy olyan különleges, háromkerekű mobil robot fejlesztése, mely képes – anélkül, hogy a saját tengelye mentén elfordulna – a sík bármely irányába azonnal elmozdulni. Mindezt a speciális kialakítása és különleges kerekeinek köszönhetően.

Nem csak maga a robot, de annak távirányítása is sajátos módon történik. A roboton egy olyan kamerarendszer van elhelyezve, mely segítségével a robot és az azt irányító személy képes 360 fokban körbelátni. Így a sajátos, mindenirányba való elmozdulás, egy sajátos, mindenirányban látó, s tájékozódó rendszerrel egészül ki.

A dolgozatban a fent említett egységek felépítését, valamint konkrét működését, és ezeknek a fejlesztésének a menetét egy lehetséges alkalmazáson keresztül mutatom be.

2. Alapvető mobil robot hajtások

A mobil robotok meghajtására és mozgására rengeteg, különböző megoldás létezik, mindegyiknek más-más előnye és hátránya van a többivel szemben. Ebben a fejezetben ezekről a hajtásokról lesz szó.

2.1. A differenciális meghajtás

A differenciális meghajtás a legelterjedtebb megoldás a robotikában. Működése rendkívül egyszerű, a mozgást két egymástól függetlenül mozgó kerék végzi, a robot két ellentétes oldalán. Az irányváltoztatást a két kerék fordulatszámának változtatásával lehet véghezvinni. Az ívelt mozgáshoz az egyik kereket nagyobb, míg a másikat kisebb fordulatszámmal kell működtetni. A robot saját tengelye menti forgáshoz a két kereket azonos fordulatszámmal, de ellentétes irányba kell vezérelni.



1. ábra. Különféle differenciális meghajtású mobil robotok.

Az 1. ábrán látható három példa az ilyen jellegű robotokra.

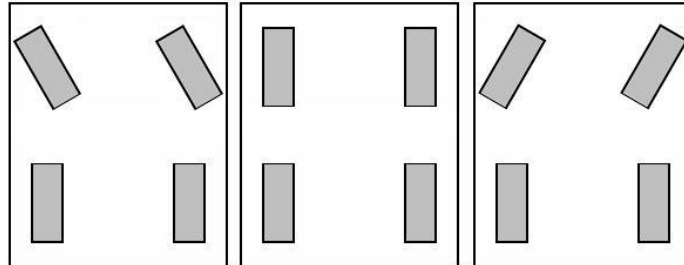
Az első egy *Khepera III* típusú robot. Két differenciális kerék van a robot hátsó részén, plusz elől egy közös támaszték, mely a robot stabilitását látja el.

A második egy *Neobotix MP-500* típusú robot. Jól láthatóak a differenciális kerekek, és hogy ezen kívül van még egy harmadik, szabadon mozgó kereke is. Ez – a *Khepera*-hoz hasonlóan – a mobil robot stabilitását hivatott ellátni.

A harmadik egy sajátos megvalósítás. Az előző kettőhöz hasonlóan ez is differenciális meghajtású, viszont itt a harmadik, szabadon mozgó kerék hiányzik. Ehelyett a robot stabilitását különféle szenzorok végzik. Ha a robot az egyik irányába megpróbál eldőlni, akkor azt a két kereke segítségével korrigálja. Itt a robot gyakorlatilag egy speciális inverz inga rendszerként viselkedik.

2.2. A tengelykormányzású meghajtás

Ez a típusú meghajtás nem túl népszerű a robotika területén. Használata bonyolult, és sok számítást igényel. Az ilyen jellegű eszközöknek négy, vagy akár három kerekeik vannak, s az elülső, vagy a hátsó kerekek közös tengelyének a kormányzásával lehet irányt változtatni.

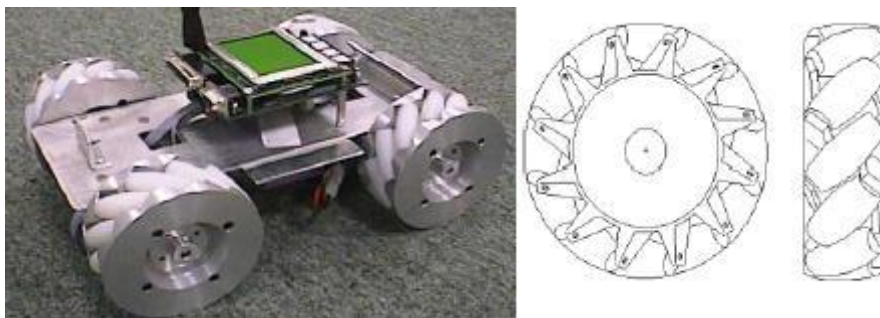


2. ábra. A tengelykormányzású meghajtás.

Ahogy a 2. ábrán is látható, a robot az első két kerekét képes mozgatni, az irányváltoztatást ezek segítségével oldja meg. A két szélső esetben ívelten kanyarodik a robot, míg a középsőn egyenesen halad előre. Ha közvetlenül oldalra kellene irányítani a robotot, akkor bonyolult módon kellene manőverezni, hiszen a tengelye mentén képtelen elfordulni az ilyen típusú robot és oldalazó mozgást sem tud végezni.

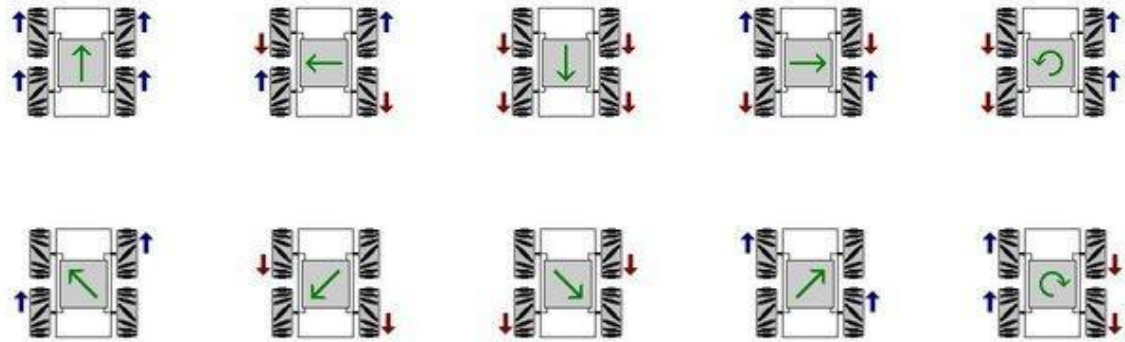
2.3. A mecanum meghajtás

Egy különleges, alternatív meghajtás a mecanum meghajtás. Ezek a különleges kerekek több, kisebb, szabadon mozgó görgőből állnak, melyek átlósan vannak elhelyezve a keréken.



3. ábra. Egy mecanum meghajtású robot és a kerék sematikus rajz. (forrás: [1]).

A mecanum mobil robotok hasonlóan működnek, mint a differenciális meghajtásúak, itt is mindegyik kerék egy-egy különálló motorhoz csatlakozik, itt viszont mindenképpen négy kerékre van szükség, még hozzá a speciális mecanumkerékre. Ezekkel a kerekekkel a robot képes oldalirányú mozgást is végezni. A mozgása attól függ, hogy a négy keréknek milyen irányú és nagyságú a fordulatszáma. A 4. ábra ezt a mozgást jól szemlélteti.



4. ábra. A mecanum robot mozgása.

Az ábrán egyértelműen látható, miért is olyan előnyösek az ilyen kialakítású mobil robotok, könnyen és sokkal gyorsabban tudnak oldalirányú mozgást végezni. Hátrányuk viszont, hogy a kerekek kialakítása miatt drágák.

2.4. Az omni meghajtás

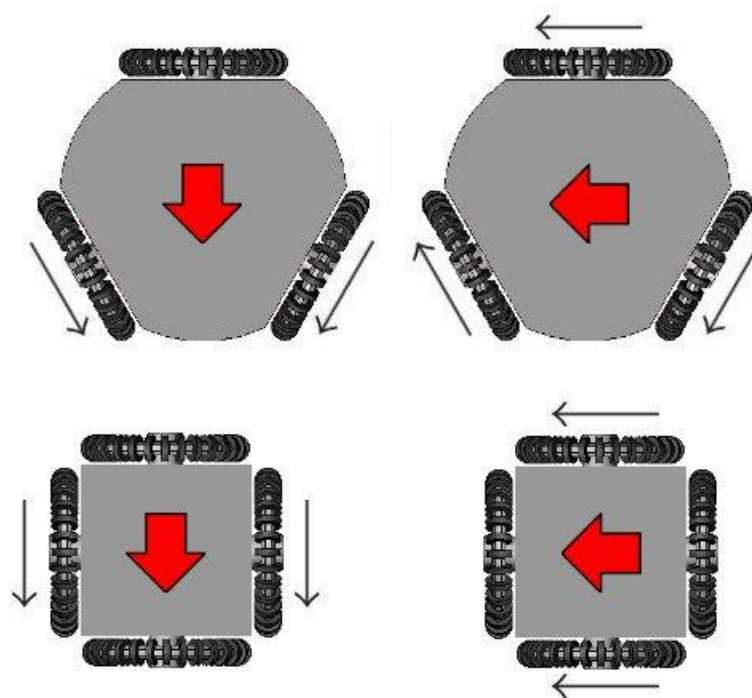
Az omni hajtás – a mecanum-hoz hasonlóan – szintén egy különleges, alternatív megoldás.



5. ábra. A speciális omnikerekek

Az omnidirekcionális mobil robot egy olyan különleges robot, mely képes a sík bármely irányába egyenesen haladni, anélkül, hogy a mobil robot testét a mozgást megelőzően el kellene forgatni. Ezeknek az eszközöknek hatalmas előnye a hagyományos, differenciális meghajtású mobil robotokkal szemben, hogy az omnidirekcionális robotoknak sokkal nagyobb a manőverezhetőségük, valamint gyorsabb és pontosabb lehet az irányításuk. Ezzel párhuzamosan természetesen a vezérlésük bonyolultabbá válik, az egyes motorok fordulatszámainak kiszámítása összetettebb módon történik. [2]

Általában három- vagy négykerekű kialakítás lehetséges. Mindkét típusnak megvannak az előnyei, illetve hátrányai.



6. ábra. Az omni robot mozgása.

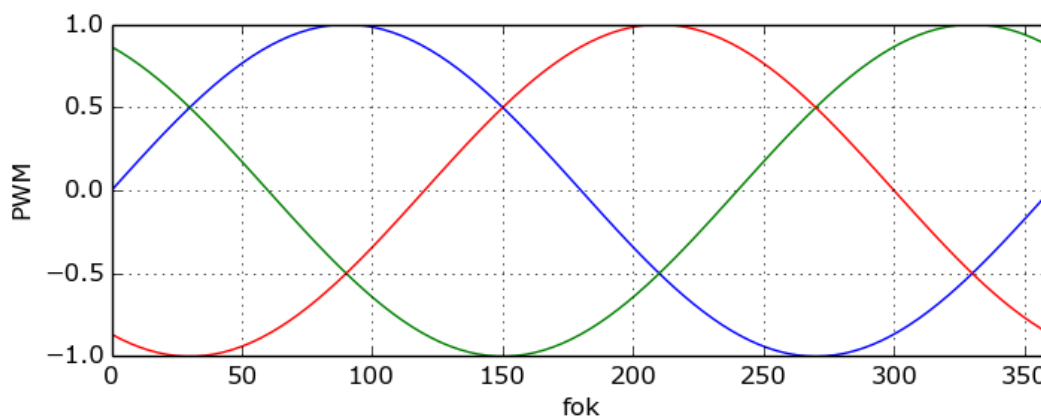
A 6. ábrán néhány példa van az omnidirekcionális robot mozgására. Részletes működés és magyarázat a 3. fejezetben olvasható majd.

3. Az omni platform

Az első fejezetben megismert omnidirekcionális meghajtásra épülő mobil robot platform részletes működéséről szól ez a fejezet.

3.1. Háromkerekű omnidirekcionális robot vezérlése

Az e fajta robotok vezérlése több számítást igényel, mint a hagyományos változatoké. A számítás során figyelembe kell venni a kerekek egymáshoz való viszonyát, ebben az esetben azt, hogy egymáshoz képest 120 fokban helyezkedik el ez a három kerék. Ezt jól szemlélteti a 7. ábra.



7. ábra. A PWM kitöltési tényező százalékos aránya a kívánt irány szögének függvényében.

A fenti ábra ismeretében leolvasható, hogy milyen nagyságban kell meghajtani az egyes motorokat. Például, ha 0 fokban, tehát „előre”, akkor az egyes motort 0 %-kal, míg a kettes motort 86 %-kal, a hármasat pedig -86 %-kal. Ha pedig 90 fokban „jobbra”, akkor a felső, egyes motort 100 %-os maximális teljesítménnyel, a kettes és hármas motorokat pedig 50 %-os teljesítménnyel, de mind a kettőt az egyes motorral ellentétes irányba.

A számítás képlet formájában az alábbi: [2]

$$\begin{bmatrix} n1 \\ n2 \\ n3 \end{bmatrix} = v * \begin{bmatrix} \sin(\varphi + 0) \\ \sin(\varphi + 120) \\ \sin(\varphi + 240) \end{bmatrix} \quad (1)$$

ahol:

- $n1; n2; n3$: a három motor PWM kitöltési tényezőjének százalékos értéke
- v : a robot sebessége
- φ : a kívánt szög

Az előző 0 fokos példa számszerűsítve:

$$\begin{bmatrix} 0 \\ -86 \\ -86 \end{bmatrix} = 100 * \begin{bmatrix} \sin(0 + 0) \\ \sin(0 + 120) \\ \sin(0 + 240) \end{bmatrix} \quad (2)$$

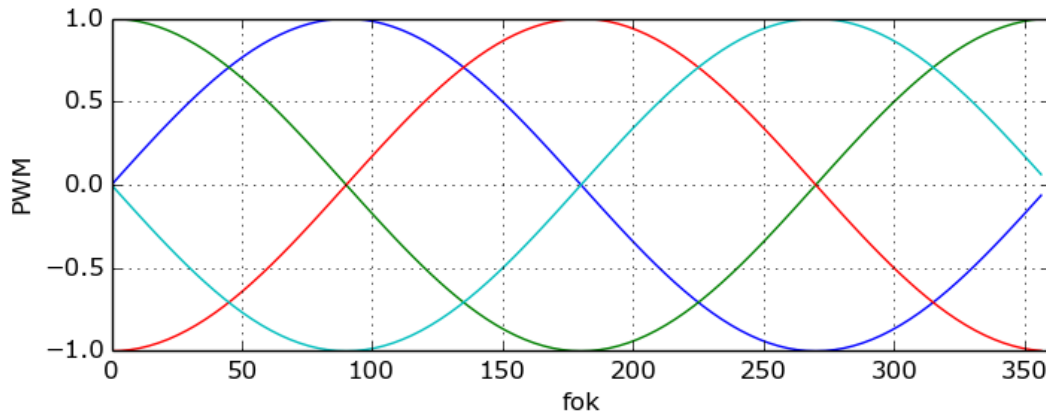
És a 90 fokos:

$$\begin{bmatrix} 100 \\ -50 \\ -50 \end{bmatrix} = 100 * \begin{bmatrix} \sin(90 + 0) \\ \sin(90 + 120) \\ \sin(90 + 240) \end{bmatrix} \quad (3)$$

Látható, hogy mindkét esetben ugyanaz lett az eredmény, mint az ábráról leolvasva.

3.2. Négykerekű omnidirekcionális robot vezérlése

A négykerekű változatnak - ugyanúgy, mint a háromkerekű párjánál, - szintén bonyolultabb, számításigényesebb a vezérlése. Itt a kerekek egymáshoz képest 90 fokos szögben helyezkednek el.



8. ábra. A PWM kitöltési tényező százalékos aránya a kívánt irány szögének függvényében.

A korábbi példákból kiindulva, ha 0 fokban „előre” szeretnénk meghajtani, akkor a két oldalsó motort maximális fordulatszámmal kell meghajtani, míg a két elülsőt 0 %-kal. 90 fokban pedig pont fordítva, a két elülső motort 100 %-kal, az oldalsókat 0 %-kal kell működtetni.

A vezérlés képlet formájában az alábbi:

$$\begin{bmatrix} n1 \\ n2 \\ n3 \\ n4 \end{bmatrix} = v * \begin{bmatrix} \sin(\varphi + 0) \\ \sin(\varphi + 90) \\ \sin(\varphi + 180) \\ \sin(\varphi + 270) \end{bmatrix} \quad (4)$$

A 0 fokos példa behelyettesítve:

$$\begin{bmatrix} 0 \\ 100 \\ 0 \\ -100 \end{bmatrix} = 100 * \begin{bmatrix} \sin(0 + 0) \\ \sin(0 + 90) \\ \sin(0 + 180) \\ \sin(0 + 270) \end{bmatrix} \quad (5)$$

A 90 fokos pedig: [2]

$$\begin{bmatrix} 100 \\ 0 \\ -100 \\ 0 \end{bmatrix} = 100 * \begin{bmatrix} \sin(90 + 0) \\ \sin(90 + 90) \\ \sin(90 + 180) \\ \sin(90 + 270) \end{bmatrix} \quad (6)$$

Mind az ábrából, mind pedig a képletből látható, hogy a szemközti motorokat ugyanolyan nagyságú, de ellentétes irányú erővel kell meghajtani. Ennek köszönhetően a két-két szemközti motort akár ugyanazzal a vezérlőjellel is lehet irányítani, pusztán ellentétesen kell bekötni a szemközti motorok pólusait. Így a négykerekű változat az alábbi módon egyszerűsíthető:

$$\begin{bmatrix} n1 \\ n2 \end{bmatrix} = v * \begin{bmatrix} \sin(\varphi + 0) \\ \sin(\varphi + 90) \end{bmatrix} \quad (7)$$

3.3. A három- és a négykerekű változatok összehasonlítása

A vezérlés megismerése után már csak az a kérdés, hogy melyik változat használata a célszerűbb.

A háromkerekű kevesebb eszközt igényel, ezért a megépítése gyorsabb és olcsóbb. A háromkerekűnek kisebb az energiaigénye, hisz négy motor helyett csak hármat kell árammal ellátni. A négykerekűnek egyszerűbb a vezérlése, hiszen az egyszerűsített képlet értelmében csak két számítást kell elvégezni. Ha a négykerekű egyenetlen terepen van, akkor megeshet, hogy az egyik kerék nem érintkezik a talajjal, így nem is tud szerepet játszani a mozgásban, tehát a robot nem olyan szögben fog elmozdulni, mint ahogy kellene, három kerékkel ez a hiba nem áll fent, a kerekek minden esetben érintkezni fognak a talajjal.

Ezen információk tükrében a három kerék jobb választás, mint a négy.

4. Körbelátó kamerarendszerek, kamera típusok

Egy kamera csak a közvetlenül előtte lévő tárgyakat látja. Ahhoz, hogy a robot képes legyen minden irányba látni, egy összetettebb optika- vagy kamerarendszerre van szükség.

Ebben a fejezetben a különböző optikai és képalkotásbeli trükkök, módszerek kerülnek bemutatásra. A cél, megtalálni a robot számára leghatékonyabb rendszert.

4.1. Forgatható tengelyre erősített kamera

Az ilyen jellegű rendszereket talán mindenki ismeri. Lényegében egy egyszerű kamera, vagy fényképezőgép van elhelyezve egy forgatható tengelyen, amelyet vagy automatikusan, vagy manuálisan forgatnak a képfelvétel közben.



9. ábra. A rendszer működése.

Könnyen belátható, hogy ez a fajta körbelátó kamerarendszer nem alkalmas a robottechnikai alkalmazásra, hiszen időigényes, amíg a robot körbefotózza a környezetét. Ráadásul, ha még közben mozog is, az a végeredmény teljes elnyúlását, elmosódását eredményezné. Egy ilyen kialakítás nem lehet alkalmas előkép közvetítésére.

4.2. Több kamera

Ez a módszer több kamerát használ, melyek mindegyike más-más irányba néz. Az előző koncepcióhoz képest előnye, hogy itt már minden fix, nincs mozgó, forgó komponense, emiatt ténylegesen képes élőképet közvetíteni.

Ilyen rendszer például a Microsoft RingCam, melyet meeting-ek, találkozók rögzítésére, illetve stream-elésére találtak ki. [3]



10. ábra. A RingCam (forrás: [3]).

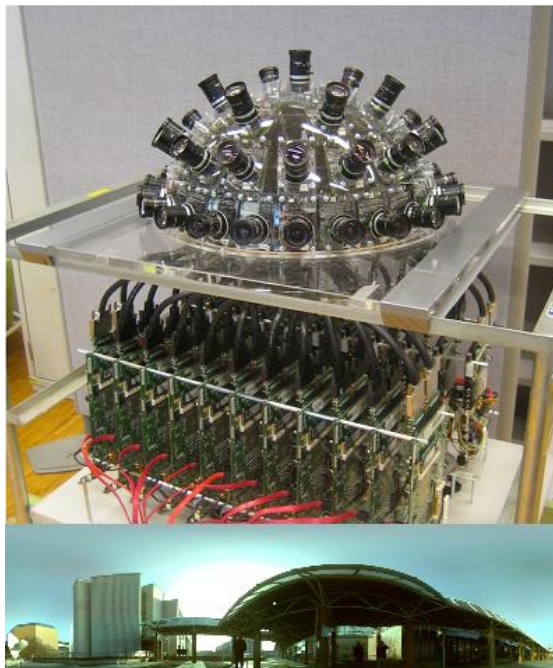
Ahogy a 10. ábrán is látható, a kialakítása meglehetősen egyszerű, egy kör mentén van elhelyezve több kamera, melyek mindegyike kifelé néz. Ezzel egy egyszerű omnidirekcionális kép- és hangrendszert alkottak.

Hasonló eszköz még a Lucent Technology FullView rendszere, mely szintén több kamera képét használja, de itt egy közös vetítési középpontú tükörrendszerrel van kiegészítve a modul. [4][5]



11. ábra. A FullView (forrás: [4]).

Ezeket a módszereket, természetesen lehet fokozni, hogy ne csak egy sík mentén körbe, hanem arra merőlegesen is lehessen látni. Ilyen rendszer a EPFL Panoptic nevű rendszere. [6]



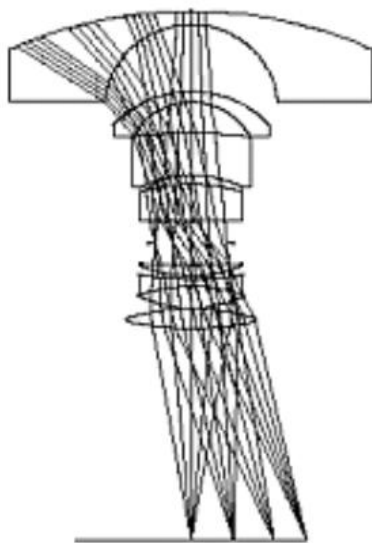
12. ábra A Panoptic rendszer (forrás: [7]).

Ez az omnidirekcionális kamerarendszert a biológia ihlette, a rovarok összetett szemét másolja. Ahogy az összetett szem, ez is sok-sok látóegységből áll, melyek egy félgömbön helyezkednek el. [6][7]

Sajnos a többkamerás rendszernek is megvan a hátránya. Olyan hardver, mely egyszerre sok kamera képét tudná kezelni és még kicsi, mobilis méretű is, sajnos megfizethetetlen. Ezt a problémát egyszerre több képfeldolgozó egység párhuzamos használatával ugyan lehetne orvosolni, de sem pénzügyileg, sem energetikailag nem a legjobb választás.

4.3. Halszemoptika

Mivel a kamerákkal való trükközés nem kivitelezhető a robot esetén, ezért érdemes annak optikájával megoldani a körbelátás problémáját.



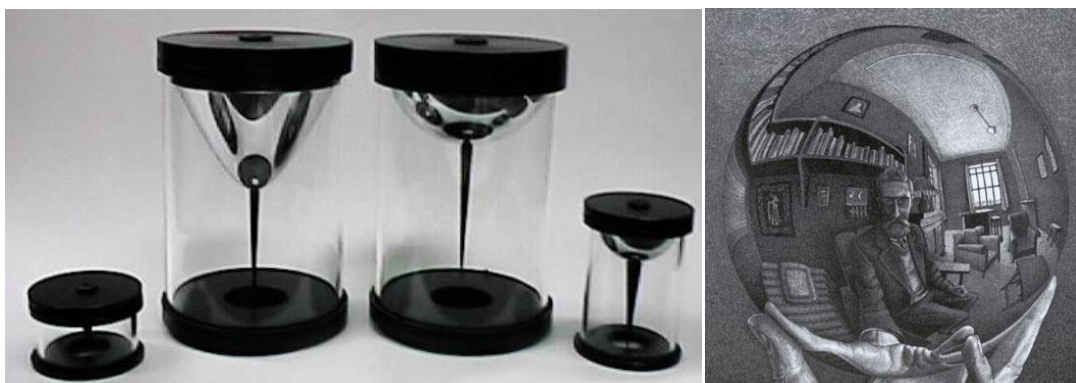
12. ábra. A halszemoptika (forrás: [5]).

A halszemoptika talán a legelterjedtebb objektív típus. Csillagászatban és fotográfiában használják leginkább. Segítségével a kamera képének látószöge megközelítőleg (fajtatól függően) 180 fok.

Ha a robot kamerájára egy ilyen optika van elhelyezve, azzal már egész jól belátható a környezete.

4.4. Parabolikus, szférikus tükörrendszer

Ezzel a módszerrel a közönséges kamera (és annak optikája) van vegyítve egy külső tükröződő felülettel. Ez a felület lehet gömb (szférikus) vagy kúp (parabolikus) alakú. Ahogy a kamera „belenéz” a vele szemközti, illetve az oldalt lévő tárgyakat fogja látni a tükörnek köszönhetően. [5]



13. ábra. A parabolikus és szférikus rendszer (forrás: [5],[7]).

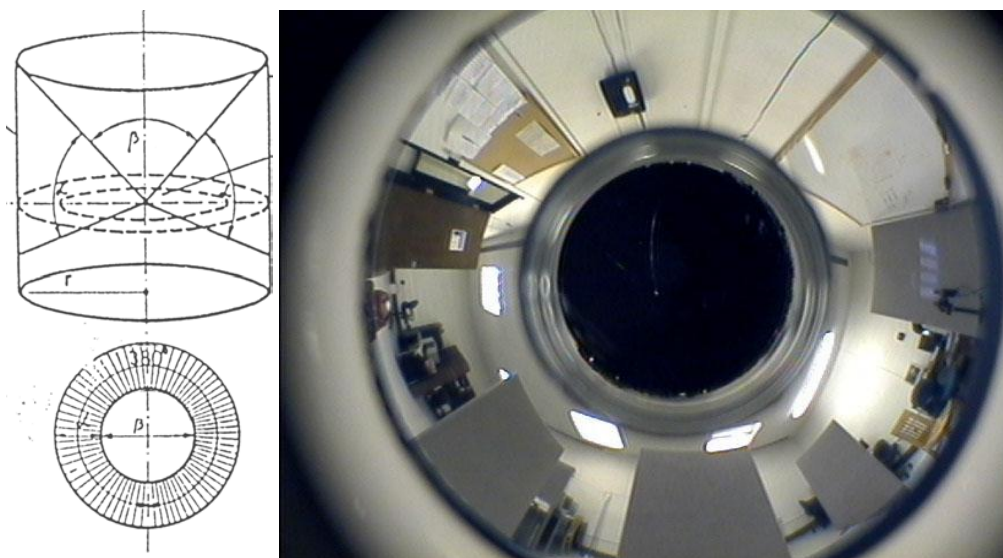
Ennek a módszernek a hatalmas előnye, hogy könnyedén, házilag is el lehet készíteni, a tükröződő felület egy sima karácsonyfadísz is lehet. Ilyen például a 14. ábrán látható minta is.



14. ábra. A házi szférikus rendszer.

4.5. PAL rendszer

A *Panoramic Annular Lens*, azaz Panoráma Gyűrűs Lencse, Dr. Greguss Pál, NASA-díjas biofizikus, egyetemi tanár nevéhez fűződik. [5][8]



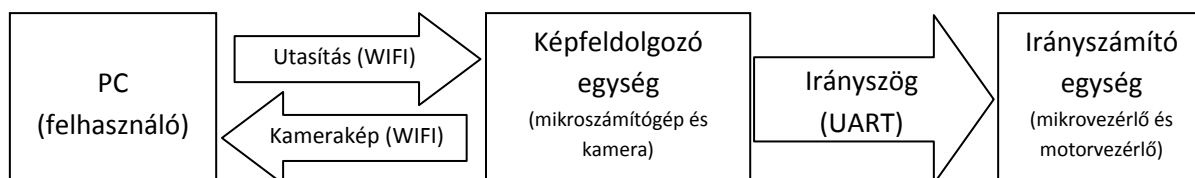
15. ábra. A PAL rendszer (forrás: [5], [8]).

Ez a rendszer azon az elven működik, hogy az optikát körülvevő világot hengeresen adja vissza, azaz egy úgynevezett *Centric Minded Imaging* (magyarul: körbelátó elvű) szemléletet követ. [8]

A CMI elv annyit jelent, hogy a háromdimenziós térben egy képzeletbeli henger felületén belül elhelyezkedő pontok a kétdimenziós képtérben egy sík mentén képződnek le. Ezt a rendszert először a haditechnikában alkalmazták, ahol periszkóp körbemozgatását ki lehetett váltani ezzel a rendszerrel. A későbbiekben pedig az űrkutatásban vetették be. [8]

5. A robot hardvere

Ebben a fejezetet a robot hardverének működéséről és felépítéséről lesz szó. A 16. ábrán látható a robot működésének elve, annak blokkvázlata. Az egyes részegységek működését a következő alfejezetek taglalják.



16. ábra. A robot működésének elve.

5.1. Az írányszámító egység

Ennek a modulnak a feladata, hogy az UART modulon át kapott írányszög alapján kiszámolja az egyes motorok PWM kitöltési értékét (a 3.1. fejezetben ismertetet módszer alapján), majd a motorvezérlő IC-k segítségével működésbe hozza a három motort.

A robot fejlesztésének korai szakaszában ezt a feladatot egy Arduino mikrovezérlő panel és egy Adafruit motorvezérlő shield biztosította.



17. ábra. Az Arduino.

Az Arduino belsejében egy ATmega328P típusú mikrovezérlő található. A fejlesztőpanel 14 digitális ki- és bemenetet tartalmaz, melyből 6 PWM-ként használható és további 6 analóg bemenete van. Az órajelet egy 16MHz-es rezonátor adja. A panelon található még egy USB csatlakozó és egy FTDI USB-UART chip, mely lehetővé teszi a

mikrovezérlővel való kommunikációt és a bootloader-es programozást. A panel kapcsolási rajza a 11.1. mellékletben található.

Az Adafruit motorvezérlő shield egy kiegészítő modul az Arduino mikrovezérlőhöz, melyet annak tetejére „ráhúzva” lehet összekapcsolni. Segítségével akár négy darab DC motort is meg lehet hajtani. Benne két darab L293D típusú kettős H-hidas motorvezérlő IC található. Mindkét IC két-két darab bidirekcionális DC motort képes vezérelni.

1. táblázat. A motorvezérlő IC igazságtáblája. [9]

EN	1A	2A	funkció
H	L	H	óramutató járásával megegyezően forog
H	H	L	óramutató járásával ellentétesen forog
H	L	L	stop
H	H	H	stop
L	X	X	stop

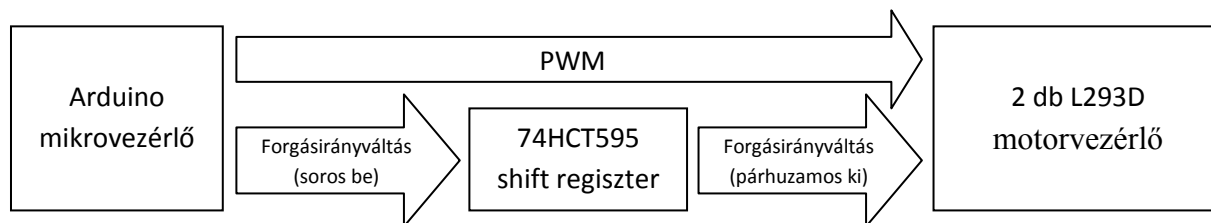
ahol: - H: logikai magas szint (high)
- L: logikai alacsony szint (low)
-X: nem számít (don't care)

Ahogy az 1. táblázat is mutatja a, motorokon egy-egy PWM kimenetre (EN) van szükség, melyek a motor fordulatszámát szabályozzák és két-két digitális kimenet (1A, 2A), melyek a forgásirányt határozzák meg. Így egyetlen motor irányításához a mikrovezérlő három kimenetét kell használni. Három motor esetén ez már kilenc használt kimenetet jelent, mely a robot kezdeti stádiumában ugyan megengedhető, de a későbbiekben, ahol már szenzorok és egyéb eszközök is bekerülhetnek a rendszerbe, már nem.

Természetesen a kimenetek számát lehet csökkenteni, úgy, hogy az EN lábakat a Vcc-re kötve állandóan logika magas állapotba tesszük, és az 1A, 2A csatlakozók pedig egy-egy PWM lábra kapcsolódnak. Ezzel a módszerrel három kimenetet lehet spórolni (motoronként egyet) ugyan, de így a motoronkénti egy-egy PWM helyett motoronként kettő kell, tehát összesen hat, ami adott esetben rosszabb, mint az eredeti koncepció szerinti.

Ekkor jön a képbe a motorvezérlő panel igazi előnye. A motorvezérlő IC-k nem közvetlenül a mikrovezérlő kimeneteire vannak kötve, hanem egy 74HCT595 típusú 8 bites shift regiszterre. A shift regiszter soros be- és párhuzamos kimenetű.

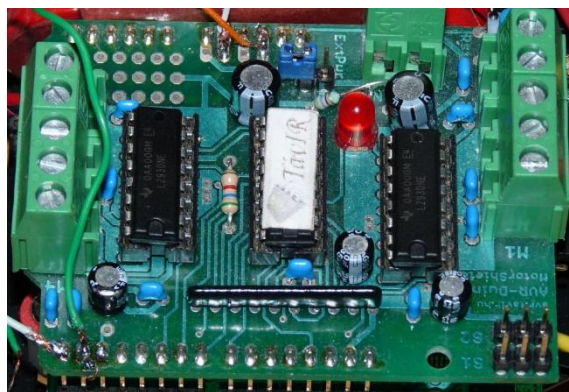
Az IC-nél lényegében bitenként egymás után küldjük az adatokat, s a 8 adatbit beérkezése után az adatbájt értékei egyszerre jelennek meg a 8 kimeneti (Q0, Q1, ... Q7) lábakon.



18. ábra. A motorvezérlő áramkör blokkvázlata.

A 18. ábra értelmében a shift regiszter segítségével történik az egyes motorok irányainak megadása (1A, 2A, stb. csatlakozók). A PWM (EN) csatlakozók pedig közvetlenül kapcsolódnak a motorvezérlő IC-kbe. Ezzel a módszerrel gyakorlatilag a shift regiszter négy vezérlőjelével (soros adat bement (DS), shift regiszter órajel (SHCP), tároló regiszter órajel (STCP), kimenet engedélyező (OE)), plusz a három PWM csatornával összesen hét kimentre redukálódott az eredeti kilenc kimenetes változat.

Ha egy további negyedik DC motor becsatlakoztatására lenne szükség, akkor csupán egy PWM kimenetet kellene felhasználni, így még mindig csak nyolc csatlakozó lenne használatban az eredeti tizenkettő helyett.

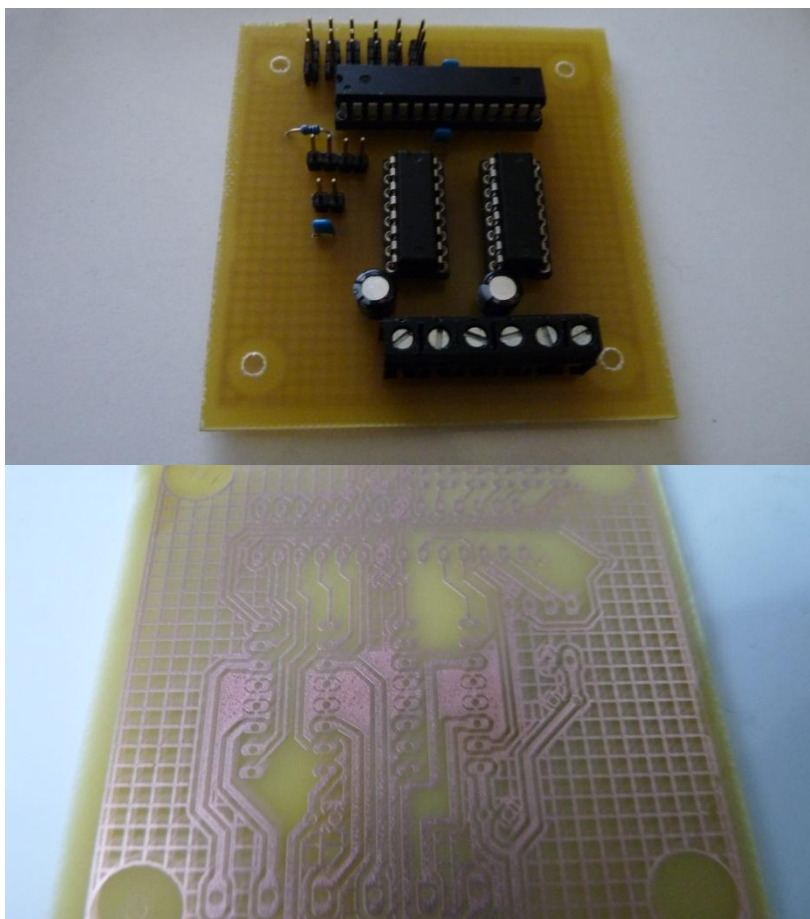


19. ábra. A motorvezérlő.

A motorvezérlő lehetőséget biztosít arra, hogy a motorokat és a vezérlést külön-külön feszültségforrásról lehessen üzemeltetni, így egyrészt a motorok nagyobb feszültségről tudnak üzemelni, mint a mikrovezérlő, másrészt pedig a motorok által keltett elektromos zaj sem befolyásolja a vezérlést.

A teljes motorvezérlő áramkör kapcsolási rajza a 11.2. mellékletben látható.

A robot végső változatában célszerűbb volt egy saját panel létrehozása, mely sokkal kisebb és kompaktabb, mint az előbb bemutatott megvalósítás.



20. ábra. *A sajátfejlesztésű modul.*

A panel készítése során figyelembe vett főbb vonások, elvek az előbb bemutatott fejlesztőpanelekről lettek átemelve. Kapcsolási rajza a 11.3. mellékletben, NYÁK terve pedig a 11.4. mellékletben.

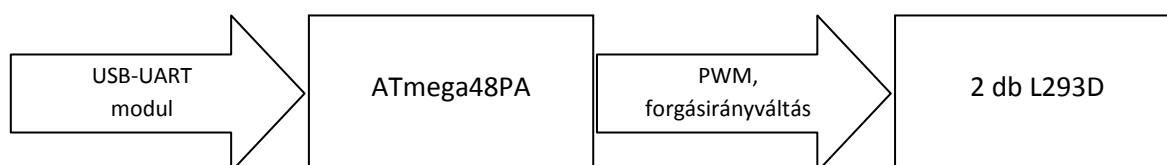
A vezérlést egy ATmega48PA mikrovezérlő látja el, mely (mint az Arduino-ban található ATmega328P is) az ATmegaX8-as szériába tartozik. A 8-as család összehasonlítását a 2. táblázat mutatja.

2. táblázat. *Az ATmegaX8 széria.*

Eszköz	Flash	EEPROM	RAM	Megszakításvektor méret
ATmega48PA	4kB	256B	512B	1 utasítás/vektor
ATmega88PA	8kB	512B	1kB	1 utasítás/vektor
ATmega168PA	16kB	512B	1kB	2 utasítás/vektor
ATmega328P	32kB	1kB	2kB	2 utasítás/vektor

A táblázatból látható, hogy a lényegi különbség az egyes típusok között csupán a memória méretben van. Mivel a végső program mérete alig több, mint 2 kB, ezért célszerű volt a család legkisebb tagját választani.

A két motorvezérlő IC ugyanúgy az L293D áramkörök lettek. Ezek korábban is tökéletesen végezték a dolgukat, nincs értelme váltani. Viszont az Adafruit motorvezérlőhöz képest változás, hogy ebből a verzióból kimaradt a shift regiszteres megoldás. Nem volt értelme spórolni a lábakkal, hiszen semmi olyan nem került a kapcsolásba, ami ezt megindokolta volna, így viszont a kód szempontjából egyszerűsödött a dolog.



21.ábra. A kapcsolás blokkvázlata.

Maga a kapcsolás igen egyszerű. Az irányszöveget UART-on (a kommunikáció menete később, a 5.4. fejezetben olvasható) keresztül kapja a mikrovezérlő, így a TX és RX lábak kivezetésre kerültek egy tűskesorba, a kapcsolás tápfeszültségével (5 V) és földjével egyetemben. Ahogy korábban, most is célszerűbb volt a motorokat külön tápforrásról üzemeltetni (az energiaellátás a 5.3. fejezetben található), ezért ennek a pozitív (6 V) és a negatív pólusai is egy-egy tűskébe lettek kivezetve. Természetes a két-két GND összekötve, hogy azonos potenciálon legyenek.

A mikrovezérlő reset lába egy 10 kΩ-os ellenálláson keresztül az 5 V-ra lett kötve. Az IC mindkét oldalán a pozitív és negatív lábak egy-egy 10 nF-os kondenzátorral vannak stabilizálva. A motorvezérlők tápcsatlakozójánál szintén egy 10 nF-os kondenzátor található a stabilitás érdekében, illetve mindkét IC bemenete mellett közvetlenül egy-egy 100 µF-os kondenzátor található.

A mikrovezérlő 6 ADC (amiből kettő I²C csatlakozóként (SDA és SCL) is funkcionál) bemente (és mindegyik mellé egy plusz-mínusz páros) egy tűskesorba került kivezetésre, az esetleges szenzoros bővítés érdekében.

A mikrovezérlő lábai és csatlakozói aszerint lettek kiválasztva, hogy lehetőleg mindegyik egy oldalon legyen, mindegyik a motorvezérlők felé nézzen, így elkerülve a feleslegesen hosszú vezetősávokat, valamint az esetleges átkötéseket.

A motorvezérlő IC motorkimenetei egy-egy sorkapocsba vannak kivezetve, hogy a motorok könnyen tudjanak a panelhez csatlakozni. Természetesen ezek a vezetősávok (a

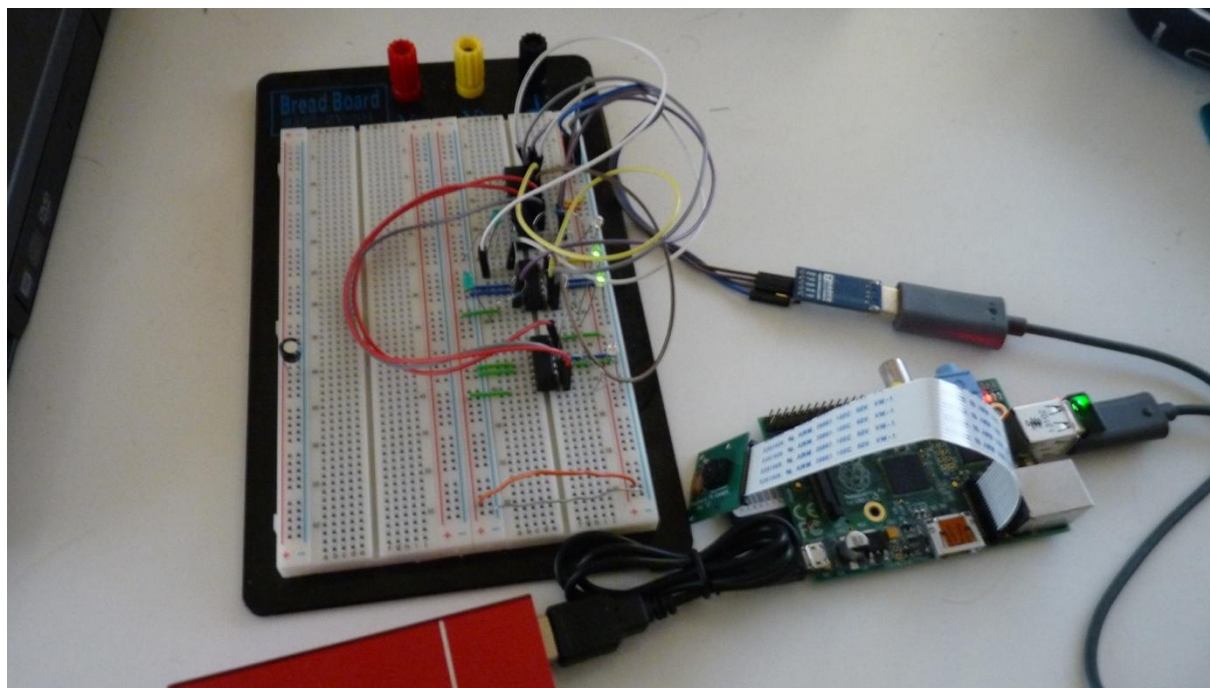
motorvezérlő tápjával együtt) vastagabbak, mint a többi, hisz ezeken jóval nagyobb áramok haladnak át.

A két motorvezérlő GND csatlakozóit (IC-nkét 4 darab) egy-egy nagyobb GND sziget veszi körül, hogy a vezérlés következtében keletkező hőfelesleg könnyedén eloszlassa. Ez magában a motorvezérlő adatlapjában javasolja a gyártó. [9] Ez a kialakítás a korábbi 20. ábrán is jól látható.

A nyomtatott áramkör egésze egy nagy GND felülettel van bevonva, ami egyrészt ad egy plusz árnyékolást, másrészt a tervezés során nem kellett külön-külön mindenhova manuálisan odavezetni azt. Ez a GND felület (a motorvezérlőknél találhatóakkal ellentétben) nem egy egységes felület, hanem rácsos kialakítású. Ennek pusztán annyi az oka, hogy a nyomtatás során az extrém mennyiségű toner elhasználása nem valami gazdaságos megoldás, így kompromisszumként ez a stílus lett kiválasztva.

5.1.1. Az áramkör készítése

Még mielőtt az áramkör elkészítése megtörtént volna a kapcsolás egy forrasztásmentes próbapanelen lett tesztelve. Mindegyik motorkivezetés helyére egy-egy LED került (motoronként kettő, azaz hat darab), melyek fényereje az egyes motorok fordulatszámát szimbolizálták.

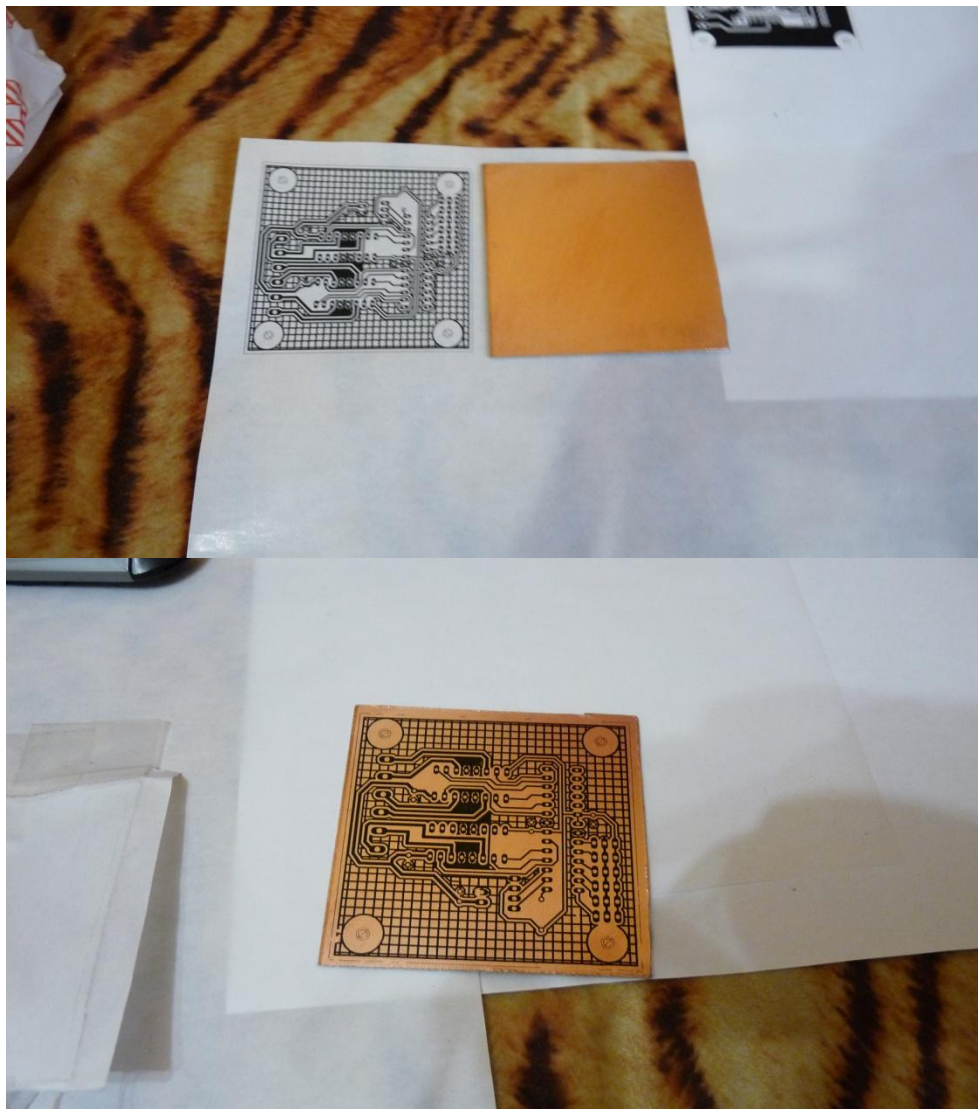


22.ábra. A fejlesztés.

Az 22. ábrán jól látható a két motorvezérlő és a mikrovezérlő. A képen éppen 0 fokban haladna a robot, tehát a háromból kettő motor aktív (kb. 86%-os nagyságban), de ellentétes irányban.

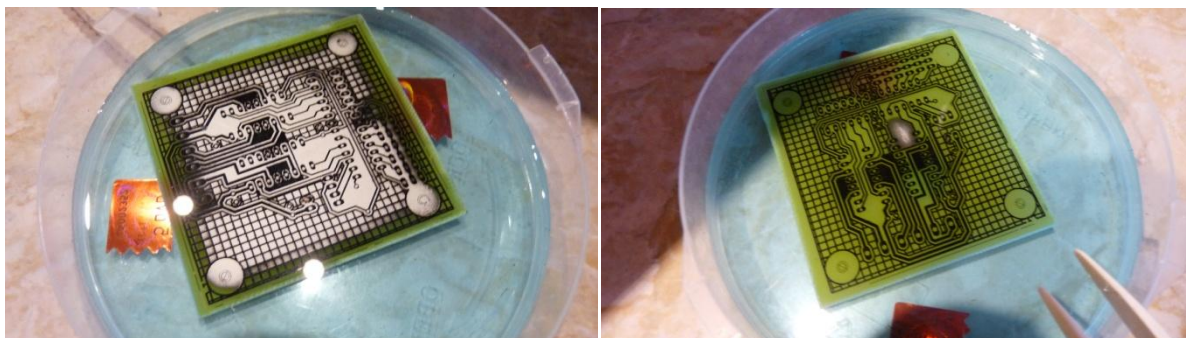
Miután minden tökéletesnek bizonyult, megkezdődhetett a nyomtatott áramkör tervezése, mely a *CadSoft EAGLE PCB Design Software*-ben történt.

A nyomtatott áramkör készítés első lépése, hogy az áramkör rajzot fel kell vinni a még üres panelre.



23. ábra. A rajz felvitele.

Itt lényegében egy lézernyomtatóval kell kinyomtatni az áramkör rajzát, de úgy, hogy a tonert a papír ne szívassa fel. Ehhez nem a hagyományos fénymásoló papír kell, hanem műnyomó papír. Ezután rá kell melegíteni papírt, ehhez a legcélszerűbb egy lamináló gépet használni, ez a panelt és a lapot két oldalról melegíti. Szépen és pontosan ráviszi a panelre a rajzot. A következő lépés a maratás, mely a 24. ábrán látható.



24.ábra. A maratás

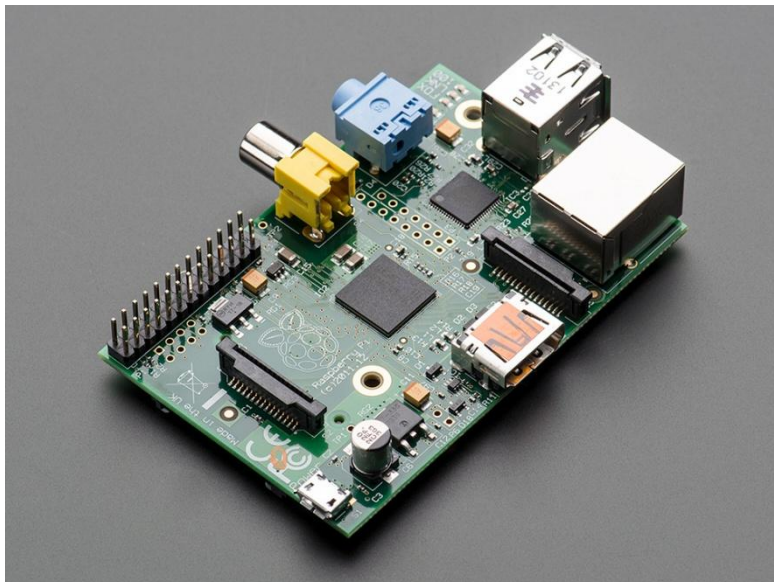
Ehhez nátrium-perszulfát, vagy más néven nátrium-peroxo-diszulfát ($\text{Na}_2\text{S}_2\text{O}_8$) került felhasználásra. Ezután pedig csak a fúrás és a forrasztás marad hátra, a végeredmény pedig látható volt a 20. ábrán.

5.2. A képfeldolgozó egység

Ez az egység - ahogy a neve is mutatja - végzi a képfeldolgozást, ez stream-eli a képet a számítógép és felhasználó számára. Három fő részből áll, a központi egység, melyben egy Raspberry PI panel található, a kamera és a panoráma optika.

5.2.1. Központi egység

A központi egység egy Raspberry PI model B, rev 2 mikroszámítógép található. A 25. ábrán a Raspberry panel látható, a főbb tulajdonságait pedig a 3. táblázat taglalja.



25.ábra. A Raspberry PI.

3. táblázat. A Raspberry PI tulajdonságai

SoC	Broadcom BCM2835
CPU	700 MHz ARM1176JZF-S core (ARM11 család, ARMv6 utasításkészlet)
GPU	Broadcom VideoCore IV @ 250 MHz OpenGL ES 2.0 (24 GFLOPS) MPEG-2 és VC-1, 1080p30 h.264/MPEG-4 AVC
Memória	512 MB (SDRAM), a GPU-val megosztva
Ki- és bement	HDMI, RCA, 3.5mm Jack, 2xUSB, 10/100 Mbit/s Ethernet (8P8C), MicroSD, 17xGPIO (UART, I2C, I2S, SPI, PWM), MIPI camera interface (CSI), MIPI display interface (DSI)

A mikroszámítógépen a Raspbian nevű RISC-alapú operációs rendszer fut. A Raspberry PI kapcsolási rajza a 11.5. mellékletben található.

5.2.2. A kamera

A környezetet pedig a hivatalos Raspberry kamera modul figyeli, melynek főbb tulajdonságait a 4. táblázat mutatja.

4. táblázat. A Raspberry PI kamera

Szenzor típus	OmniVision OV5647 Color CMOS QXGA (5-megapixel)
Szenzor méret	3.67 x 2.74 mm
Pixel szám	2592 x 1944
Pixel méret	1.4 x 1.4 μm
Video	1080p @ 30 fps (<i>megj.: stream-elve sajnos nagyságrendekkel rosszabb</i>)
Csatlakozó	CSI

5.2.3. Az optika

Az optika teszi lehetővé, hogy Raspberry és annak kamerája képes legyen 360 fokban körbelátni.



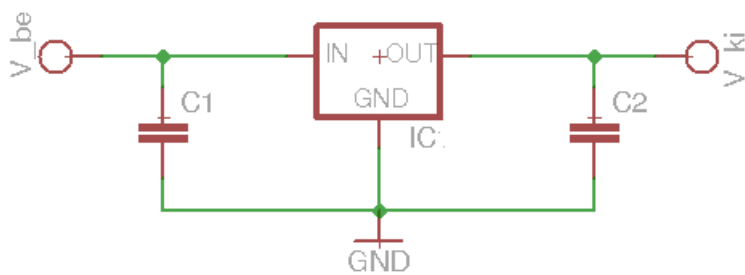
26.ábra. Az optika.

Működési elvét tekintve a már korábban, a 4.5. fejezetben említett PAL elv szerint működik.

5.3. Energiaellátás

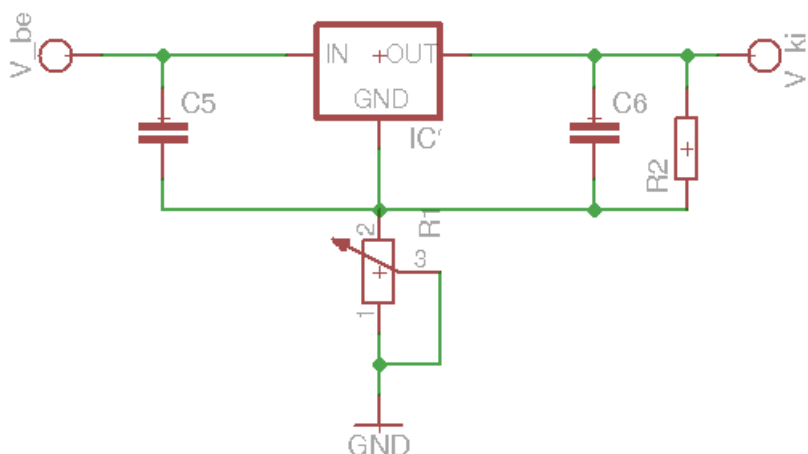
Már a robot kezdeti stádiumában is világossá vált, hogy az a célszerű, ha a vezérlőelektronika és hajtás külön-külön tápforrásról üzemel. Egyrészt így a motorok által keltett elektromos zaj és feszültségingadozások (még, ha stabilizáló kondenzátorok is vannak a rendszerben) véletlenül sem fogják befolyásolni (ezáltal az életciklusát csökkentve) a digitális áramköröket. Másrészt pedig ezáltal nagyobb feszültség és áram jut a motoroknak.

Kezdetben, amikor még a robotban az Arduino fejlesztőpanel volt, kényelmes volt a dolog, hiszen a beleépített feszültségszabályozó IC révén szinte bármilyen akkumulátort nyugodtan rá lehetett kötni. Ennek elhagyása miatt viszont már saját fejlesztésre volt szükség. Erre az KA7805 IC állt rendelkezésre, melynek a gyártója által javasolt kapcsolását a 27. ábra mutatja. [10]



27.ábra. A KA7805 adatlapjában ajánlott kapcsolás.

Ez a kapcsolás tökéletesen működött (stabil 5.01 V), mindaddig, amíg a Raspberry PI csatlakoztatásra nem került. Ekkor a feszültség szint 4.6-4.7 V lett, mely nem volt elégséges a rendszer bekapcsolásához. Nagyobb feszültségre volt szükség, szerencsére az adatlapban volt segítség. [10]



28.ábra. A megnövelt feszültség szintű kapcsolás

Ekkor a következő képlet szerint alakul a kimeneti feszültség:

$$V_{ki} = 5(1 + R_1/R_2) + I_Q R_1 \quad (8)$$

ahol:

- I_Q : a nyugalmi (Quiescent) áram, aminek 5mA az értéke

A kapcsolásból és képletből látható, hogy a kimeneti feszültség szintet az R_1 potenciométerrel lehet szabályozni. R_2 ellenállásnak 1 k Ω , R_1 -nek pedig 100 Ω rezisztenciák lettek kiválasztva. Ekkor a kimeneti feszültség szint (elméletben) 5 és 6 V között szabályozható. E módszer első próbálkozására tökéletes megoldást nyújtott, sikerült bekapcsolni a Raspberry PI panelt. Sajnos későbbi próbálkozások után, újra működésképtelen

lett a rendszer, ahogy merült az akkumulátor (10-15 perc használat után), a feszültség szint újra a minimális alá csökkent.

Ezután sikerült beszerezni egy 5 V-os *powerbank*-ot, melyet elsősorban telefonok, tablet-ek töltésére fejlesztettek ki. Ez a beleépített elektronikának köszönhetően minden körülmények között stabil feszültség szintet ad, így a probléma megoldódott. Ráadásul ez több szempontból is jobb: mivel telefonokhoz készült, ezért USB csatlakozóval rendelkezik, aminek köszönhetően közvetlenül egy USB kábellel összeköthető a Raspberry PI-vel, mivel Li-ion, ezért könnyebb, mint a korábban használt NiMh akkumulátor.

Az egyes elemek energiaellátása tehát a következőképpen alakul:

5. táblázat. *Energiaellátás*

Eszköz	Tápforrás
DC motorok	6 V 2000 mAh NiMh akkumulátor
Képfeldolgozó egység	5 V 5200 mAh Powerbank
Írányszámító egység	Raspberry PI USB csatlakozója

5.4. Kommunikáció

Egy mobil robot esetén a külvilággal való kommunikáció igen fontos, ezen múlik, hogy hogyan lehet irányítani, milyen gyorsan lehet vezérelni. Így e fejezet erről a komponensről fog szólni.

Kezdetben a robot egy Bluetooth modullal volt ellátva, ennek segítségével fogadta az irányzókat. Amilyen számot kapott, olyan fokban mozdult el. Ám a későbbi dizájnban ez a protokoll már nem bizonyult elégségesnek, ennél jobbra volt szükség. A mostani modell egy USB-s Wi-Fi stick-kel végzi a szükséges kommunikációt. A robot oldaláról ez küldi a kamera képét a felhasználónak, a felhasználó pedig az utasításokat küldi vissza. Ez először is gyorsabb kommunikációt, másodszor stabilabb kapcsolatot biztosít, ráadásul az USB-s kivitelnek köszönhetően könnyedén csatlakoztatható a Raspberry PI modulhoz. A használt Wi-Fi stick egy TP-Link TL-WN725N típusú modul, főbb tulajdonságait az 5. táblázat taglalja.

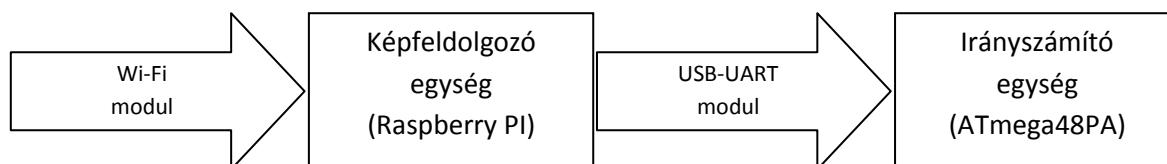
6. táblázat. A Wi-Fi stick tulajdonságai

Szabványok	IEEE 802.11b/g/n
Jel ráta	11b: <11Mbps 11g: <54Mbps 11n: <150Mbps
Frekvencia	2.400-2.4835 GHz
Átviteli teljesítmény	<20dBm
Moduláció	DBPSK, DQPSK, CCK, OFDM, 16-QAM, 64-QAM

A Wi-Fi önmagában még nem elég, hiszen az, hogy a felhasználó Wi-Fi-n keresztül elküldi a kívánt szöveget, az még nem fog eljutni a Képfeldolgozó egységből az Irányszámítóba, a kettő között is kell valamilyen kommunikációnak lennie.

Ennek megvalósítására több lehetőség is volt, mind a Raspberry PI, mind a mikrovezérlő több szabványt is ismer, többek között: U(S)ART, I²C, SPI. A választás pedig az UART-ra esett, egy USB-UART modullal könnyedén, *plug'n play* módon össze lehet kapcsolni a két eszközt és ennek révén a mikrovezérlő tápellátása is megoldódott (ahogy az előző fejezetben említve volt).

Így végezetül a robot külső és belső kommunikációjának menetét a 28. ábrán látható blokkvázlat mutatja.



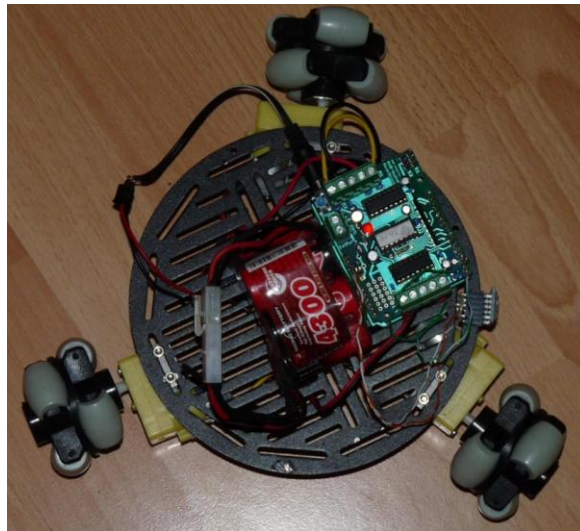
28. ábra. A kommunikáció menete.

5.5. A robot szerkezeti felépítése

Végezetül csak a robot szerkezetének a felépítése maradt, ennek két fő szakasza volt a fejlesztés során.

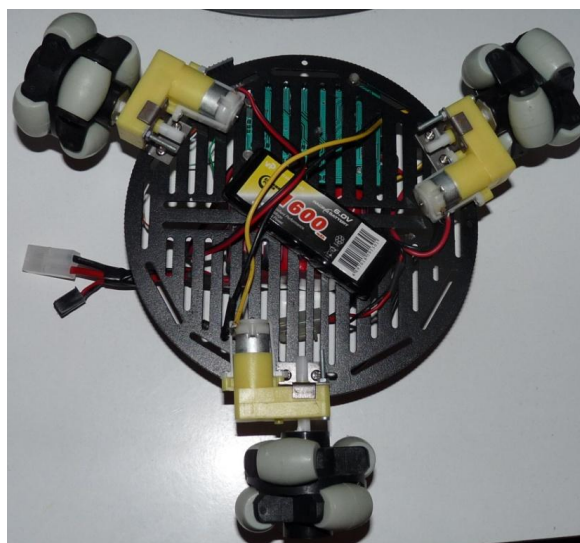
5.5.1. Az első koncepció

A robot váza egy kerek, rácsos szerkezetű fém-lemez. A rácsos szerkezet lehetővé teszi, hogy könnyedén plusz elemeket, alkatrészeket lehessen rászerezni.



29. ábra. A robot felülnézetből.

A vázon a három szénkefés DC motor egymáshoz képest 120 fokban van elhelyezve, rajtuk pedig a speciális omnikerekek.



30. ábra. A robot alulnézetből.

A könnyebb akkumulátor a váz alsó részéhez van rögzítve, a nehezebb pedig a lemez másik oldalán.

Mindkét akkumulátor a robot központi részén van elhelyezve, hiszen ezek képzik a robot tömegközéppontját. Ha nem pont középen, hanem az egyik motor közelében lennének, akkor a robot mozgás közben elfordulna, az egyenetlen súly-eloszlás miatt.

Az elektronika közvetlenül az akkumulátor mellett van elhelyezve, ennek pozíciója voltaképpen lényegtelen, tömege elhanyagolható a többi eszközhöz képest, így nem is befolyásolja a robot mozgását.

5.5.2. A végső koncepció

A fejlesztés második (és egyben végleges szakasza), az első koncepció hibáira és tapasztalataira építve lett megalkotva.

Az első koncepció nagy hibája a kerekek voltak. Ezek a kerekek tömör műanyagok, teljesen merevek, így rossz a tapadásuk is, nem csak a motor tengelyével párhuzamosan (ahogy gördülniük kellene), hanem merőlegesen is. Gyakorlatilag csak speciális talajon, mint például gumin vagy szivacson tudott a robot haladni. A szoba járólapján teljesen irányíthatatlan volt, sokszor annyira csúsztak a kerekek, hogy a robot el sem indult, egy helyben állt, miközben a kerekek forogtak.

Ezt a hibát a kerekek lecserélésével lehetett orvosolni, olyanra melyeknek a görgői nem szilárdak, hanem puhábbak, gumiból készültek.



31.ábra. A gumigörgős kerekek.

Gyakorlatilag a gumis kivitelnek köszönhetően a tengelyre merőleges irányban tapadnak a kerekek, így a motor segítségével stabilan tud haladni a robot. A görgők miatt

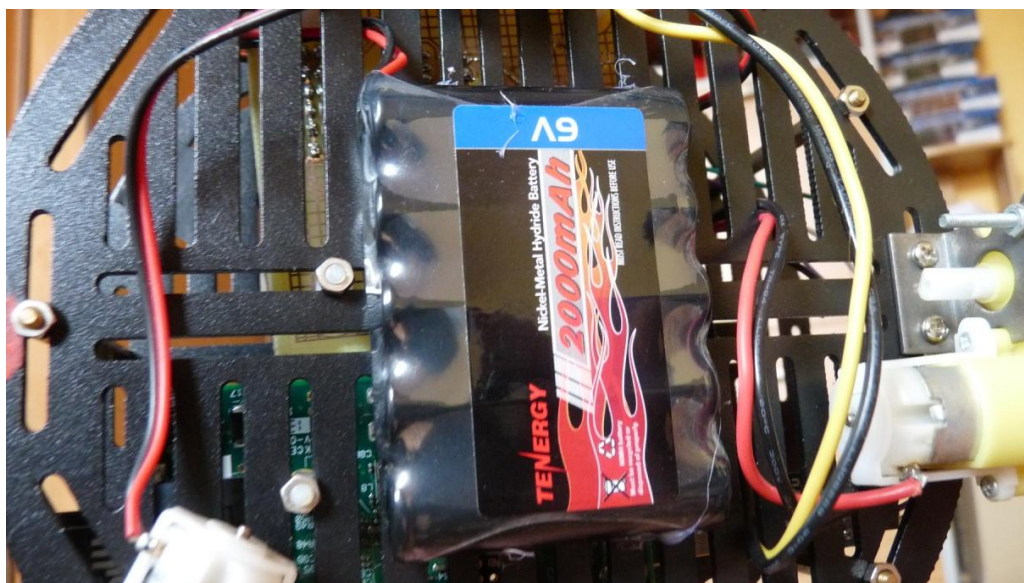
viszont még mindig el tud csúszni a robot a tengellyel párhuzamos irányban, így már – a szilárd változattal ellentétben – tényleg rendeltetésszerűen működnek az omnikerekek.

A fémrács itt is a szerkezet egyik fő alkotó elem, ennek a tetejére került mindkét áramkör, a vázon pedig a korábbihoz hasonlóan a három szénkefés DC motor egymáshoz képest 120 fokban elhelyezve, rajtuk pedig a speciális omnikerekek.



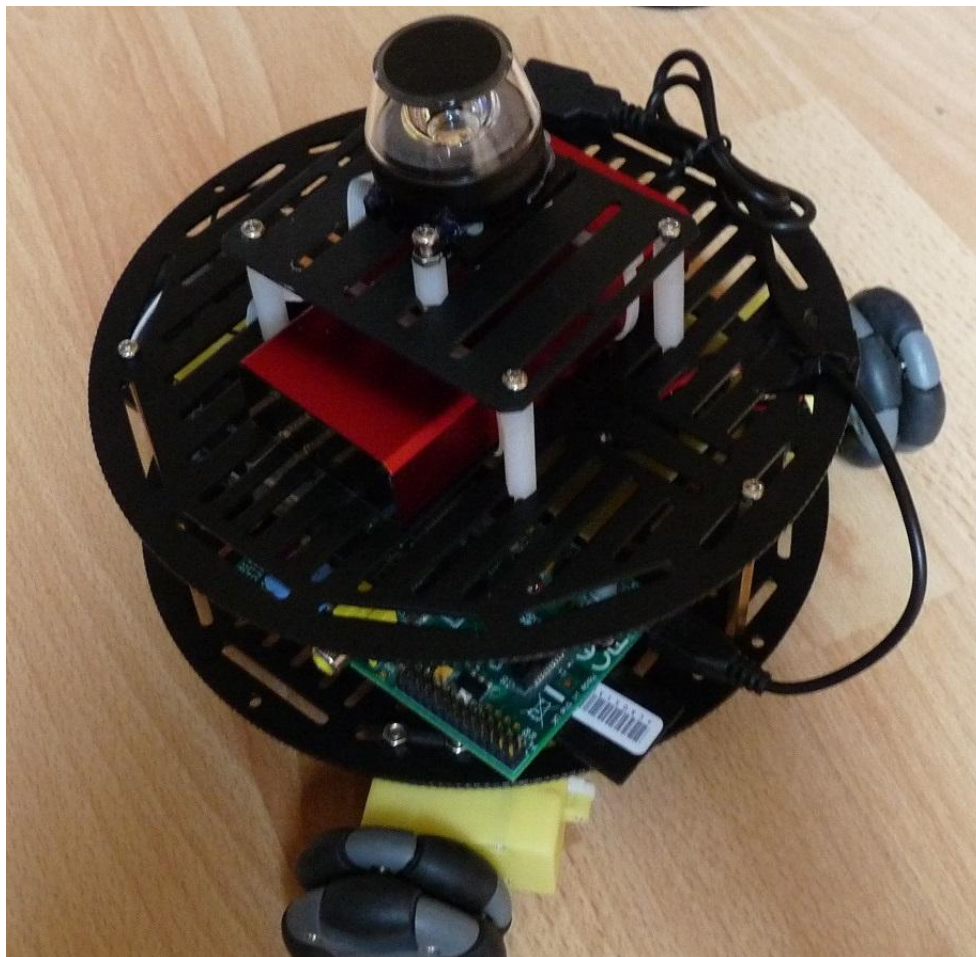
32. ábra. Az elektronika.

A fémrács aljára pedig a motorok tápfeszültségét adó NiMH akkumulátor került.



33. ábra. A robot alsó fele.

A fémlemez fölé került még egy, ugyanolyan nagyságú rács, amire került a Raspberry PI tápfeszültségét adó powerbank. Ennek tetejére került egy kisebb téglalap alapú rács, melyen a kamera és az optika található.



34. ábra. A robot.

6.A robot szoftvere

Ugyan a robot minden egyes programja megtalálható a mellékletben, érdemes a főbb elemeket részletesebben is megvizsgálni, így e fejezet a robot szoftveréről fog szólni.

6.1. Az irányszámító egység szoftvere

Mint már korábban említésre került, ennek a modulnak a feladata az egyes motorok fordulatszámának kiszámítása a kapott irányszög függvényében. Az irányszög USART modulon keresztül érkezik a mikrovezérlőbe, majd az kiszámítja a motorok PWM értékét és forgásirányát, és ezeket a motorvezérlőknek átadja, melyek vezérlik a három szénkefés DC motort. A teljes kód a 11.6. mellékletben található.

6.1.1. USART kommunikáció

Az ATmega48PA család USART moduljának főbb tulajdonságai: [11]

- Full Duplex üzemmód
- Aszinkron és szinkron mód
- Nagy felbontású baud rate generátor
- 5, 6, 7, 8 vagy 9 adat bit és 1 vagy 2 stop bit
- Paritás bit
- Zajszűrés
- Három különálló megszakítás (TX complete, TX data Register Empty, RX complete)
- Multi-processzor kommunikációs mód

Az első lépés eldönteni, hogy szinkron vagy aszinkron üzemmódban működjön-e. Mivel az USB-UART modulból hiányzik az órajel, ezért értelemszerűen csakis aszinkron mód lehetséges.

Másodsorban a kommunikáció sebességéről kell dönteni. A soros kommunikációnál (mikrovezérlős alkalmazások esetén) a legelterjedtebb baud rate érték a 9600 bps, melytől eltérő érték megadása ebben az esetben sem szükséges. A mikrovezérlő belső órajeléből (8 MHz) és a baud rate értékből (9600 bps) kiszámítható az USART Baud Rate Register (UBRRn) értéke a következő képlettel [11]:

$$UBRRn = \frac{f_{osc}}{16BAUD} - 1 \quad (8)$$

ahol:

- UBRRn: USART Baud Rate Register
- f_{osc} : A mikrovezérlő órajele
- BAUD: Baud Rate

Behelyettesítve:

$$51.0833 = \frac{800000}{16 \cdot 9600} - 1 \quad (9)$$

Mivel törtszám nem adható meg a regiszter értékének, ezért a mindenkor kerekítési szabályoknak megfelelően kell kerekíteni, ezért a regiszter értéke: 51.

Az Atmel ajánlása [11] szerint valódi és a kerekített érték között maximálisan 0.5%-os eltérés lehetséges, máskülönben a kommunikáció nem fog létrejönni. Ezt a hibarátát a 10-es képlettel lehet kiszámolni [11].

$$Hiba[\%] = \left(\frac{Kerekített\ érték}{Valós\ érték} - 1 \right) * 100 \quad (10)$$

Behelyettesítve:

$$-0.163 = \left(\frac{51}{51.0833} - 1 \right) * 100 \quad (11)$$

Mivel ez bőven benne van a félszázalékos hibahatárban, ezért a 9600 bps sebességet, a 8 MHz-es rendszerórajel mellett lehet használni.

A következő lépés kiválasztani az adat bit hosszát, a stop bit hosszát és a paritásbitet. Ennek segítségével az USART Control and Status Register **n C** (UCSRnC) szolgál. A beállítási lehetőségeket a következő három táblázat foglalja magában:

7. táblázat. Mód beállítása [11]

UMSELn1	UMSELn0	Üzem mód
0	0	Aszinkron üzemmód
0	1	Szinkron üzemmód
1	0	fenntartott
1	1	Master SPI

8. táblázat. Stop bit [11]

USBSn	Stop bit(ek)
0	1 bit
1	2 bit

9. táblázat. Adat bit [11]

UCSZn2	UCSZn1	UCSZn0	Karakter méter
0	0	0	5 bit
0	0	1	6 bit
0	1	0	7 bit
0	1	1	8 bit
1	0	0	fenntartott
1	0	1	fenntartott
1	1	0	fenntartott
1	1	1	9 bit

A korábban bemutatottak szerint az USB-UART modul nem tartalmaz szinkron csatlakozót, így csak az aszinkron üzemmód használható. Ezen felül – ahogy a sebesség beállításánál is – a többi beállításnál is elegendő az úgynevezett „klasszikus” USART beállítási lehetőségek. Tehát egy stop és nyolc adat bit, valamint nincs paritás bit. Így a következő biteket kell egybe írni: UCSZn1, UCSZn0.

A kommunikáció inicializálásának utolsó lépése annak az engedélyezése, mely az UCSRnB regiszter Receiver Enable (RXENn) bitjének aktiválásával tehető meg.

Az USART modul beállításának a kódja:

```
#define F_osc 8000000
#define baud 9600
#define ubrr F_osc/16/baud-1

void usart_init(int myubrr) { //usart inicializalasa
    UBRRL = (char) myubrr;
    UBRRH = (char) (myubrr >> 8);
    UCSRB = (1 << RXEN0);
    UCSRC = (1 << UCSZ01) | (1 << UCSZ00);
}
```

Az adatok fogadására az **USART Control and Status Register n A** (UCSRnA) regiszter USART Receive Complete (RXCn) bitje szolgál. A bit egyes értékű, ha olvasatlan adat van a fogadó buffer-ben és nulla értékű, ha a buffer üres. Lényegében addig kell várni a programnak amíg ez a bit logikai magas értékre nem vált, addig pedig folyamatosan vizsgálnia kell, hogy milyen állapotú. [11]

Ha adat érkezett (tehát egyes értékű a bit), akkor az **USART I/O Data Register n** (UDRn) 8 bitjének a kiolvasásával kapható meg. [11]

Ez kód formájában a következő:

```
char usart_get(void) { //usart - karakter fogadas
    while(!(UCSR0A & (1 << RXC0)));
    return UDR0;
}
```

Ez önmagában még nem elég az írányzó fogadására, hiszen 8 biten csak 2^8-1 , azaz 255 számot lehet ábrázolni, az írányzóhoz pedig minimum 360 szám kell. Ez a probléma úgy lett kiküszöbölve, hogy karakterenként történik az adat fogadása, a további feldolgozás pedig csak akkor történik meg, ha az írányzó minden eleme fogadásra került. Például 120 fok esetén először egy ASCII egyes karakter kerül küldésre/fogadásra, majd egy kettes, végezetül a nulla.

A fogadott adat eltárolódik egy változóban, ha új adat érkezik, akkor odébb shift-elődik egy másik változóba, majd egy következőbe, és így tovább, egészen addig, amíg egy *Kocsi Vissza* (Carriage Return, CR) bit nem érkezik. Ekkor nagyságrend szerint összeadódnak az értékek, majd ez lesz a végleges forma. [12]

Maradva a 120-as példánál: megérkezik az egyes, ez bekerül az egyesek helyére, jön a kettes, ekkor az egyes tovább kerül a tízesek helyére, a kettes pedig az egyesek helyére. Érkezik a nulla, ekkor az egyes a százask, a kettes a tízesek, a nulla pedig az egyesek helyére kerül. Végezetül pedig érkezik egy CR karakter, ekkor összegződnének az elemek, majd összeadódnak nagyságrend szerint: $1*100+2*10+0=120$.

A számok fogadásának kódja:

```
int usart_get_num(void) { //usart - szam fogadas
    char hundreds = '0';
    char tens = '0';
    char ones = '0';
    char indata = '0';
    do {
        hundreds = tens;
        tens = ones;
        ones = indata;
        indata = usart_get();
    }
    while(indata != '\r');
    return (100*(hundreds-'0')+10*(tens-'0')+(ones-'0'));
}
```

Ezzel a módszerrel már ezer (0-999) szám fogadható, így a 360 fölött még egyéb vezérlő utasítások is küldhetők.

6.1.2. PWM generálása

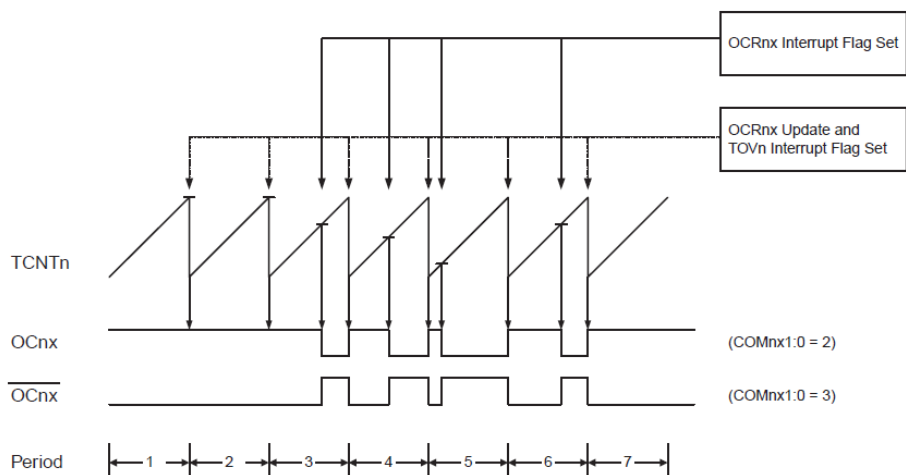
Az impulzusszélesség moduláció (**P**ulse **W**idth **M**odulation – PWM) segítségével kapják meg a motorok a megfelelő jelszintet, ennek beállítására számos opció létezik.

Az ATmega48PA mikrovezérlőben három különálló időzítő található, melyeknek 2-2 kimeneteik vannak, így összesen hat darab PWM csatorna áll rendelkezésre. Ebből három darab kell a motorok vezérlésére.

A három időzítőből kettő 8 bites, egy pedig 16 bites, így értelemszerűen a PWM felbontása 8 bites lett. A kimenetek kiválasztásánál pedig azoknak a fizikai helye volt domináns, ahogy az már korábban az áramkör tervezésénél említésre került. Ebből kifolyólag a Timer0 két kimenet, valamint a Timer2 egyik kimenete került felhasználásra.

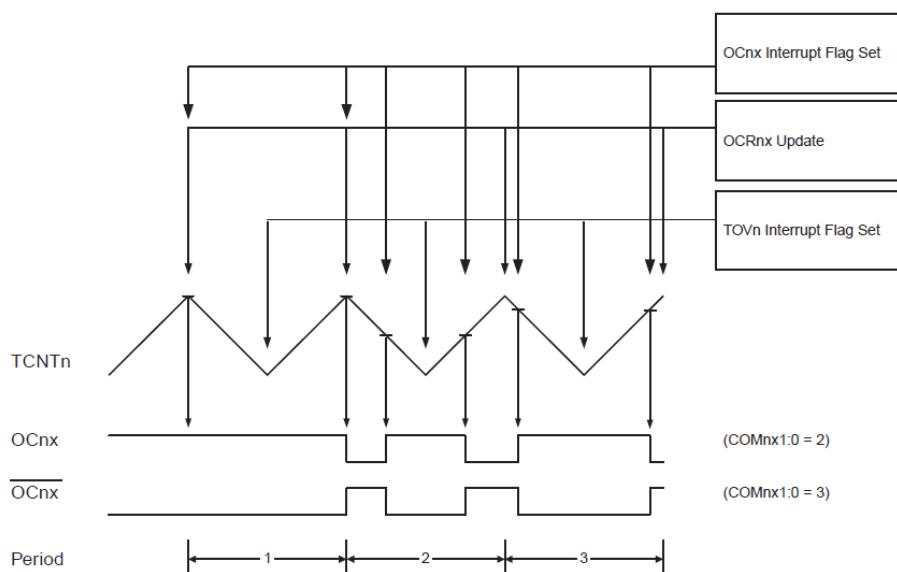
A PWM működési módjának két opciója van: az egyik az úgynevezett Fast (gyors) PWM Mode, a másik pedig az úgynevezett Phase Correct (fázis korrekt) PWM Mode. [12]

A gyors PWM nagyfrekvenciás jel előállítására szolgál, neve is innen ered. Elsősorban teljesítmény szabályozásra, DAC alkalmazásokra, vagy más, nagyfrekvenciát igénylő megvalósításra használják. Működését tekintve egy számláló (ebben az esetben egy 8 bites) nulláról addig növekszik, amíg el nem éri a maximális 255-ös értéket. Ekkor a következő órajel ciklusnál a számláló értéke törlődik, majd a megfelelő túlcsoordulást jelző regiszterek aktiválódnak, s az egész kezdődik előről. Ennek működését a 35. ábra mutatja.



35. ábra. A Fast PWM idődiagramja (forrás: [13]).

Ezzel szemben a Phase Correct PWM, ahogy a neve is mutatja, fázis korrekt jelalakot generál. A számláló az alsó, nulla értéktől az órajel hatására felfelé számol, ám a Fast PWM-mel ellentétben amikor eléri a maximális értéket, a számláló nem törlődik, hanem lefelé számol, 255-től 0-ig. Ennek az idődiagramja az 36. ábrán. látható.



36. ábra. A Phase Correct PWM idődiagramja. (forrás: [13]).

A két ábrából könnyen észrevehető, hogy előbbi esetén pontosan kétszer akkora frekvencia nyerhető ki, míg a másik esetén pontosabb, szimmetrikusabb a jelalak. Az Atmel ajánlása szerint ez utóbbi üzemmód alkalmasabb a motorvezérlő alkalmazásokhoz. [12]

A kimenet beállítására is több lehetőség van. Ha a számláló eléri a kívánt értéket, akkor a GPIO logika magasra vált, ez az invertáló mód. Ha alából magas állapotban, majd egyezés

esetén vált nullára, ez a nem-invertáló. Illetve van még egy mód, mely felváltva vált a magas és alacsony szintek között (Toggle).

Ezeknek a regiszter (TCCRnA és TCCRnB) szintű beállításait a 9-11 táblázatok mutatják.

10. táblázat. Kimenet módja

COMnA1 COMnB1	COMnA0 COMnB0	Leírás
0	0	Normál működés, a Timer kimenet inaktív
0	1	Toggle
1	0	Nem-invertáló
1	1	Invertáló

11. táblázat. Jelalak

WGMn2	WGMn1	WGMn0	Mód	Max. érték
0	0	0	Normál	0xFF
0	0	1	Fázis korrekt PWM	0xFF
0	1	0	CTC	OCRA/OCRB
0	1	1	Gyors PWM	0xFF
1	0	0	Fenntartott	-
1	0	1	Fázis korrekt PWM	OCRA/OCRB
1	1	0	Fenntartott	-
1	1	1	Gyors PWM	OCRA/OCRB

12. táblázat. Órajel

CSn2	CSn1	CSn0	Leírás
0	0	0	Nincs órajel
0	0	1	Nincs előosztás
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	Külső órajel, lefutó élre vált
1	1	1	Külső órajel, felfutó élre vált

A nem-invertáló mód használata a célravezető. Az előzőekben vázoltak szerint a fázis korrekt PWM-et kifejezetten ajánlja a gyártó, a maximális értéket nem szükséges bolygatni. Az előosztás pedig kell, hiszen a nagyfrekvenciás jelet a motorvezérlő nem tudja kezelni. Ennek értéke igazából mindegy (csak ne legyen túl kicsi), ezért a 256-ra esett a választás.

Összegezve: TCCR0A és TCCR2A regiszterekből a COM0A1/COM2A1, COM0B1/COM2B1 és WGM00/WGM20 biteket kell aktiválni. A TCCR0B és TCCR2B regiszterekből pedig a CS02/CS22 biteket kell aktiválni.

A fent leírtak kód formájában:

```
void pwm_init(void) { //pwm inicializalasa (timer 1 & 2)
    TCCR0A = (1 << COM0A1) | (1 << COM0B1) | (1 <<
WGM00);
    TCCR0B = (1 << CS02);

    TCCR2A = (1 << COM2B1) | (1 << WGM20);
    TCCR2B = (1 << CS22);
}
```

A PWM inicializálásán túl érdemes még egy olyan függvényt létrehozni, mely a PWM érték és a port alapján beállítja a timer értékét, stílusosabb, és egyszerűbb is. Ez a következőképpen néz ki:

```
void pwm_set(char channel, char val) {
    if (val == 0) {
        if (channel == mot1en) {
            PORTB &= ~(1 << mot13);
            PORTD &= ~(1 << mot14);
        }
        else if (channel == mot2en) {
            PORTB &= ~(1 << mot21);
            PORTB &= ~(1 << mot22);
        }
        else if (channel == mot3en) {
            PORTD &= ~(1 << mot31);
            PORTD &= ~(1 << mot32);
        }
    }
    if (channel == mot1en)
        OCR0A = val;
    else if (channel == mot2en)
        OCR0B = val;
    else if (channel == mot3en)
        OCR2B = val;
}
```

Érdemes megemlíteni, hogy amikor a számláló 0 értéket kap, az nem eredményezi, hogy a kimenet is logikai 0 állapotban lesz, hiszen amikor túlcsordul a számláló, akkor egy órajel ciklusig logikai magas állapotú lesz a kimenet. Ez ugyan nagyon rövid ideig tart előosztás függően az órajel 256-od része, de ezzel akkor is számolni kell. Tehát, ha nulla

értéke van a számlálónak, akkor a motorvezérlő forgásirányváltó lábai közül mindkettő 0 értéket kap, így függetlenül attól, hogy az engedélyező bementre milyen jelet küld a mikrovezérlő, a motor mindenképpen állni fog, ahogy azt korábban az 1. táblázatban is lehetett látni.

6.1.3. A főprogram

Az USART és a PWM beállítása után már csak a főprogramrész maradt hátra.

A szükséges inicializálások és beállítások után a program addig vár, amíg USART-on keresztül meg nem kapja az irányszöveget. Ekkor megvizsgálja annak értékét.

Ha a szám értéke 666, akkor mind a három motor álló helyzetbe kerül, ez a *stop* utasítás, majd vár a következő utasításra.

Ha ettől különbözött, akkor a kapott fokot átváltja radiánba, majd a korábban említett (1) képlet szerint kiszámolja az egyes motorok PWM kitöltési tényezőjét. Annak függvényében, hogy a szám pozitív vagy negatív kerül a forgásirány megállapításra, egyik esetben az óramutató járásával megegyező, míg a másik esetben azzal ellentétes irányba forognak. Ha a kapott irányszög negatív, akkor a PWM érték kap egy -1-es szorzót, hiszen az eredeti negatív szám nem alkalmas a motorok meghajtására, mondhatni negatív impulzusokat nem tud kiadni a mikrovezérlő.

A forgásirány és a negatív előjelek elrendezése után már csak a PWM érték tényleges beállítása van hátra, s a robot már el is indult a kívánt szögbe. Mindezek után pedig az egész kezdődik előről.

Összefoglalva az előbbieket kód formájában:

```
deg = usart_get_num();
if (deg == 666) {
    pwm1 = 0;
    pwm2 = 0;
    pwm3 = 0;
} else {
    rad = deg * 0.0174532925;
    pwm1 = (sin(rad)*200);
    pwm2 = (sin(rad + 2.0943951)*200);
    pwm3 = (sin(rad + 4.1887902)*200);

    if (pwm1 >= 0) {
        PORTB |= (1 << mot13);
        PORTD &= ~(1 << mot14);
    } else {
        PORTB &= ~(1 << mot13);
        PORTD |= (1 << mot14);
        pwm1 *= -1;
    }

    if (pwm2 >= 0) {
        PORTB |= (1 << mot21);
        PORTB &= ~(1 << mot22);
    } else {
        PORTB &= ~(1 << mot21);
        PORTB |= (1 << mot22);
        pwm2 *= -1;
    }

    if (pwm3 >= 0) {
        PORTD |= (1 << mot31);
        PORTD &= ~(1 << mot32);
    } else {
        PORTD &= ~(1 << mot31);
        PORTD |= (1 << mot32);
        pwm3 *= -1;
    }

    pwm_set(mot1en, pwm1);
    pwm_set(mot2en, pwm2);
    pwm_set(mot3en, pwm3);
}
```

6.2. A képfeldolgozó egység szoftvere

A korábban ismertettek szerint ennek a modulnak a feladata a kamera képének a fogadása, majd annak továbbítása a hálózatra, az irányszámító egységgel való kommunikáció, valamint a külvilágtól, az embertől kapott utasítások kezelése. A teljes kód a 11.7. mellékletben található.

6.2.1. A kamera képének közvetítése

Ahhoz, hogy a robotot irányító személy láthassa a robot által észlelt környezetet, valamilyen szoftver segítségével stream-elni kell a kamera képét.

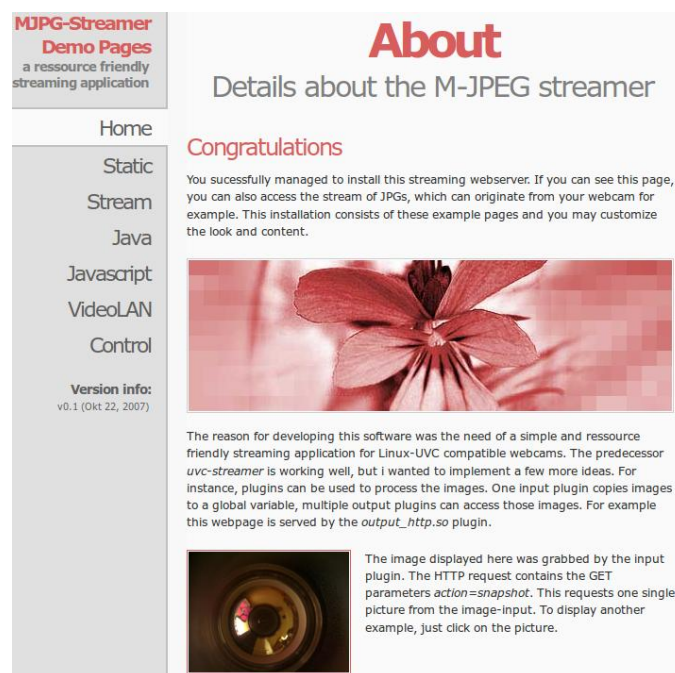
Ám még ennek a folyamatnak az elkezdése előtt egy probléma akadt a kamerával. Egy erős fényű állapotjelző LED található a kamera panelján, melynek fénye az optika műanyag burkolatáról visszaverődik, és ezáltal bezavar a képbe, a fénytől nem látni semmit.

Ehhez a Raspberry PI konfigurációs fájlját kell módosítani, mely lényegében a Raspberry PI BOIS megfelelője. Ez fájl a /boot/config.txt helyen található, s a következő utasítást kell beleírni:

```
disable_camera_led=1
```

Ezzel máris kikapcsolt állapotban marad a LED, visszakapcsolni a parancs törlésével, kikommentelésével vagy 0-ra módosításával lehet.

A kamera képének megtekintésére az MJPG-Streamer nevű alkalmazás került felhasználásra. A program automatikusan generál egy *html* honlapot, ahol azonnal megtekinthető a kamera képe, különböző környezetben.



37. ábra. Az MJPG-Streamer kezelőfelülete.

Lehetőség van statikus pillanatkép megtekintésére JPG formátumban, amit lehet folyamatosan frissíteni javascript-tel, így egy folytonos kép keletkezik. Létezik Java verzió is, valamint mjpeg stream-en keresztül is megtekinthető a kamera képe, melyet közvetlenül a

böngészőből, vagy valamilyen filmlejátszó szoftverrel lehet megtekinteni. Lehetőség van továbbá a html kezelőfelületen belül módosítani a kép tulajdonságait, tükrözni, forgatni, méret beállítását módosítani, max. fps, stb.



38. ábra. A kamera (plusz az optika) képe

640x480-as felbontás esetén átlagosan 1.5-1.7 képkocka/sec a stream képfrissítése, mely ugyan nem valami sok, de a robot kezeléséhez még elégséges.

6.2.2. Utasítások fogadása, kommunikáció a külvilággal

A kamera képének közvetítésével a robot irányítása megoldottá vált, a felhasználó láthatja a környezetet. Ám ahhoz, hogy utasításokat is tudjon adni, ahhoz kell még egy programot futtatni.

A program Python 2.7 script nyelven lett megírva. Mivel ez egy általános célú, magasszintű programozási nyelv, ennek elsajátítása, valamint kezelése roppant gyorsan és egyszerűen megtehető. Ráadásul a hozzáírt plusz modulokkal tovább egyszerűsödik a feladat, nagyon sok plusz dolgot lehet segítségükkel megvalósítani.

A robot vezérlésénél három ilyen modul került felhasználásra. A *Pyro4*, azaz a **Python Remote Object 4** nevű kiegészítő, mely segítségével egyszerűen és felhasználóbarát módon lehet Python objektumokat továbbítani a hálózaton. Segítségével két különböző számítógép tud egymás között Python elemeket megosztani.

A másik a *serial* nevű modul, mely segítségével soros kommunikációt lehet létesíteni más eszközökkel, ebben az esetben az Irányszámító egység számára lehet továbbítani a célirányt.

A harmadik a *time*, mely segítségével különböző időzítéssel kapcsolatos feladatokat lehet megoldani.

Ahhoz, hogy a két rendszer tudjon kommunikálni, el kell dönteni, melyik legyen a szerver és melyik a kliens: lényegében, hogy melyik rendszer kapcsolódik a másikhoz. Ebben az esetben a robot a szerver, az irányító számítógépe pedig a kliens, így a számítógép kapcsolódik a robothoz. A szerver létrehozásához le kell futtatni egy Python script-et (a parancssorban), ami létrehoz egy úgynevezett *nameserver*-t, mely a szervergép azonosítására szolgál. Paraméterként a szerver (helyi) IP címét kell megadni.. Ezt a következő utasítással lehet megoldani:

```
python -m Pyro4.naming -n 192.168.1.109
```

A kapcsolathoz meg kell adni a szerver host-ját, annak a nevét és a megosztani kívánt objektum nevét.

```
server = in_data()

daemon=Pyro4.Daemon(host="192.168.1.106")
ns=Pyro4.locateNS()
uri=daemon.register(server)
ns.register("omnibotserver", uri)
```

A mikrovezérlővel való kommunikáció esetén csupán meg kell adni az USB portot, ahova a mikrovezérlő kapcsolódik, illetve a kommunikáció sebességét, ami 9600 bps.

```
uart = serial.Serial("/dev/ttyUSB0", 9600, timeout=0)
```

A kommunikáció során egy kételemű listát kap a robot a hálózaton keresztül. A lista egyik, azaz a nulladik eleme tartalmazza a kívánt irányszöveget, melyet azonnal továbbít is USB-UART-on keresztül az Irányszámító egységnek. A másik elem pedig az idő (másodpercben), amíg a mozgást végrehajtja. Miután elküldte az irányszöveget a következő alegységnek, a program várakozik (ameddig a felhasználó utasította), majd elküldi a stop

utasítást, s újra várakozik a felhasználó következő utasítására. Ez kód formájában a következő:

```
class in_data(object):
    def get_data(self, deg_wait):
        uart.write("%d\r" % deg_wait[0])
        time.sleep(deg_wait[1])
        uart.write("666\r")
```

Természetesen ez önmagában nem elég, hiszen így minden egyes alkalommal kézzel kellene elindítani a programokat, ami egy mobil robot esetén nem túl praktikus. Ezért be kell állítani, hogy rendszerinduláskor a programok is induljanak el, ezt az /etc/rc.local fájl szerkesztésével lehet elérni. Ez a fájl minden egyes betöltéskor lefut. Ennek tartalma:

```
#nameserver
(sleep 10;python -m Pyro4.naming -n 192.168.1.109)&

#omnibot server
(sleep 10; python ~pi/python/omnibot_server.py)&

#mjpg streamer
export LD_LIBRARY_PATH=/home/pi/mjpg-streamer/mjpg-
streamer-experimental
/home/pi/mjpg-streamer/mjpg-streamer-
experimental/mjpg_streamer -i "input_raspicam.so -d 10 -x
640 -y 480" -o "output_http.so -w /home/pi/mjpg-$
&

exit 0
```

7. A távvezérlés

Végezetül a felhasználó által látott, illetve használt kezelőfelület felépítése és belső működése kerül bemutatásra.

7.1. A kezelőfelület felépítése

A program felépítése rendkívül egyszerű, de használatáról azért érdemes pár szót ejteni. A program kezelőfelületét a 39. ábrán látható.



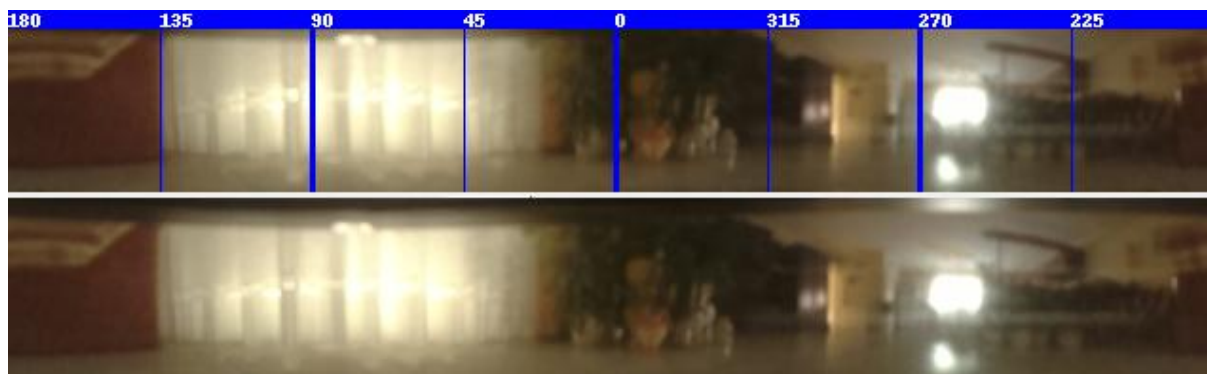
39. ábra. A program.

Az ablak nagy részét a 360 fokos panorámakép tölti ki, alatta vannak a gombok, illetve a beviteli mezők. Az első szövegdobozba írandó a robot haladásának irányszöge fokban (Angle). Ez 0 és 359 fok között bármilyen egész szám lehet. Bár az előlotti számok megadása is lehetséges, de például a 0, a 360 vagy 720 fokok között nem tesz különbséget a robot, mindhárom esetén ugyanabba az irányba megy.

Mellette pedig az az idő másodpercben (Time), mely a robot mozgásának az időtartamát jelenti. Ez a mező float típusú, tehát akár tört szám is megadható késleltetésnek, így akár tizedmásodperces apró elmozdulások is kiadhatóak a robot számára.

Ezek mellett pedig a küldés (Send) gomb, mely értelemszerűen a robotnak küldi el az utasítást.

Található még ezeken felül egy jelölőnégyzet (Scale) is, melynek aktiválása után a kamera képen megjelenik egy fok-skála, aminek a segítségével a felhasználó be tudja határolni, hogy az egyes elemek, tárgyak milyen irányban helyezkednek el a robothoz képest.



40. ábra. A skálázott és a skálázatlan kép.

Látható, hogy a skála 45 fokenként van jelölve, a középpontban a 0 fok, jobbra és balra pedig 180 fokig csökken vagy nő a skála, így könnyedén megállapítható, hogy a robot eleje a kijelző közepén található, a hátulja pedig a két szélén.

7.2. A kezelőfelület belső működése

A kezelőfelület használata után már csak a belső működésének mechanizmusa maradt hátra, a kód részletezése. A teljes script a 11.8. mellékletben található.

A program – a robot szoftveréhez hasonlóan – Python 2.7 környezetben íródott, plusz több kiegészítő modul is felhasználásra került. Többek között a **Python Remote Object 4** (*Pyro4*), hogy tudjon kommunikálni a robottal, a *Tkinter*, mely grafikus kezelőfelület (GUI) létrehozását támogatja. Az *urllib*, mely segítségével egy külső webcímet lehet elérni. Az *os*, mely az operációs rendszerrel kapcsolatos funkciók meghívását támogatja. És végezetül a *PIL*, azaz a **Python Image Library**, mely képek szerkesztését, módosítását és megjelenítését támogatja.

A *Pyro4* kliens esetén csupán meg kell adni a szerver nevét a kapcsolat létrehozásához:

```
server=Pyro4.Proxy("PYRONAME:omnibotserver")
```

A *Tkinter* kezelőfelület létrehozásához meg kell adni az egyes elemek helyét. Hol, melyik sorban, illetve oszlopban helyezkednek el. A kép a nulladik oszlop, nulladik sorában található. A feliratok, a szövegdobozok, a gomb és a jelölőnégyzet pedig az első sorban, az oszlopok pedig egymás után következnek.

Ezen felül még a szélességük, a szövegük (ha van), valamint a gomb és a jelölőnégyzet esetén az aktiválásukat követő folyamat változójának nevét kell megadni.

Mindez a következő módon néz ki:

```
pic = Tkinter.Label(root)
pic.grid(row=0, column = 0, columnspan=20)

label_deg = Tkinter.Label(root, text="Angle:")
label_deg.grid(row=1, column=0)

entry_deg = Tkinter.Entry(root, width=5)
entry_deg.grid(row=1, column=1)

label_wait = Tkinter.Label(root, text="Time:")
label_wait.grid(row=1, column=2)

entry_wait = Tkinter.Entry(root, width=5)
entry_wait.grid(row=1, column=3)

btn_send = Tkinter.Button(root, text="Send", width=5,
command=send)
btn_send.grid(row=1, column=4)

v = Tkinter.IntVar()
check = Tkinter.Checkbutton(root, text="Scale", variable=v)
check.grid(row = 1, column = 5)
```

A gomb megnyomása után a két szövegdoboz szövege átkonvertálódik a megfelelő adattípussá: az irányszög esetén integer, az idő esetén pedig float. Ezeket továbbítja a *Pyro4* a robotnak, majd a két szövegmező értéke törlődik, hogy a felhasználónak ne kelljen kézzel elvégeznie.

```
def send():
    server.get_data([int(entry_deg.get()),
float(entry_wait.get())])
    entry_deg.delete(0,3)
    entry_wait.delete(0,3)
```

A kép megjelenítéséhez először a robot által közzétett jpg kerül letöltésre (ehhez kell az *urllib*), majd az a kép az *Imagemagick* nevű képszerkesztő szoftver segítségével kerül transzformálásra. A képszerkesztő előnye, hogy konzolablakból is lehet vezérelni, s így a Python és az *os* modul segítségével lehet használni. Illetve mivel csak egy egyszerű parancsról van szó, a futási sebességben rendkívül gyors. Ezeken kívül a programozás esetében is nagyban megkönnyítette a dolgot, hiszen nem kellett manuális leprogramozni a transzformációt. Ezek után már csak annyi a teendő, hogy az elkészült képet megjeleníti a rendszer, s az egész folyamat kezdődik előlről.

A jelölőnégyzet állapotát egy feltétel figyeli, ha aktiválva van, akkor megjelennek a szögeket jelölő vonalak, valamint a számok. Ezek a *PIL* segítségével lettek elkészítve.

A fent leírtak kód formájában:

```
def video():
    urllib.urlretrieve("http://192.168.1.106:8080/?action=snapshot", "camera.jpg")
    os.system('convert camera.jpg -distort DePolar "140 62 297,276 0,360" -resize 100x20% final.jpg')
    image = Image.open("final.jpg")
    draw = ImageDraw.Draw(image)
    if v.get() == 1:
        draw.line((80,0, 80,96), fill="Blue", width=1)
        draw.line((160,0, 160,96), fill="Blue", width=2)
        draw.line((240,0, 240,96), fill="Blue", width=1)
        draw.line((320,0, 320,96), fill="Blue", width=2)
        draw.line((400,0, 400,96), fill="Blue", width=1)
        draw.line((480,0, 480,96), fill="Blue", width=2)
        draw.line((560,0, 560,96), fill="Blue", width=1)
        draw.line((0,0, 640,0), fill="Blue", width=20)
        draw.text((0,0), text="180")
        draw.text((80,0), text="135")
        draw.text((160,0), text="90")
        draw.text((240,0), text="45")
        draw.text((320,0), text="0")
        draw.text((400,0), text="315")
        draw.text((480,0), text="270")
        draw.text((560,0), text="225")
    img = ImageTk.PhotoImage(image)
    pic.configure(image = img)
    pic.image = img
    pic.after(100, video)
```

8. Jövőbeli tervek

A robot továbbfejlesztésére számtalan lehetőség van. Elsősorban a kamerarendszert kellene megújítani. A képet jobb minőségben (nagyobb felbontás, több fps) kellene továbbítani, ehhez új hardverek kipróbálása, tesztelése szükséges. A robotot ezenfelül különféle szenzorokkal is ki lehetne egészíteni (erre már a nyomtatott áramkör készítése során biztosítottam lehetőséget).

Mindezeken felül pedig érdemes lenne a robot matematikai modelljét is elkészíteni valamilyen szimulációs szoftver segítségével.

9. Összefoglalás

A téma kidolgozása során az egyes fejezetekben részletesen ismertettem a robot funkcionális egységeit, felépítését, bemutattam a speciális omnikerekek működését, a mozgás számításának a menetét, a felhasznált (és épített) áramköröket, az elektronika működését.

Gondosan kitértem a kamerarendszerre is, milyen körbelátó kamerarendszerek, optikák léteznek, s ezek közül melyik került felhasználásra.

A hardverek és szoftverek fejlesztésének részletes menetét, valamint milyen problémákba ütköztem a fejlesztés során.

Összegezve a robot egy egészen sajátos és újszerű megvalósítás lett, a különleges meghajtásának köszönhetően valóban gyorsan és dinamikusan képes mozogni. Összetett kamerarendszerének köszönhetően pedig a teljes környezetét egyszerre lehet vizsgálni.

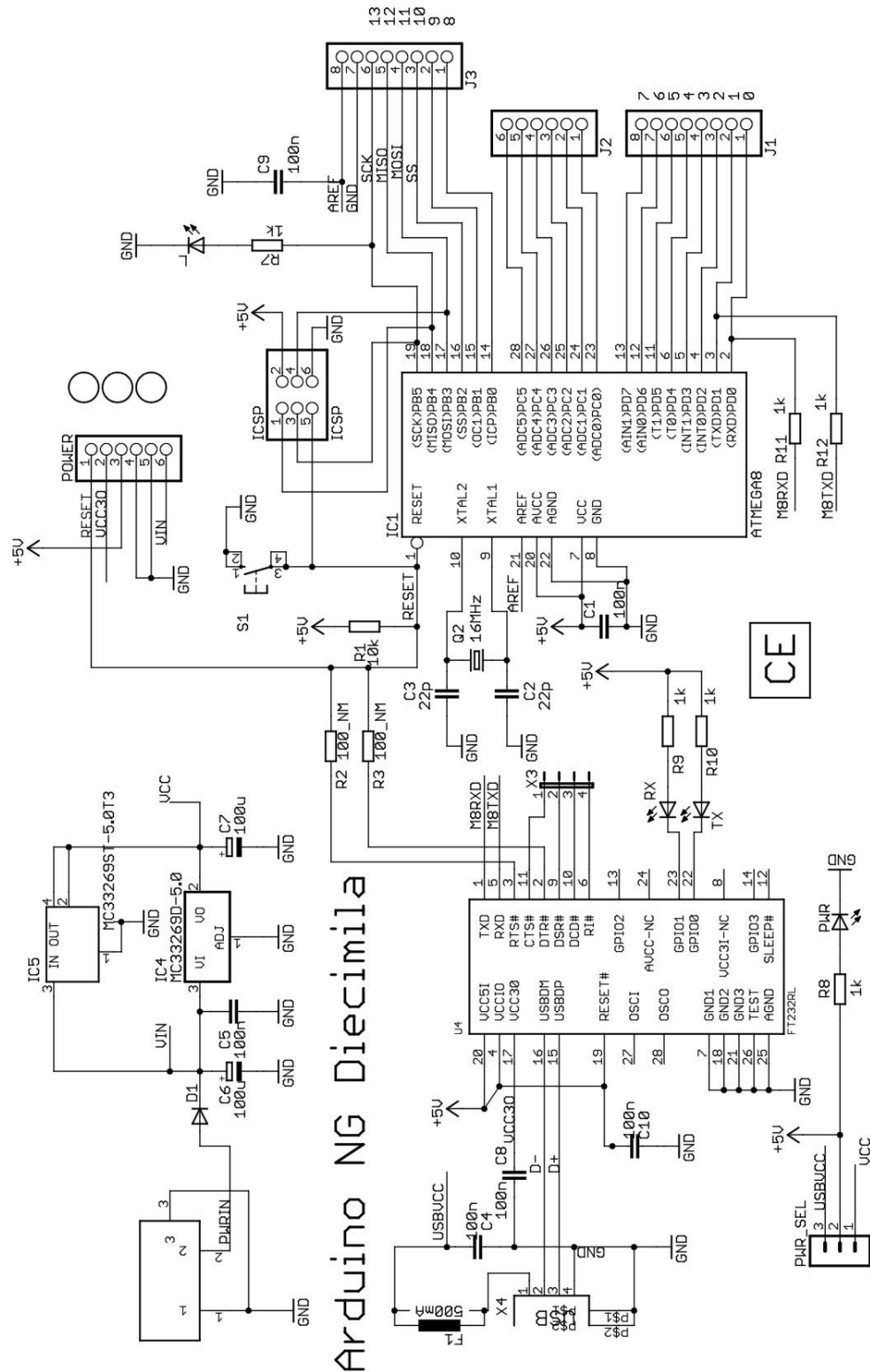
10. Irodalomjegyzék

- [1] Naveen Suresh Kuppaswamy, Se-Hyoung Cho, Daniel Stonier, Sung-Lok Choi, Jong-Hwan Kim: *DESIGN OF AN OMNIDIRECTIONAL ROBOT FOR FIRA ROBOSOT* Korea Advanced Institute of Science and Technology (2006)
- [2] Németh Ferenc: *Omnidirekcionális mobil robot távvezérlése gépi látással* TMDK dolgozat, Széchenyi István Egyetem, Győr (2013)
- [3] Ross Cutler, Yong Rui, Anoop Gupta, JJ Cadiz Ivan Tashev, Li-wei He, Alex Colburn, Zhengyou Zhang, Zicheng Liu, Steve Silverberg: *Distributed Meetings: A Meeting Capture and Broadcasting System* Microsoft Research, One Microsoft Way, Redmond, WA, USA (2002)
- [4] <http://www.fullview.com/> (utolsó látogatás: 2014.10.11.)
- [5] <http://www.inf.u-szeged.hu/~kato/teaching/IpariKepfeldolgozas/02-OmniCamera.pdf> (utolsó látogatás: 2014.10.11.)
- [6] H. Afshari, L. Jacques, L. Bagnato, A. Schmid, P. Vanderghenst et al.: *The PANOPTIC Camera: A Plenoptic Sensor with Real-Time Omnidirectional Capability* Journal of Signal Processing Systems for Signal Image and Video Technology, Vol. 70, No. 3. (2013), pp.305-328.
- [7] <http://lsm.epfl.ch/page-52820-en.html> (utolsó látogatás: 2014.10.11.)
- [8] <http://epa.oszk.hu/00000/00025/00008/pdf/greguss.pdf> (utolsó látogatás: 2014. 10.11.)
- [9] Texas Instrument: *L293, L293D QUADRUPLE HALF-H DRIVERS* adatlap (2004)
- [10] Fairchild Semiconductor: *KA78XX/KA78XXA 3-Terminal 1A Positive Voltage Regulator* adatlap (2001)
- [11] Atmel: *8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash: ATmega48PA/ATmega88PA/ATmega168PA/ATmega328P* adatlap (2009), pp. 176-203.
- [12] Elliot Williams: *Make: AVR Programming*, Maker Media (2014), pp. 77-104.

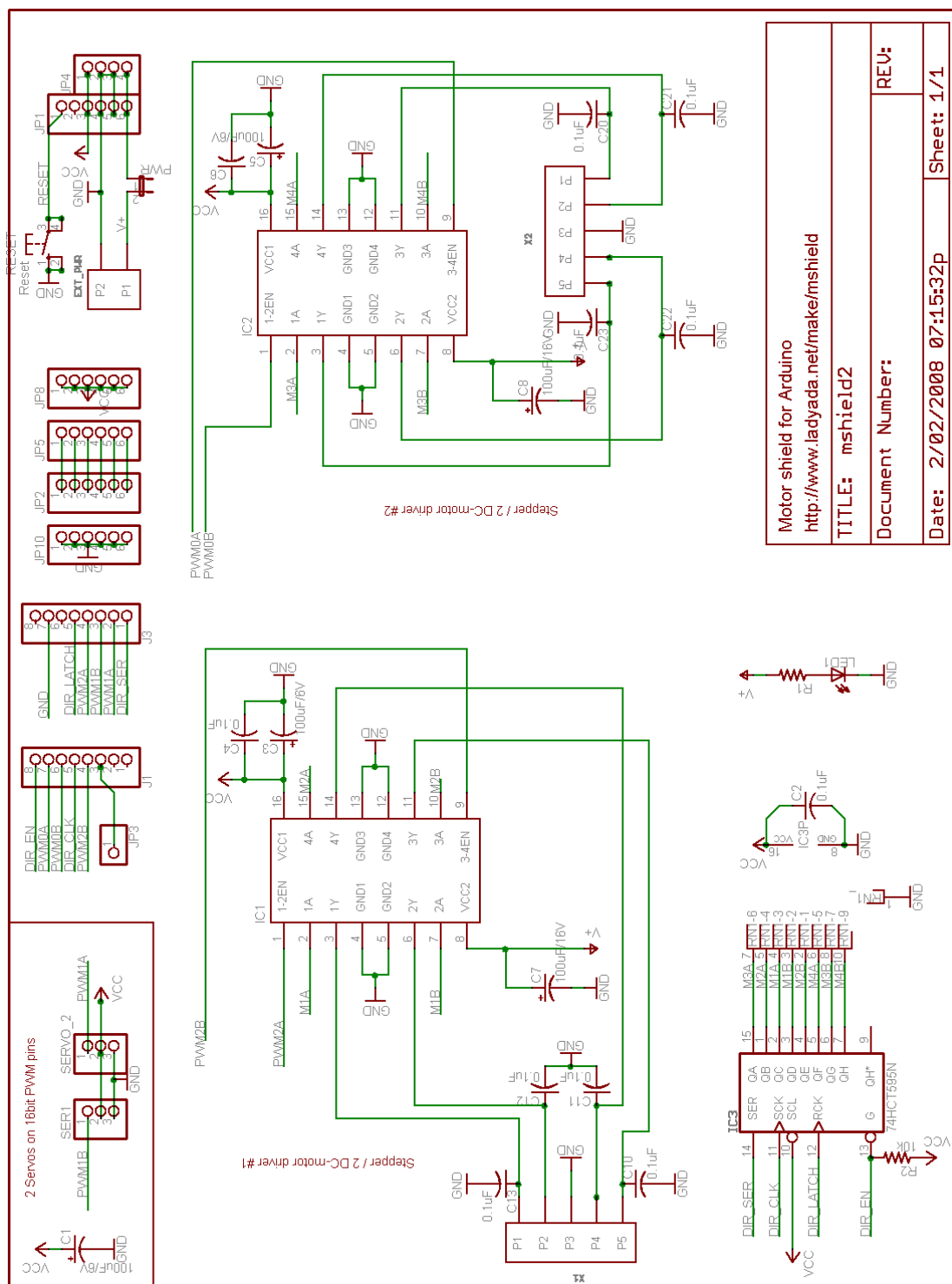
- [13] Atmel: *8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash: ATmega48PA/ATmega88PA/ATmega168PA/ATmega328P*
adatlap (2009), pp. 113-140.

11. Melléklet

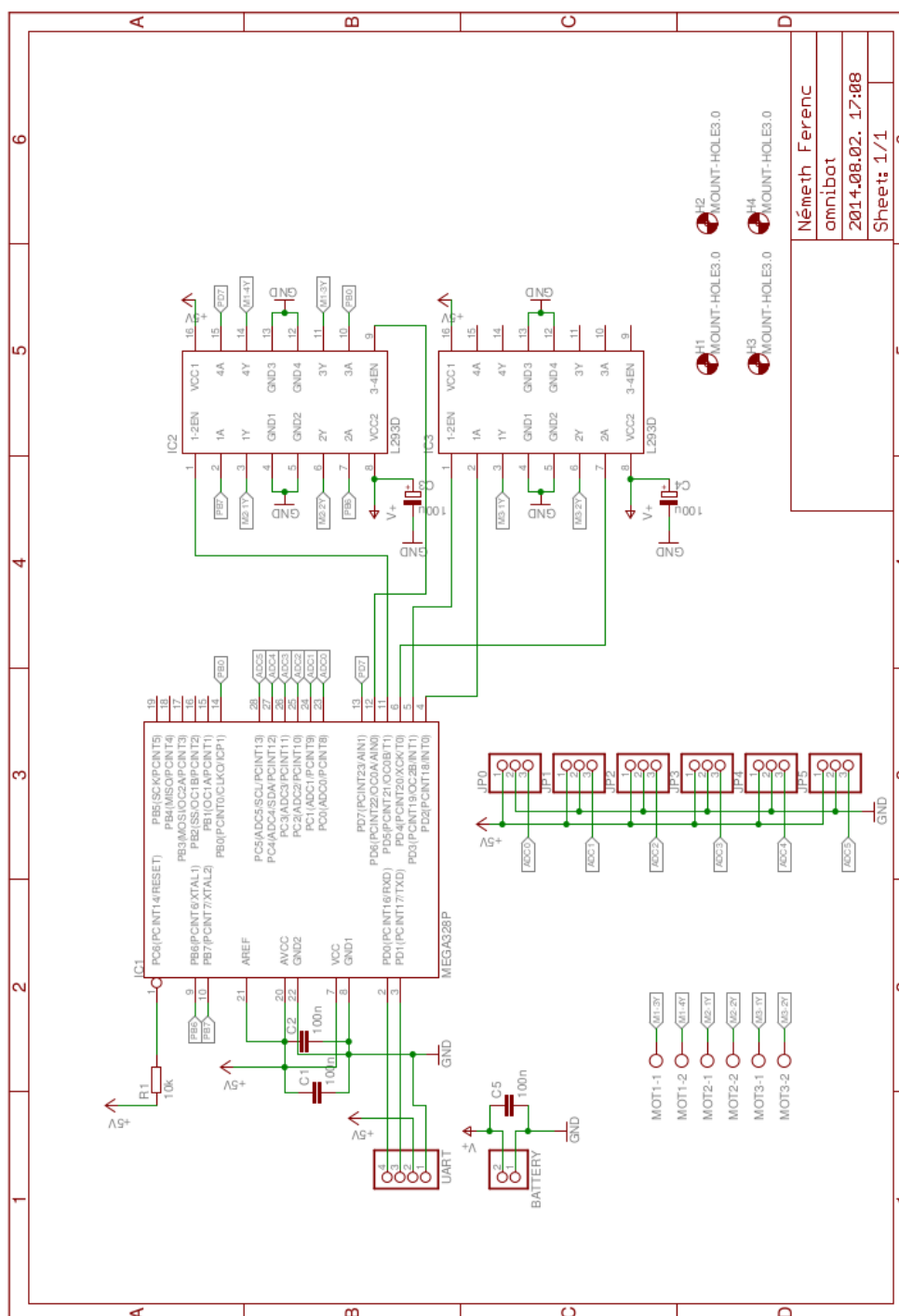
11.1. Az Arduino kapcsolási rajza



11.2. Az Adafruit motorvezérlő kapcsolási rajza

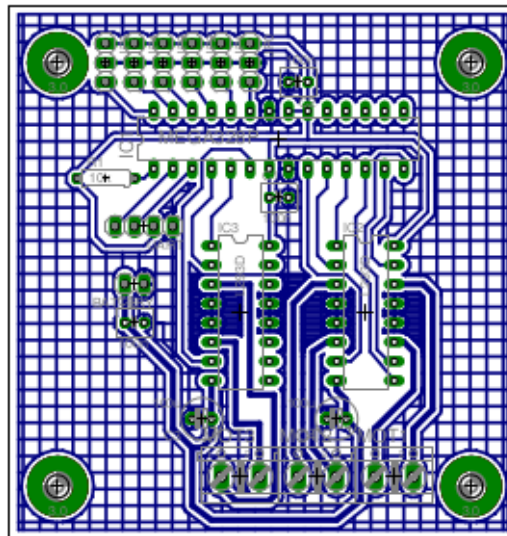


11.3. Az irányszámító egység kapcsolási rajza

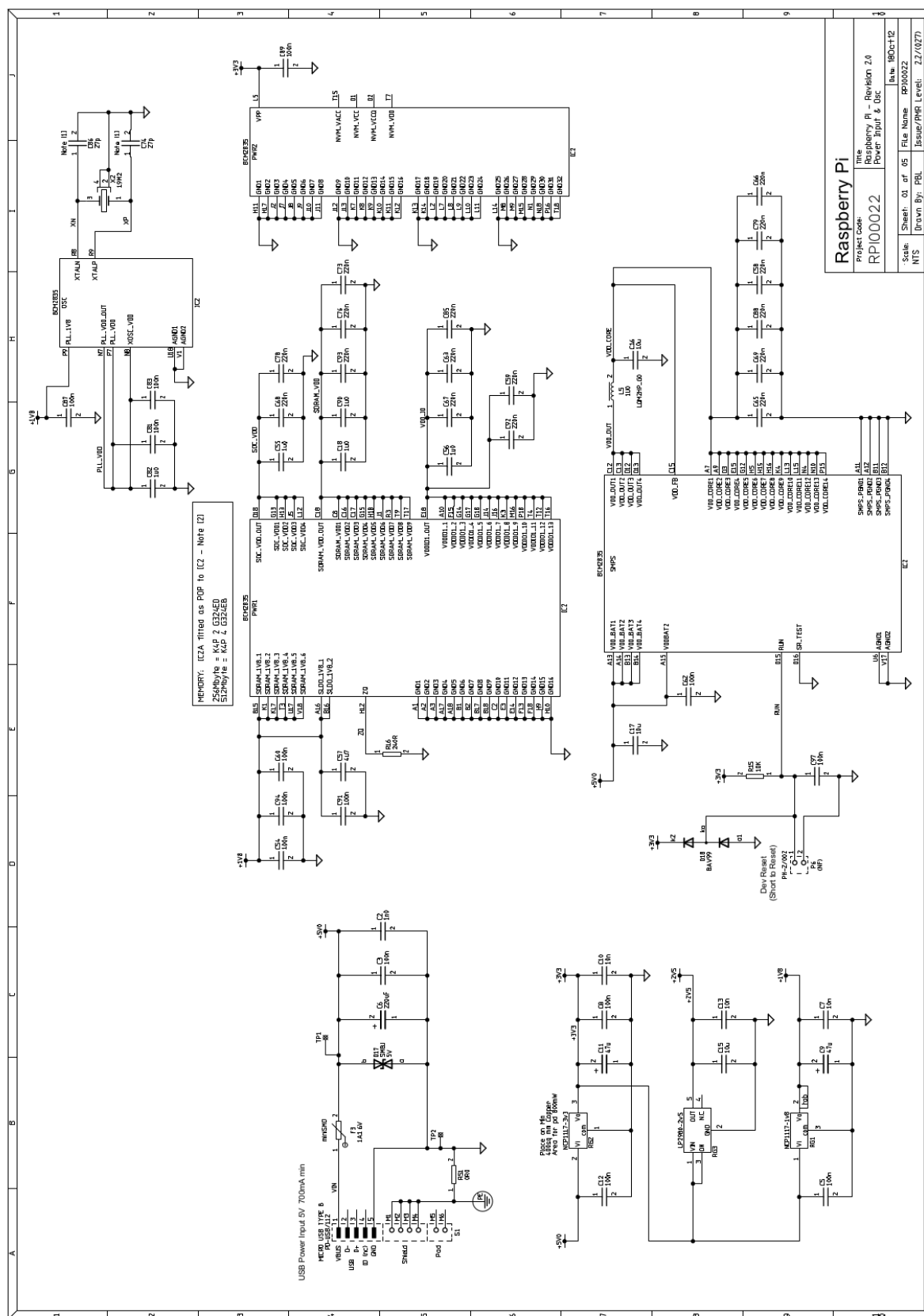


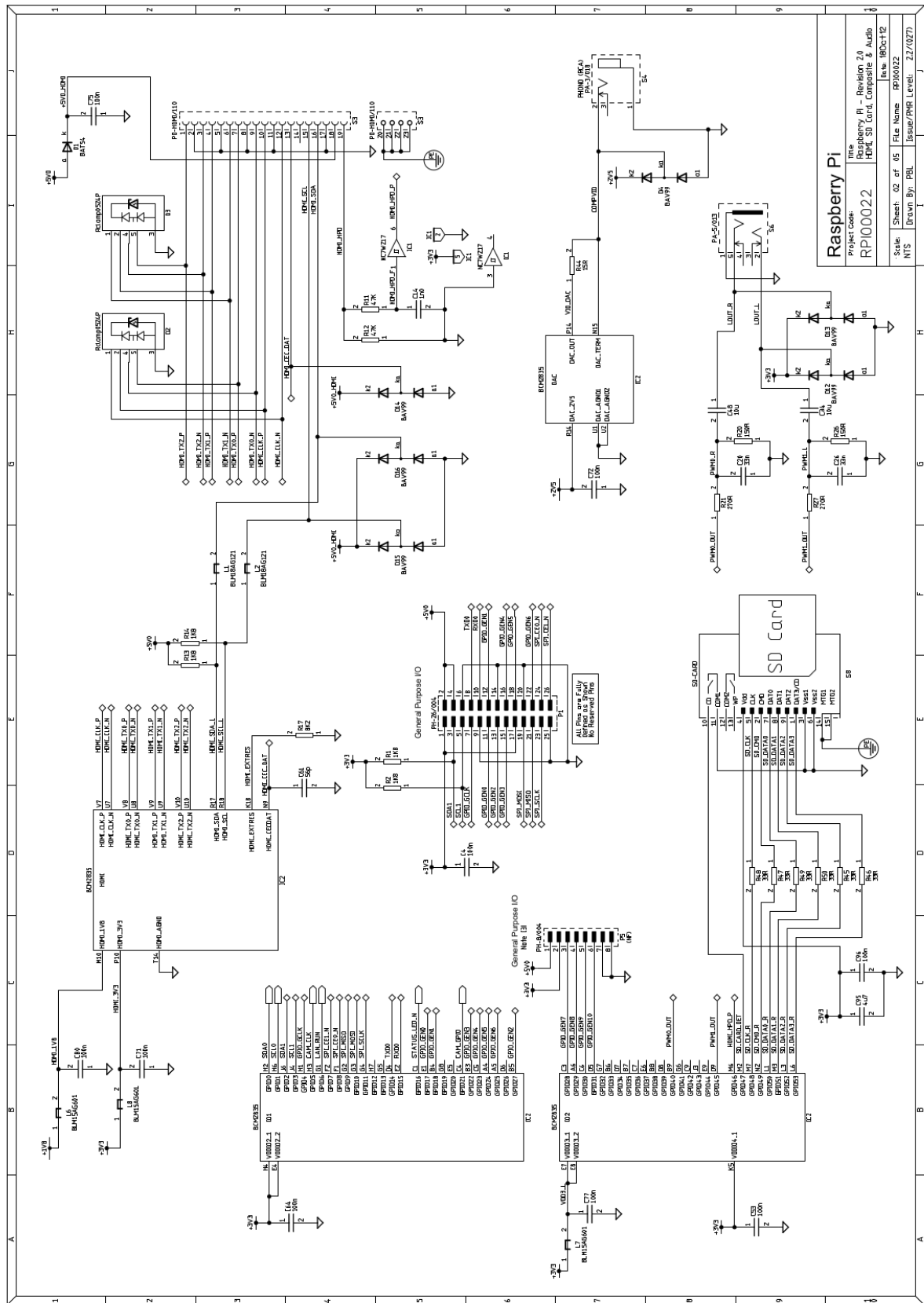
Németh Ferenc
omniBot
2014.08.02. 17:08
Sheet: 1/1

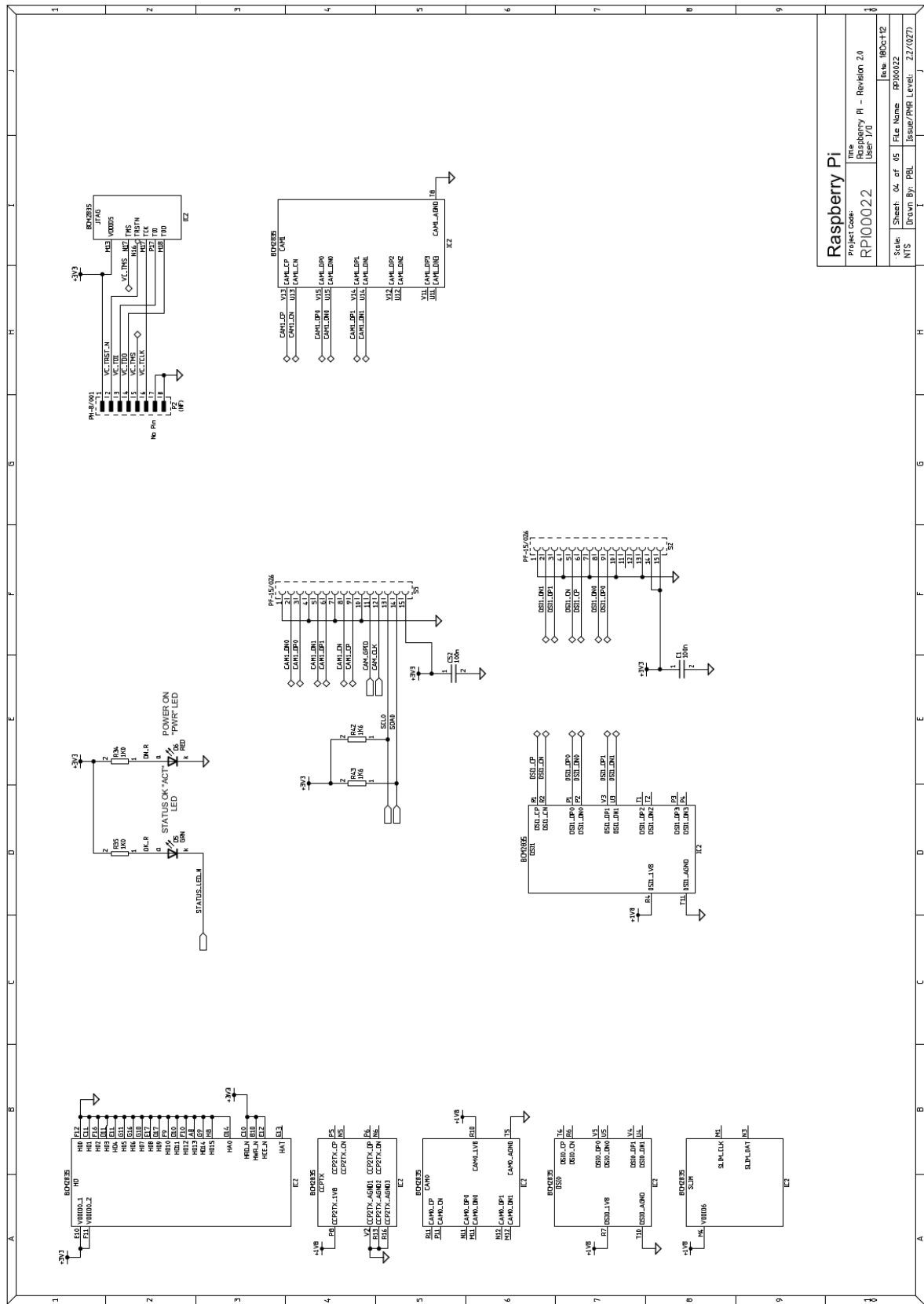
11.4. Az irányszámító egység NYÁK terve



11.5. A Raspberry PI kapcsolási rajza







Raspberry Pi			
Project Code	File Name	Issue/Rev	Rev
RP00022	RP00022	Issue/Rev	Rev
Scale	Drawn By: PBL	Issue/Rev	Rev
NTS	Drawn By: PBL	Issue/Rev	Rev

11.6. Az irányszámító egység kódja

```
#include <avr/io.h>
#include <math.h>

#define F_osc 8000000UL
#define baud 9600
#define ubrr F_osc/16/ baud-1

#define mot1en PD6 //egyes motor engedelyezo (pwm)
#define mot2en PD5 //kettes motor engedelyezo (pwm)
#define mot3en PD3 //harmas motor engedelyezo (pwm)

#define mot13 PB0 //egyes motor forgasirany valtas (első IC)(3A, 4A)
#define mot14 PD7
#define mot21 PB7 //kettes motor forgasirany valtas (első IC)(1A, 2A)
#define mot22 PB6
#define mot31 PD2 //harmas motor forgasirany valtas (második IC)(1A, 2A)
#define mot32 PD4

void usart_init(int myubrr) { //usart inicializalasa
    UBRR0L = (char) myubrr;
    UBRR0H = (char) (myubrr >> 8);
    UCSR0B = (1 << RXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

char usart_get(void) { //usart - karakter fogadas
    while(!(UCSR0A & (1 << RXC0)));
    return UDR0;
}

int usart_get_num(void) { //usart - szam fogadas
    char hundreds = '0';
    char tens = '0';
    char ones = '0';
    char indata = '0';
    do {
        hundreds = tens;
        tens = ones;
        ones = indata;
        indata = usart_get();
    }
    while(indata != '\r');
    return (100*(hundreds-'0')+10*(tens-'0')+(ones-'0'));
}

void pwm_init(void) { //pwm inicializalasa (timer 1 & 2)
    TCCR0A = (1 << COM0A1) | (1 << COM0B1) | (1 << WGM00);
    TCCR0B = (1 << CS02);

    TCCR2A = (1 << COM2B1) | (1 << WGM20);
    TCCR2B = (1 << CS22);
}

void pwm_set(char channel, char val) { //pwm portja es nagysaga
    if (val == 0) {
        if (channel == mot1en) {
            PORTB &= ~(1 << mot13);
            PORTD &= ~(1 << mot14);
        }
        else if (channel == mot2en) {
            PORTB &= ~(1 << mot21);
        }
    }
}
```

```

        PORTB &= ~(1 << mot22);
    }
    else if (channel == mot3en) {
        PORTD &= ~(1 << mot31);
        PORTD &= ~(1 << mot32);
    }
}
else {
    if (channel == mot1en)
        OCR0A = val;
    else if (channel == mot2en)
        OCR0B = val;
    else if (channel == mot3en)
        OCR2B = val;
}
}

int main(void) {
    double rad;

    int deg = 666;

    int pwm1 = 0;
    int pwm2 = 0;
    int pwm3 = 0;

    DDRB = (1 << PB0) | (1 << PB6) | (1 << PB7);
    DDRD = (1 << PD2) | (1 << PD3) | (1 << PD4) | (1 << PD5) | (1 << PD6)
| (1 << PD7);

    usart_init(ubrr);
    pwm_init();

    while(1) {
        deg = usart_get_num();
        if (deg == 666) {
            pwm1 = 0;
            pwm2 = 0;
            pwm3 = 0;
        }
        else {
            rad = deg * 0.0174532925;
            pwm1 = (sin(rad)*200);
            pwm2 = (sin(rad + 2.0943951)*200);
            pwm3 = (sin(rad + 4.1887902)*200);

            if (pwm1 >= 0) {
                PORTB |= (1 << mot13);
                PORTD &= ~(1 << mot14);
            }
            else {
                PORTB &= ~(1 << mot13);
                PORTD |= (1 << mot14);
                pwm1 *= -1;
            }

            if (pwm2 >= 0) {
                PORTB |= (1 << mot21);
                PORTB &= ~(1 << mot22);
            }
            else {
                PORTB &= ~(1 << mot21);
            }
        }
    }
}

```

```

        PORTB |= (1 << mot22);
        pwm2 *= -1;
    }

    if (pwm3 >= 0) {
        PORTD |= (1 << mot31);
        PORTD &= ~(1 << mot32);
    }
    else {
        PORTD &= ~(1 << mot31);
        PORTD |= (1 << mot32);
        pwm3 *= -1;
    }

    }
    pwm_set(mot1en, pwm1);
    pwm_set(mot2en, pwm2);
    pwm_set(mot3en, pwm3);
}
return 0;
}

```

11.7. A képfeldolgozó egység kódja

```
import Pyro4
import serial
import time

Pyro4.config.ONEWAY_THREADED = True

uart = serial.Serial("/dev/ttyUSB0", 9600, timeout=0)

class in_data(object):
    def get_data(self, deg_wait):
        uart.write("%d\r" % deg_wait[0])
        time.sleep(deg_wait[1])
        uart.write("666\r")

server = in_data()

daemon=Pyro4.Daemon(host="192.168.1.106")
ns=Pyro4.locateNS()
uri=daemon.register(server)
ns.register("omnibotserver", uri)

print "Ready."
daemon.requestLoop()
```

11.8. A kezelőfelület kódja

```
import Pyro4
import urllib
import os
import Tkinter
from PIL import Image, ImageTk, ImageDraw

def send():
    server.get_data([int(entry_deg.get()), float(entry_wait.get())])
    entry_deg.delete(0,3)
    entry_wait.delete(0,3)

def video():
    urllib.urlretrieve("http://192.168.1.106:8080/?action=snapshot",
"camera.jpg")
    os.system('convert camera.jpg -distort DePolar "140 62 297,276 0,360"
-resize 100x20% final.jpg')
    image = Image.open("final.jpg")
    draw = ImageDraw.Draw(image)
    if v.get() == 1:
        draw.line((80,0, 80,96), fill="Blue", width=1)
        draw.line((160,0, 160,96), fill="Blue", width=2)
        draw.line((240,0, 240,96), fill="Blue", width=1)
        draw.line((320,0, 320,96), fill="Blue", width=2)
        draw.line((400,0, 400,96), fill="Blue", width=1)
        draw.line((480,0, 480,96), fill="Blue", width=2)
        draw.line((560,0, 560,96), fill="Blue", width=1)
        draw.line((0,0, 640,0), fill="Blue", width=20)
        draw.text((0,0), text="180")
        draw.text((80,0), text="135")
        draw.text((160,0), text="90")
        draw.text((240,0), text="45")
        draw.text((320,0), text="0")
        draw.text((400,0), text="315")
        draw.text((480,0), text="270")
        draw.text((560,0), text="225")
    img = ImageTk.PhotoImage(image)
    pic.configure(image = img)
    pic.image = img
    pic.after(100, video)

server=Pyro4.Proxy("PYRONAME:omnibotserver")
server._pyroOneway.add("get_data")

root = Tkinter.Tk()
root.title("Omnibot controller")

pic = Tkinter.Label(root)
pic.grid(row=0, column = 0, columnspan=20)

label_deg = Tkinter.Label(root, text="Angle:")
label_deg.grid(row=1, column=0)

entry_deg = Tkinter.Entry(root, width=5)
entry_deg.grid(row=1, column=1)
```

```
label_wait = Tkinter.Label(root, text="Time:")
label_wait.grid(row=1, column=2)

entry_wait = Tkinter.Entry(root, width=5)
entry_wait.grid(row=1, column=3)

btn_send = Tkinter.Button(root, text="Send", width=5, command=send)
btn_send.grid(row=1, column=4)

v = Tkinter.IntVar()
check = Tkinter.Checkbutton(root, text="Scale", variable=v)
check.grid(row = 1, column = 5)

video()
root.mainloop()
```