BARABÁSI RICHÁRD
SZAKDOLGOZAT

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
GÉPÉSZMÉRNÖKI KAR
GYÁRTÁSTUDOMÁNY ÉS –TECHNOLÓGIA TANSZÉK

SZAKDOLGOZATOK

SZÁMÍTÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓINTÉZET

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
GÉPÉSZMÉRNÖKI KAR
GYÁRTÁSTUDOMÁNY ÉS –TECHNOLÓGIA TANSZÉK

BARABÁSI RICHÁRD

SZAKDOLGOZAT

Demonstrációs autonóm logisztikai robot fejlesztése

Konzulens:
Abai Kristóf
rendszerfejlesztő mérnök

Témavezető:
Dr. Erdős Ferenc Gábor,
docens

Budapest, 2019

Ide kell befűzni az eredeti feladatkiírási lapot!

NYILATKOZATOK

Beadhatósági nyilatkozat

A jelen szakdolgozat az intézmény által elvárt szakmai színvonalnak mind tartalmilag, mind formailag megfelel, beadható.

Kelt, 2019.12.19.

Az üzem részéről:

_____

üzemi konzulens

Elfogadási nyilatkozat

Ezen szakdolgozat a Budapesti Műszaki és Gazdaságtudományi Egyetem Gépészmérnöki Kara által a Diplomatervezési és Szakdolgozat feladatokra előírt valamennyi tartalmi és formai követelménynek, továbbá a feladatkiírásban előírtaknak maradéktalanul eleget tesz. E szakdolgozatot a nyilvános bírálatra és nyilvános előadásra alkalmasnak tartom.

A beadás időpontja: 2019.12.19.

_____

témavezető

Nyilatkozat az önálló munkáról

Alulírott, *Barabási Richárd* (C3N40R), a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója, büntetőjogi és fegyelmi felelősségem tudatában kijelentem és sajátkezű aláírásommal igazolom, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, és dolgozatomban csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a hatályos előírásoknak megfelelően, a forrás megadásával megjelöltem.

Budapest, 2019.12.19.

_____

szigorló hallgató

TARTALOMJEGYZÉK

KÖSZÖNETNYÍLVÁNÍTÁS

Köszönöm Dr. Erdős Gábor témavezetőmnek, aki segítségével hozzá járult a dolgozat elkészítéséhez.

Köszönöm Abai Kristóf konzulensemnek a szakdolgozat során adott hasznos ötleteit és jó tanácsait.

A jelen szakdolgozat megszületését az "Ipar 4.0 kutatási és innovációs kiválósági központ" című GINOP-2.3.2-15-2016-00002 támogatás tette lehetővé.

* * *

Budapest, 2019.12.19.

Barabási Richárd

JELÖLÉSEK JEGYZÉKE

Latin betűk

Jelölés	Megnevezés, megjegyzés, érték	Mértékegység
a_A	felvett kép szélessége	mm
d	kerék átmérő	mm
D	elmozdulás / lépés	mm
d_A	felvett kép átlója	mm
h	hajtómű áttétel	1
k	kerék kerület	mm
l	szükséges lépésszám adott elmozduláshoz	1
s	lépés / rotor átfordulás	1
S	lépés / tengely átfordulás	1
t	kerék középponttól vett távolsága	mm
t_A	távolság optika és felület között	mm
v	vonalvastagság	mm

Görög betűk

Jelölés	Megnevezés, megjegyzés, érték	Mértékegység
$\alpha 1$	látószög szélességben	fok
$\alpha 2$	látószög átlóban	fok
φ	szükséges lépésszám adott elforduláshoz	1
ϕ	elfordulás / lépés	fok
$\varphi 1$	felbontás szélességben	fok
$\varphi 2$	felbontás átlóban	fok

Indexek

Jelölés	Megnevezés, értelmezés
A	ADNS szenzorhoz kapcsolódó paraméterek

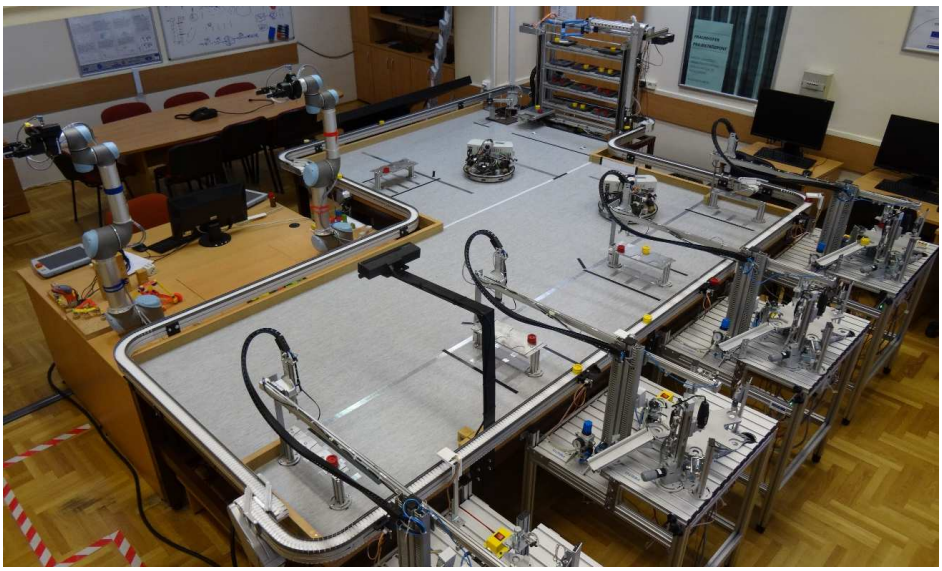
Rövidítések

Jelölés	Megnevezés
AGC	Automated Guided Carts
AGV	Autonomous Guided Vehicle
BLE	Bluetooth Low Energy
CAD	Computer-Aided Design
EMI	ElectroMagnetic Interference

FDM	Fused Deposition Modeling
GPIO	General Purpose In and Output
I2C	Inter Integrated Circuit
ICSP	In Cirtuit Serial Programmer
IDE	Integrated Development Enviroment
IDF	IoT Development Framework
LED	Light Emitting Diode
LIDAT	Light Detection and Ranging
MI	Mesterséges Intelligencia
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
RFID	Radio-Frequency IDentification
RGB	Red Green Blue
RISC	Reduced Instruction Set
SMD	Surface Mount Device
SoC	System on a Chip
THD	Trough Hole Device

1. BEVEZETÉS

A szakdolgozat megírását egy 2019. év tavaszi tárgy előzte meg, melynek a folytatásaként készítem ezt a dolgozatot. A tárgy neve Mechatronika Projekt, amely a BME és a SZTAKI közös tárgya volt, melynek során a SZTAKI-ban található SmartFactory rendszerben fejleszthettem. Ez az 1. ábrán látható. Az ott található minta gyártórendszerben a munkadarabok mozgatását többek között AGV-k végzik. Az akkori feladat ezen egységek leváltása volt, a lehető legtöbb képesség átemelésével. A probléma egyszerűsítésének érdekében egy kereskedelemben kapható robot platformra építkeztünk, mely végül nem elégítette ki teljes mértékben az igényeket. Ez főleg az eszközön már jelen lévő felesleges alkatrészek és a kész kialakítás miatt volt. A szakdolgozatom készítése folyamán ismét ezen a feladaton dolgoztam, ez alkalommal azonban – tanulva a korábbiakból – sokkal kisebb építő elemekből állítottam össze az egységet, ezzel teljes mértékben az igényekre szabhattam azt.



1. ábra SmartFactory

Az elkészült eszköz segítséget nyújthat azon mérnökök számára, akik pályafutásuk során találkozhatnak AGV rendszerekkel. Ezen a demonstrációs autonóm logisztikai roboton megtalálható néhány az iparban használt technológiákból és bemutatható rajta a legtöbb üzemszerű működés közben felmerülő kihívás. Ezek közé tartoznak a pozicionálás megoldása az asztalon optikai szenzoros elven. A kommunikációs protokollok melyekkel az egység alrendszerei és az egységek maguk kommunikálnak. Az RFID megoldások melyekkel a munkadarab azonosítható a munkatérben. A felmerülő problémák például az autonóm mozgás mely csupán a rendszerben levő szenzorokra alapoz. Az optimális útvonal megtalálása a kiinduló pont és cél között, az akadályok és egymás elkerülése figyelembe vételével. Az egységek közti kooperáció egy feladat optimális megoldásához. Az alapvető karbantartási igények, akkumulátor csere, üzemszerű hibaelhárítás vagy akár a fejlesztési folyamat megismerése.

1.1. Célkitűzések

Ahhoz hogy a már meglévő rendszerbe illeszkedjen az eszköz, illetve kielégítsem az egyéb megrendelői igényeket, a fejlesztés elején minimális specifikációt határoztunk meg a konzulensem segítségével. A célom a szakdolgozat készítése során a minimális funkcionalitás implementálása és hibamentesítése, tesztelése, az eszközök és termékek dokumentálása és átadása volt.

A feladat tehát a korábbhoz hasonló, egy gyártó rendszer bővítése AGV-vel: A meghatározott feladatok elvégzésére alkalmas hardverhez szükséges alkatrészek megtervezése, legyártása és tesztelése. Az ezt meghajtó, viselkedésben megfelelő firmware csomag fejlesztése, mely képes magas szintű parancsok értelmezésére, hibák visszajelzésére hasonlóan magas szintű jelzésekkel, anyag mozgatására minimális emberi beavatkozással.

1.2. Dolgozat áttekintés

A dolgozat egyfajta dokumentációként is szolgál, melynek alapján az AGV és eszközei könnyen megérthetők, javíthatók, tovább fejleszthetők. Ebben a bekezdésben rövid áttekintés található a dolgozat felépítéséről, a fejezetek tartalmáról. A szövegesen megjeleníthető anyagok a dokumentum végén találhatók. Az egyéb típusú fájlokat külső mellékletekként a saját formátumaikban csatolom.

A fejlesztés során használt számítógépes, vagy ahhoz kapcsolódó eszközöket a „Fejlesztői eszközök" főcím alatt mutatom be. Minden további érdemleges munkát ezek segítségével tudtam elvégezni. Ide tartoznak a CAD programok, az IDE-k és egyéb létfontosságú eszközök, mint például a logic analyzer.

A „Fejlesztés" főcím alatt a hardverrel és firmware fejlesztését részletezem saját alfejezeteikben. Az előbbit fizikailag elkülöníthető részek szerint, az utóbbit logikailag elkülöníthető funkció csoportokra bontva. A projektet a hardveres tervezéssel kezdtem, emiatt ez a fejezet van előrébb, de nagy szerepe volt az iteratív fejlesztésnek, mert a hardveres összetevők nagyban befolyásolják a programot, és a program korlátozásai egyes hardveres módosításokat igényeltek.

A „Tesztelés" főcím alatt található a fejlesztés során és a kész prototípussal végzett tesztek és mérések leírása, eredményei, tanulságok.

Az összefoglalásban összegzem az elvégzett feladatokat, a sikeres részfeladatok tanulságát, illetve a további fejlesztéseket igénylő részeket.

Ez után a hivatkozások és angol nyelvű összefoglaló található a dolgozat utolsó oldalain. Végül pedig a szöveges mellékletek, mely már nem a dolgozat része.

2. SZAKIRODALMI ÁTTEKINTÉS

Az AGV rövidítés az Autonomous Guided Vehicle angol kifejezést takarja. „A szó magyar jelentése, automatikus irányítási jármű, de szokták vezető nélküli járműnek is hívni. Ez az általános elnevezés sokféle szállító járműre vonatkozhat, (…) A nagyon eltérő komplexitású szerelvényekben az a közös, hogy adott pontok közötti navigációjuk automatikusan történik, azaz az adott távot kezelői beavatkozás nélkül teszik meg.” [1]

Sokan Ifj. Arthur Barrett-et tartják az első AGV szerű jármű feltalálójának. Az 1954-ben elkészített első prototípus egy a mennyezetre akasztott vezetékeket követő, átalakított kistraktor volt. Belső felépítésében és felhasznált technológiákban még nem nagyon hasonlított a mai ipari alkalmazásban lévőkre, de ez volt az első automatizálási feladatot végző vezető nélküli jármű. Egy későbbi típusról a 2. ábrán látható egy 1958-as kivágott újságcikk. [2] Ez a típus már a padlóban lévő jeladókat követte az egység aljában lévő szenzorok segítségével.

2. ábra Újságcikk részlet a „Guide-O-Matic”-ról [2]

A gyártók hamar felfedezték az alkalmazás előnyeit és megjelent a kereslet az AGV szerű járművekre. Manapság pedig már a legnagyobb gyárakban elképzelhetetlen lenne a munka ezen eszközök nélkül. Ilyen például az Intel mikrochip gyártósora, ahol a szennyeződés kockázatának minimalizálása érdekében a gyártás kritikus szakaszai emberektől mentes területen történik. [3] Itt az AGV egységek a mennyezeten rögzített sínrendszeren mozognak és szilikon wafereket tartó kazettákat szállítanak.

Hasonlóan lehetetlen lenne a helyzet olyan környezeteben is ahol az ember számára túl veszélyes vagy lehetetlen lenne a munka. Egy cikkben Jayesh Mehta AGV rendszer telepítését ajánlja a következő esetben: „bármilyen helyzetben ahol az embernek veszélyes környezetben kellene dolgoznia, mint például extrém hőmérsékleti körülmények között vagy erősen maró hatású veszélyes vegyszerek között, nagy távolságú anyagmozgatás vagy ismétlődő feladat esetén.” [4]

Egy cikkben összegyűjtve több fő AGV típust különböztet meg a szerző. [5] Ezek röviden jellemezve:

- Az AGC, vagyis Automated Guided Cart a legegyszerűbb kialakítás. Ezek olyan kiskocsik, melyek emberi beavatkozás vagy erőbefektetés nélkül szállítják a rájuk pakolt tárgyakat. Egy példa erre egy kórházi egység mely eljuttatja a gyógyszereket, a biohulladékot, az éttermi használt tálcákat a megfelelő helyre. Egy ilyen szerkezet a 3. ábrán látható [6].

3. ábra Automated Guided Cart – AGC [6]

- A villás AGV lényegében egy targonca, amely önműködő. Képes raklapok mozgatására, mivel egyes helyeken a speciális paletták használata nem megoldható. Ezen felül kompatibilis hagyományos ember vezérelt targoncás környezetekkel. Egy példa erre a 4. ábrán látható [7].

4. ábra Villás AGV [7]

- A vontató AGV általában önmagában nem képes anyag mozgatásra, de kialakításának köszönhetően képes egy vagy akár több sorba kapcsolt, nem hajtott kiskocsi vontatására. Ilyen típusú a korábban említett első AGV, a „Guide-O-Matic" is. Ezeket autógyárakban előszeretettel használják félkész alkatrészek mozgatására, mint például ajtók, motorok, stb. Ez az 5. ábrán látható [8].

5. ábra Vontató AGV [8]

- Az egység rakomány kezelők egyedi csomagok kezelésére specializálódnak. Ez lehet akár egy speciális alkatrész is, vagy akár egy egyedi kialakítású paletta. Ilyen típusú az általam készített egység is, mivel egyszerre csak egy munkadarabot képes tartani, illetve ebbe a kategóriába tartozik a korábban említett Intelnél működő AGV-k is.

Egy AGV számára a pozicionálás alapvető, de nem egyszerű feladat. Az autonóm járművek pontos beállásán múlik szinte minden. Hiba esetén egy rossz illeszkedés akár a munkadarab leejtését vagy megrongálását is okozhatja. A tervezők ezért általában több rendszert is felszerelnek és párhuzamosan működtetnek. Bár ezek akár egyenként is képesek lennének a feladat ellátására, együtt kizárják a hibázás lehetőségét. Az előző listát tartalmazó írásban az elterjedtebb navigációs technikákat is összegyűjtötték.

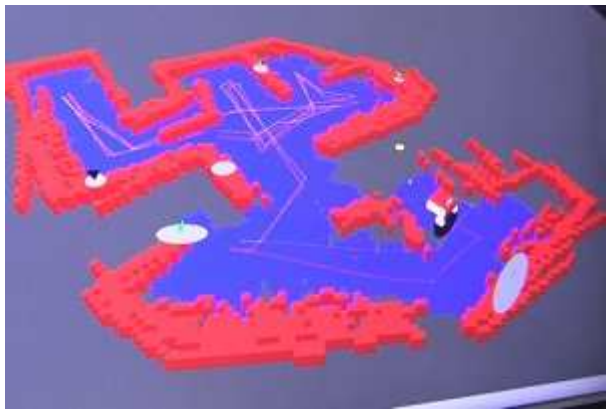
- A vonalkövető fő kategória több gyakorlati megoldással is rendelkezik. A pálya jelölő minden esetben lehet vonal és pontszerű is. Vonalas jelölő esetén az AGV igyekszik a vonalon tartani egy vagy több pontját. Az elmozdulás során első sorban valamilyen belső szenzorral számolja inkrementális módon az elmozdulását. Adott távolságokban a pályán található speciális jel, melynek a gyárban lévő pozíciója állandó és ismert adat. Ezekkel az egység újra nullázhatja az esetlegesen felgyűlt hibáját. Ez a megoldás a 6. ábrán látható baloldalon. A vonalpályát fekete szalag jelzi, a szakasz jelölő pedig az RFID címke [9]. Pontszerű pálya esetén a leggyakoribb megoldás a jelölők négyzetháló metszéspontjain történő elhelyezése. Az egységek csak a szomszédos pontokra mozoghatnak, átlós mozgás nem megengedett. Az egység elindul a következő pont vélt irányába és halad, amíg nem érzékeli a következő pontban lévő jelölőt. Ez a megoldás is a 6. ábrán látható jobb oldalon [10]. A jelölő és szenzor páros működhet mágneses alapon. Ekkor használható állandó- és elektromágnes is. Működhet rádió frekvenciás alapon. A passzív RFID jelölős négyzethálós elrendezés igen gyakori megoldás. Lehet a jelölő vonal valamilyen szalag, melyet optikai módon lehet beazonosítani. Amennyiben ez a szalag fémes, induktív szenzorok is használhatók. A vonalkövetésnek egy speciális változata a síneken futó AGV rendszer. Itt csupán egy dimenzióban szükséges az elmozdulás mérése. egyéb iránt az elv hasonló a vonalszerű pályák működéséhez. A módszer hátránya hogy a kiépítés költséges és gyár átrendezése esetén ismételt beruházást igényel. Ezek a megoldások voltak az elsők és mára elavultnak számítanak. Gyerekjátékok kaphatók ilyen szenzor rendszerrel szerelve. Általában a beruházók igyekeznek elkerülni a gyártócsarnok módosítását, és ennek megfelelő eszközöket részesítenek előnyben.

6. ábra Vonalkövető AGV-k [9] [10]

- A háromszögelésen alapuló módszer is többféle megvalósításban létezik. Van ahol az AGV rendelkezik lézer adó vevővel. Ez lényegében egy speciális LIDAR. Egy síkban minden irányában lézer sugarakat bocsájt ki, és figyeli mely pontokról érkezett vissza nagy intenzitással. A gyárterületen tükröződő felületek elhelyezésével és ezek pontos helyének ismeretével az egység képes meghatározni saját helyzetét. Az 7. ábrán egy ilyen rendszer látható, ez azonban csupán síkban pásztáz a lézerrel [11]. A lézer helyett használhatunk rádióhullámos adóvevőket is. A falakon elhelyezett adók egyedi jelet sugároznak, melyeket az AGV képes érzékelni és megkülönböztetni. A jelekből és a források ismert pozíciójából háromszögeléssel képes az AGV saját helyzetét meghatározni. Ez gyakorlatilag a jól ismert GPS rendszer kis hatótávú beltéri verziója. Ez utóbbi hátránya, hogy az irány nehezebben meghatározható. Bár már létezik a rádióhullám irány érzékeléséhez szükséges technológia, ezen a területen lévő elterjedtségé még nagyon kicsi. A Silicon Labs dolgozik egy megoldáson, amely már a jól ismert és megbízhatónak vélt Bluetooth-on alapszik. A Bluetooth 5.1 szabvány már tartalmazza az irányított sugárzáshoz és irányérzékeléshez szükséges elemeket [12]. Bár ezekből még csupán fejlesztői csomagok kaphatók és kész termékek nem igazán, hamarosan ezzel is bővülhet a beltéri pozicionálási lehetőségek palettája. A kiépítés és a szenzorrendszer nagyjából egy árban lehet az előző kategóriával, azonban amíg a falak és oszlopok nem kerülnek elmozdításra, nincs szükség a jelölők áthelyezésére. Ezeknek a megoldásoknak is van azonban hátránya. A lézeres megoldás csak direkt rálátás esetén működik. A rádióhullámnak nem szükséges ez, de zavaró tárgyak mindenképpen rontják a pontosságot. A szenzorok maguk is sokkal összetettebbek, ami az újabb egységek költségét növeli. Szoftveres oldalról szintén bonyolultabb ezeknek a megoldásoknak a megvalósítása, átrendezés esetén pedig a térkép frissítése bonyolultabb, mint a korábbinál.

7. ábra Lézeres pozicionáló rendszer [11]

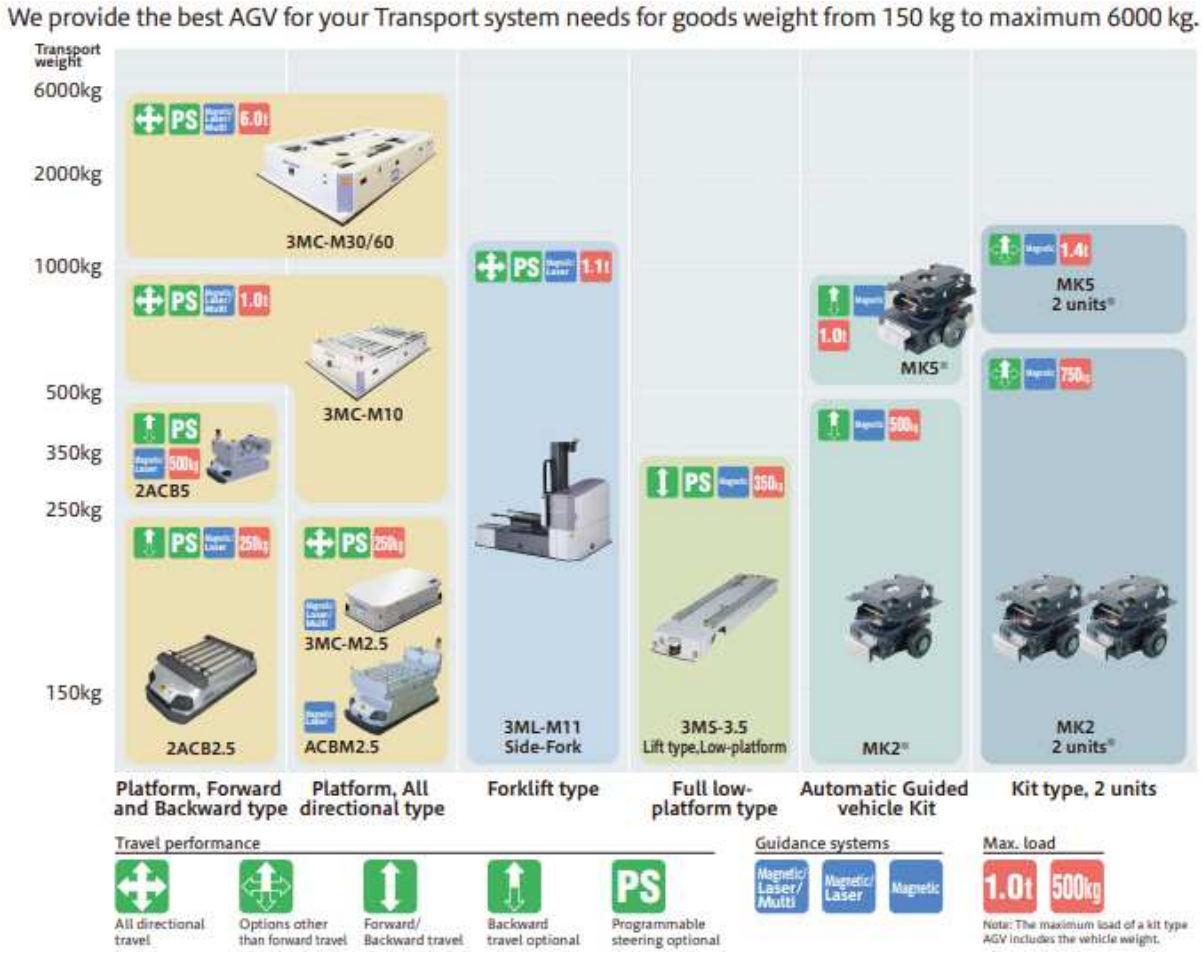
- A látás alapú pozicionálás megvalósítható kamerák és szofisztikált képfelismerő szoftverek alkalmazásával. Ez erősen alapoz az MI tudományterületre is, hiszen a kialakítás egyáltalán nem bonyolult, itt a szoftveren van a hangsúly. A kamerák elhelyezhetők az AGV-n vagy a gyártócsarnokban is. Az előbbihez szükség van egy nagyméretű adatbázisra, amiben el van tárolva a csarnok 3D modellje és textúrái. Hasonló a Google térkép utca nézetéhez. Az MI szoftver felismeri az alaksajátosságokat és elhelyezi magát a térben. A csarnokban elhelyezett kamerák esetén az AGV valamilyen címkével van ellátva. Ez legtöbb esetben egy QR kód. Amennyiben legalább egy kamera látja az egységet, meg tudja állapítani a pozícióját és az irányát is. Elterjedt még a LIDAR használata is, mely teljes gömbben képes a távolságok valós idejű mérésére, ezzel megalkotva a jármű körüli tér 3D modelljét. A 8. ábrán egy LIDAR térkép látható [13]. A kék területen nincs akadály, ez a mozgástér. A piros pontok akadályt jelölnek. Ez a térkép megtalálható a központi feldolgozóban, de egy egység ennek csupán egy részét látja egyszerre, hiszen a távolságmérés csak a rálátott pontokban lehetséges. Az egység elhelyezése könnyű a térképen, az alapján hogy milyen alakú akadályokat lát. A kialakítás nagy előnye hogy semmiféle módosítás nem szükséges a területen. Kivétel ez alól a kamerák felszerelése, de ez is elhanyagolható költségű. Nyilvánvaló hogy nagy terület vagy rosszul belátható területen sok kamera elhelyezése lenne szükséges minden pont látásához. Ilyen esetben ez a megoldás nem optimális. A rendszer bővítése nem jár extra költségekkel, ugyanis a térkép másolására van csupán szükség. A rendszerbe bekerülő ideiglenes akadályok felismerhetők, az ütközés elkerülhető. Ez különösen fontos olyan környezetben ahol emberek is dolgoznak. A csarnok átrendezése esetén a térképet újra kell generálni, de ez a megfelelő eszközök segítségével szintén nem nagy kihívás. A nagy akadály ismét kiemelve a rendszer szoftveres oldalon lévő bonyolultsága.



8. ábra LIDAR térkép megjelenítése [13]

Ezzel tárgyaltuk az AGV-k főbb típusait és a fontosabb pozicionálási megoldásait, most pedig a meghajtási és kanyarodási módszerek következnek. Ezek három csoportba oszthatók, két fő típusra és ezek kombinációjára. Az autóknál hagyományos kanyardó kerék és fix irányú kerék kombináció a nagyobb méretű és vontató egységeknél terjedt el. A kisebbek helygazdaságosság és manőverezhetőség érdekében nem rendelkeznek kanyarodó kerekekkel. A kanyarodást a kerekek különböző sebességű forgatásával oldják meg. Ezek általában képesek akár egy helyben is megfordulni. Ilyen típusú az általam tervezett AGV is. A kettő kombinációja az egyedileg hajtott elfordítható kerekes megoldás. Ezek főleg speciális igényű alkalmazásokhoz használatosak, de a legtöbb targonca AGV is ilyen kerékkel rendelkezik.

A piacon akár katalógusból rendelhető modelleket is kaphatunk, számtalan gyártótól számtalan konfigurációban. Szinte minden esetben megtalálhatjuk az adott alkalmazáshoz tökéletes egységet. Egy katalógus kivágás a 9. ábrán látható [14].



9. ábra AGV katalógus részlet [14]

3. FEJLESZTŐI ESZKÖZÖK

A célok eléréséhez előre látható volt, hogy komoly fejlesztési munkára lesz szükség. Manapság a tervezés elengedhetetlen elemei a magas szintű programok használata. Legyen szó akár az áramköri akár a mechanikai tervezésről vagy akár a dokumentálásról, szofisztikált programok nélkül kivitelezhetetlen lenne. Éppen ezért ebben a részben felsorolom és röviden jellemzem a fejlesztés során használt eszközöket. A 3D modellezésre Autodesk Inventror 2019-et használtam. A nyák tervezésére Altium Designer 2017-et. A szoftver fejlesztés nagy részét Eclipse IDE-ben végeztem, kisebbik részét Arduino IDE-ben. Néhány egyéb eszközt is megemlítek a „Kiegészítő eszközök" bekezdés alatt, melyekben jelentős munkát nem végeztem, de amelyek nélkül nem készülhetett volna el ez a projekt.

3.1. Inventor

A korábbi tapasztalatokból látszott, hogy a test nem boltban kapható elemekből lesz össze szerelve, hanem saját tervezésű lesz. Az egyéb elemek ütközésmentes elhelyezésének és a szerelhetőség érdekében pedig a teljes szerkezetet szerettem volna vizsgálni még a tervezési fázisban. Az emelőkar csuklós mechanizmusát is a programban lévő eszközök segítségével terveztem meg. A beállítandó paraméterek első sorban a karok hossza volt a megfelelő emelési magasság és pofa nyitás eléréséhez. A tervezési folyamat elején készítettem néhány koncepciós modellt melyekből témavezetőm segítségével kiválogatva a jó ötleteket elkészült az első prototípus.

Maga a program az egyik legismertebb 3D modellező szoftverek között. Számos beépülő modullal rendelkezik, például a VEM és a prezentáció tervező. Mint kizárólagosan használt 3D modellező program, minden modell fájl az itt használt formátumban található meg a külső mellékletben, „.ipt" és „.iam" kiterjesztéssel. A további munkák megkönnyítése érdekében általános formátumba is exportáltam a végső terveket. A 3D nyomtatás alapjául szolgáló „.stl" fájlok is az Invertorból exportáltak.

3.2. Altium

A kezdeti koncepció még próbapanelen állt össze, ám hamar kiderült, hogy ez nem elég a jelenlegi projekthez. Az alkatrészek számának növekedésével pedig a kapcsolás bonyolultsága is nőtt. Ez egyrészt kezdett átláthatatlanná válni a próbapanelen, másrészt egyre nagyobb lett az igény a jó dokumentáltságra. Az áramköri kapcsolás dokumentálásához is kell egy megfelelő program és mivel korábbi munkáim során már megismerkedtem egy professzionális áramkörtervezővel, úgy gondoltam megéri a nagyobb energia befektetés az eredmény érdekében. Végül az Altium lett a projekt áramköri tervező programja.

Ez egy régre visszanyúló gyökerekkel rendelkező nyomtatott áramkör tervező program. Bár a funkciók mennyisége és minősége ipari sztenderdé tette, én csupán kis részét használtam ki a programnak. Meredek tanulási görbéje először ellenszenves lehet, de ha sikerült elsajátítani az alapokat, egy minden várakozásnak eleget tevő robosztus és megbízható programot kapunk.

Mint kizárólagosan használt PCB tervező, minden modell fájl az általa generált könyvtárstruktúrában található. A gyökérkönyvtárban található projekt fájl megnyitásával a program megjeleníti a belső fájlkezelőjében a kapcsolási rajzot, a nyáktervet és a könyvtár fájlokat. Igen hasznos funkció a megrajzolt nyák 3D exportálása. Az így exportált nyák modellt be tudtam építeni az Inventorban készített 3D modellbe még részletesebb tervezést téve lehetővé.

3.3. Eclipse

Az Eclipse az Inventor után, de még az Altium előtt került a projekt eszköztárába. Ez a program egy IDE, vagyis integrált fejlesztői környezet. Kerestem megoldásokat a központi feldolgozó egységre és a végül kiválasztott mikrokontroller, az ESP32, ezzel programozható. Arduino IDE is képes programozni az eszközt, de szerettem volna az eddigi mentalitáshoz tartani magam, és minél alacsonyabb szintű elemekből felépíteni a projektet. Az Arduino alatti rendszer a hivatalos fejlesztői keretrendszerre épül, de elemelkedik attól. Az említett keretrendszer az ESP-IDF. Az Eclipse képes a kertrendszer integrációjára, a projektek fordítására és a program égetésére.

Az Eclipse egy IBM-es kezdeményezésként indult, majd miután több nagyobb cég is beszállt, open source projektként futott tovább. Bár első sorban Java fejlesztésre használják, számos más nyelvet is támogat a közösség által írt számtalan mennyiségű bővítménynek köszönhetően.

A bővíthetőségének köszönhető hogy a közösség körében ez a program vált az ESP32 elsődleges fejlesztői környezetévé. Itt érdemes megjegyeznem azoknak, akik a Visual Studio-t részesítik előnyben, hogy a Visual Studio Code is beállítható az ESP-IDF projektek fordítására. Néhány a mellékletben leírt lépésben bemutatom, hogyan lehet előkészíteni az Eclipse-t és felprogramozni vele az ESP32 mikrokontrollert.

A telepítés több lépéses, néhány aprósággal melynek kihagyása gondot jelent. Ezek szintén a mellékletben találhatók. A dokumentum írásakor az Eclipse integráció még nem támogatott hivatalosan, azonban a honlapon található leírás szerint az IDF 4.0 már alapból olyan beállításokkal érkezik, ami igen könnyűvé teszi ezt a lépést [15]. Érdemes ott kezdeni és amennyiben megjelent az új verzió nem az általam leírtakat követni. A telepítés két fő lépésből áll, az első a megfelelő verziójú ESP-IDF telepítése, a második az Eclipse telepítése és konfigurálása.

3.4. Kiegészítő eszközök

Ezek az eszközök úgymond támogató szerepet töltöttek be. Nem sok időt töltöttem velük, vagy nem közvetlen ezeket használtam, de ezek is nélkülözhetetlenek voltak.

3.4.1. ESP-IDF

A rövidítés az Espressif IoT Development Framework szavakat tömöríti. Az Espressif egy kínai cég amely SoC-k fejlesztésére specializálódott. Legújabb termékcsaládjuk az ESP32, mely jelen projektben is a központi feldolgozóegységet képezi. Ehhez a termékhez készítettek és jelenleg is aktívan fejlesztenek egy keretrendszert is, ennek a neve: ESP-IDF. [16]

Ezen eszköz és az Arduino IDE népszerűségének köszönhetően elkészült a támogatás Arduino fejlesztői környezetre is, azonban ez főleg közösségi fejlesztésű, és az ESP-IDF-re épül, ami mindenkori lemaradást jelent a legújabb funkcióktól, hibajavításoktól, amit a fejlesztő kiad. Ezen felül az Arduino-s támogatás azt jelenti, hogy az ESP által nyújtott API-t megpróbálják az Arduino-ban jól megszokott API alá szervezni. Ez nem jelent gondot kisebb projektekben, de előre nem belátható problémákat okozhat egy ekkora projekt során. Mind ezek fényében úgy döntöttem, hogy az ESP-IDF-et használom Eclipse segítségével a könnyebb, de adott esetben korlátozottabb Arduino IDE helyett.

3.4.2. GIT

Volt már egy kevés tapasztalatom nagyobb méretű szoftverfejlesztésben és tudtam, hogy verziókövetés nélkül nem lehet egy két három forrásfájlnál többből álló projektet kezelni. Létre is hoztam egy privát GitHub repositoryt. A Git intézi a verziókövetést, a GitHub pedig egy központi tárhelyként szolgál, ahova fel lehet tölteni a Git programból a kódot. Ezen felül hasznos eszközöket is ajánl mind az online megtekintéshez, mind a kód analizálásához. Én főleg az előbbit használtam, mivel korábbi verziójú kódot is meg lehet tekinteni az online felületen.

A Git Linus Torvalds által fejlesztett elosztott verziókövető rendszer. A szoftver fejlesztés során végzett változtatásokat tartja számon, optimalizálja a nagyobb csapatok párhuzamos munkáját egy adott kódbázison, de sok előnnyel jár kisebb projektek során is, mint például az AGV firmware csomagja. Számomra a korábbi verziók visszanézhetősége volt a leghasznosabb funkció, de szabadabban, kockázat nélkül írhattam át egész forrásfájlokat, a korábbi munka elvesztésének veszélye nélkül ezen eszköznek köszönhetően.

Az Espressif által fejlesztett ESP-IDF is git verziókövetett, a kód GitHub-on elérhető. A fejlesztéshez kell ebből egy másolat. A kiinduló ESP-IDF verzió v3.2.2 volt, de jelenleg 4.0-n alapszik a projekt. Minden verzió váltás után ki kell adni egy parancsot, hogy

a git feloldja a repository külső függőségeit is. Ez azért szükséges, mert az ESP-IDF tartalmaz néhány modult, melyek más git repository-k beágyazva az ESP-IDF repositoryba. Alap esetben ezeket nem másolja le a git, de a működéshez ezek is szükségesek.

3.4.3. LOGIC ANALYZER

Itt is korábbi – digitális áramkör fejlesztéséből eredő – tapasztalatok miatt került a projektbe az eszköz. A hardveres hibakeresés alapvető eszköze, de akkor is hasznos volt, amikor nem tudtam, hogy a programom vagy a kapcsolásom hibás. A fejlesztés során a legnehezebb részek a digitális kommunikációs vonalak szoftveres kialakítása volt. Ekkor órákon keresztül iteráltam az apró kód módosítás és a alaplapon valóban lejátszódó digitális jelek megfigyelése között. A problémát az jelentette, hogy az összes alkatrész egyedi igényekkel rendelkezik, finomhangolásra volt szükség.

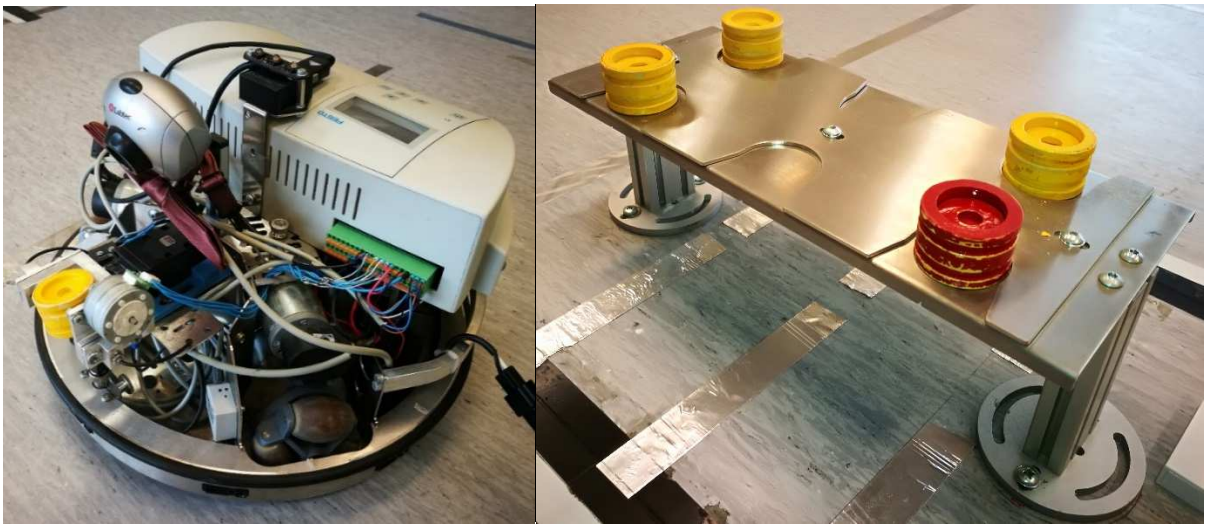
Az általam használt eszköz márka és típusjelzés nélküli. 24MHz-es és 8 csatornás. Ez a 10. ábrán látható baloldalon. A hozzá használt program ingyenesen letölthető. A neve Logic és a Saleae cég oldaláról letölthető [17]. Az eszköz USB-vel csatlakoztatható számítógéphez, ahol egy grafikus programon keresztül használható. Hasznos funkció hogy a használt kommunikációs protokoll megadása után dekódolni tudja a digitális jelsorozat információ tartalmát. Az lent látható 10. ábrán, jobboldalon például ez I2C buszon történő adatmozgás és a dekódolt bájtok láthatók.

10. ábra Logic analyzer és szoftvere

4. FEJLESZTÉS

Már az elején beláttam, hogy bármennyi időt is fordítok a tervezésre az első prototípusokban sok apró és néhány kritikus hiba is elő fog fordulni. Ezért a fejlesztés több iterációból állt, némely alkatrész és még több a szoftveres komponens több generációt is megélt mire végleges formájukat elnyerték. A régebbi verziókat külön kiemelem ott ahol egyes hibás döntéseket vagy megfontolásokat a prototípusnak köszönhetően fedeztem fel, vagy esetleg más okból érdemes megemlíteni.

Kiindulási alapként a már meglévő Robotino nevű mobil robot szolgált. Ez a 11. ábrán, baloldalon látható. A rendszerben már meglévő elemek a fontosabbak említésével: az alapterület, a munkadarabok, a fészkek. Robotino-ból két egység is található a SmartFactoryban (1. ábrán látható). Képesek az asztalon navigálni, kiadott parancs alapján pozícióra állni, a megfelelő állásban egy munkadarabot a fészkekből felvenni vagy letenni. A fészkek is a 11. ábrán láthatók jobb oldalon. A pozicionáláshoz az alapterületen elhelyezett jelölőket használnak. A váz kerületén infravörös ütközés érzékelő szenzorok vannak elhelyezve. Az egység tetején kijelző található, melyen belső állapot adatai megjeleníthetők. A megfogója át tudja fordítani a munkadarabot egy tengelye körül. Képes a két egység kooperatív mozgása, vagyis úgy közlekedni a munkatérben, hogy nem ütköznek össze.



11. ábra Robotino és fészek

Ezek az egységek nagyméretűek és nehezek. Egyszerre legfeljebb kettő tud kényelmesen dolgozni a munkatérben. Az energia ellátásért egységenként két darab gondozásmentes akkumulátor felel. Főleg ez és a megerősített karosszéria adja a tömeg nagy részét. A központi feldolgozó egység a feladathoz képest túlméretezett és bonyolult. A számítási kapacitása egy asztali számítógépéhez közelít, mely feleslegesen nagy fogyasztással is jár. Röviden össze foglalva a munkadarabhoz és a feladathoz képest a korábbi egységek nagyok és drágák.

4.1. Feladat specifikálása

A korábban említett képességekből a munkadarab átforgatásán kívül mindennek meg kell jelennie az új egységben. A felismert gyengeségeket is javítani kell. A célok tehát a következők: Könnyű és kisméretű test tervezése. A test legyen könnyen beszerezhető és olcsó. A kisebb méretű test miatt megjelent az igény a magasságkülönbség kompenzációjára, mivel így a munkadarabok tartói relatíve magasabbra kerültek, így az új kialakításhoz új megfogó szerkezet szükséges. A tömegből az akkumulátor típusának cseréjével is sokat lehet lefaragni. A kapacitás is csökkenhet, hiszen kisebb testhez kisebb motor kell, illetve a feldolgozó egység is kisebb áramfelvételű lesz. Emiatt fontos a lehető legegyszerűbb, de még kielégítő teljesítményű központi feldolgozó processzor megtalálása. Itt is fontos az ár és fontos a használhatóság. Nagyon új eszközt nem érdemes használni, hiszen ott kevés a közösség tapasztalata, kevés a fórumozók száma. A korábbi egységek képességein felül további bővítésekre is merültek fel igények. Ezek közül egy a hordozott munkadarab azonosítása. A munkadarabok tartalmaznak passzív RFID címkéket és a „gyárterületen" több helyen található RFID olvasó, de a korábbi AGV egységekben nem volt. Felmerült az igény egy új, a munkatérben elhelyezett jelölő szalagot nem igénylő pozicionáló rendszer kialakítására. A közlekedési jelzőfények és gyors státusz jelző fények pedig implementálás esetén látványos extraként szolgálnának.

A funkciók elvégzéséhez minden esetben kell egy szoftveres összetevő is az elhelyezett hardveres alkatrészek meghajtására. A programmal szemben támasztott igények az átátható forráskód, a hibamentes működés és az alapvető funkciók ellátása. A firmwarenek képesnek kell lennie az összes az alaplapon lévő eszköz használatára. Ehhez első sorban az alaplapon elhelyezett chipek közötti kommunikáció kialakítása szükséges. Ezekre a kommunikációért felelős függvényekre épülhetnek a magasabb szintű funkciók. Ezek a funkciók képesek szöveg megjelenítésére a kijelzőn, vagy a motor elforgatására adott szöggel, illetve ezek felelnek a munkadarab felvételéért és a pozíció folyamatos számon tartásáért is.

Listába szedve a következő feladatok megoldása szükséges:

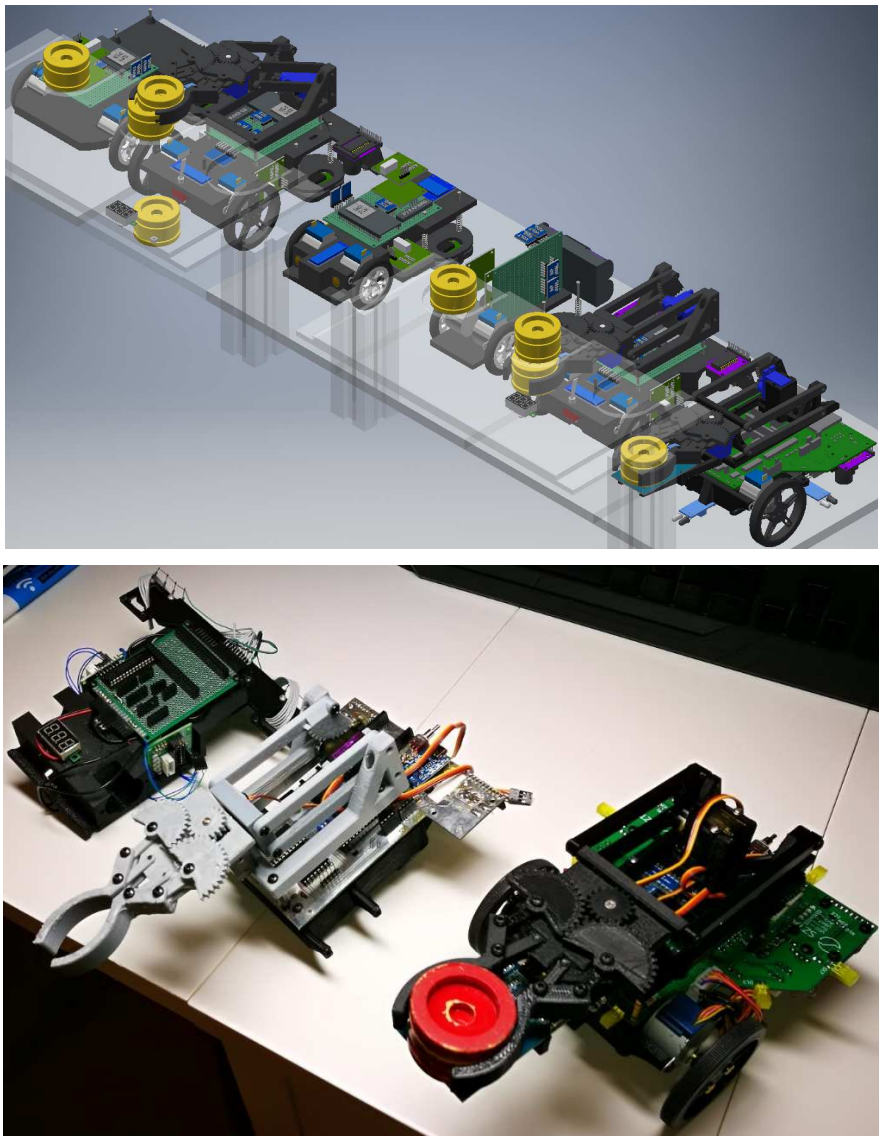
- Legyen az egység olcsóbban előállítható és kisebb, mint az előző
- Megfogó szerkezet tervezése
- Könnyű, nagy kapacitású akkumulátor, töltő áramkör beszerelése
- Megfelelő feldolgozó egység választása beépített vezeték nélküli kommunikációval
- Kommunikációs csatornák kiépítése
- Munkadarab azonosítás megoldása
- Újfajta pozicionáló rendszer kifejlesztése
- Kijelző és jelzőfények elhelyezése
- Magasabb szintű funkciók implementálása
- Nyáktervek dokumentálása, forráskódok kiegészítése magyarázatokkal

A következőkben pedig bemutatom magát a fejlesztési folyamatot. Az alfejezetek nem időrendi sorrendben vannak, hanem funkcionális blokkokban mutatják be a fejlesztést. A két fő blokk a hardver és a szoftver, melyek további alfejezetekre vannak bontva szintén a funkcionális alegységek szerint.

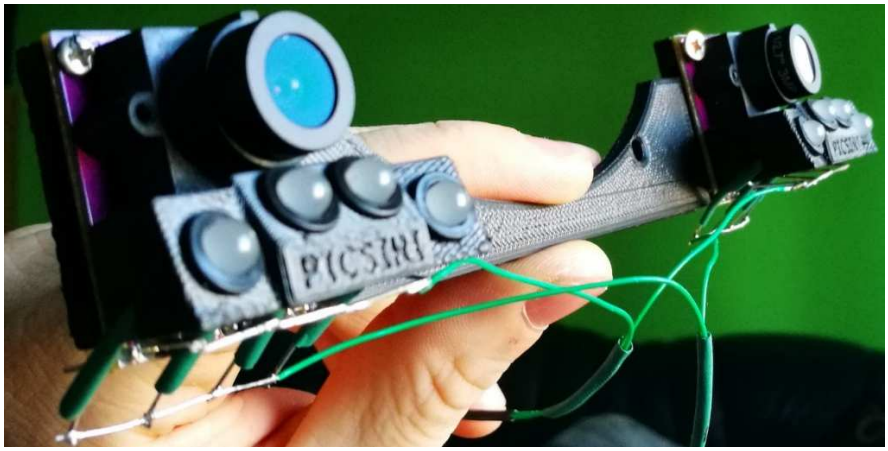
4.2. Hardver készítése

Miután tisztáztam a követelményeket, nekikezdtem a megvalósításnak. Elsőként felvázoltam mely nagyobb egységekre bontva tudom párhuzamosan, egymástól többé kevésbé függetlenül tervezni az alkatrészeket. A felbontás egy szempontból nagyon lényeges volt. A párhuzamos tervezéssel időt spórolhattam, mikor egyik folyamatban várnom kellett – mondjuk a gyártásra – a másik elem tervezésével tudtam haladni. Első ilyen egység a test, melynek a formatervét főleg a platform típusa határozza meg. Befolyásolják még más alkatrészek is, de ezek csupán apró módosításokat jelentenek az általános kialakításon belül. A második fő egység az alaplap, mely önálló projektként is fejleszthető. A belső kapcsolások száma mellett messze elmarad a külső egységek kapcsolatainak száma, így értelem szerű önálló egységként kezelni. A gyártás helye, ára és átfutási ideje is erősen eltér e két egységnél. Mikor már nyákot is terveztem, igyekeztem azt tökéletesíteni, és probléma esetén, a testen apró módosításokat végezni és újra gyártatni. A motor főleg azért kapott külön egység státuszt, hogy szükség esetén könnyen kicserélhető legyen egy nagyobb nyomatékú vagy gyorsabb motorra. A megfogó kar az AGV felépítményeként csupán néhány csavarral és csatlakozóval kapcsolódik. Az első kar nagyjából a fejlesztés közepén rövid idő alatt elkészült, működését pedig az alépítmény megléte nélkül is tesztelni tudtam.

Az első prototípus soha nem került fel a tervezett helyére, mivel az a verziójú test, amihez készült soha nem került gyártásba. Az újratervezett, második generációs kar szerkezet látható a végleges prototípuson a 12. ábrán alul. A verziók balról jobbra I-től III-ig láthatók. Karból csak két fő verzió készült. Ugyanitt a felső részen látható a modellek generációs fejlődése. Van köztük soha meg nem épült koncepció terv is. Bár szívesen meghagytam volna az eredetieket a teljes kiépítettségükben, a drágább és könnyen szerelhető alkatrészeket a költségek csökkentése érdekében kibontottam az elődökből. Az ábrán az is látható hogy az első verzió még próbapaneles nyákkal rendelkezett és az ADNS modulok egy 3D nyomtatott szerkezeten vannak. Ez a 13. ábrán látható. Ugyanígy különálló modulként volt a testen az ULN2003 is. Az ez utáni egységeken közvetlenül a nyákon van mind a két modul típus.



12. ábra 3 generációnyi AGV



13. ábra ADNS-3080 és első tartója összeszerelve

4.2.1. PLATFORM TÍPUSA, A TEST

A „Mechatronika projekt” című tárgy alatt megismert eszközből, az AlphaBot2 nevű vonalkövető robotból indultam ki [18]. Ez a 14. ábrán látható. Ötletelési alapnak jó volt, azonban sok apró hibával rendelkezett mely végső soron alkalmatlanná tette, mint platform a jelen projekthez.

14. ábra AlphaBot2 [18]

A fontosabbakat kiemelve:

- Az AlphaBot2 hajtóműves DC motoros meghajtást tartalmazott. A DC motornak csupán a feszültségét lehetett vezérelni egy PWM jel segítségével. Minden komolyabb pozicionáláshoz használható szenzor hiányában azonban a robot még csak egyenesen sem tudott haladni.
- A kerék tengelye körül billegett a jármű a nem rugalmas, négy pontos alátámasztása miatt.
- A felhasznált mikrokontroller minden lába funkciót látott el, így nem maradt elég GPIO a saját szenzorok felszerelésére, miközben az eszközön sok volt funkció betöltéséhez felesleges elektronika.
- Gyenge volt a felhasznált mikrokontroller.
- Nem volt könnyen hozzáférhető az akkumulátor tartó.

A saját tervezésű platformnál mindezeket figyelembe vettem, és amit megtartottam, az a felfüggesztés típusa, vagyis két meghajtott kerék és csúszó alátámasztás. Az 1. táblázatban látható három alternatíva, melyek magyar forgalmazóknál kaphatók, így potenciális jelöltek voltak. A választásnál figyelembe vett szempontok a következők:

- A hatkerekű platform lépcsőmászásra való, nekünk sima talajon kell mozognunk, feleslegesen túlbonyolított mechanika lenne.
- A négykerekű legfőbb problémája a hatkerekűben is megvan, mégpedig hogy nem tud helyben fordulni. Ebből sok plusz munka adódik programozás során, hiszen így a pozícióba állás sokkal körülményesebé válik.
- Végül a kétkerekűre esett a választás, mely mellett az is szól, hogy a korábban a SmartFactoryban dolgozó Robotino is két hajtott kerekű, igaz azok omnidirectionális kerekek.

2 Kerék + támasztás	4 Kerék	6 Kerék
[19]	[19]	[20]
		

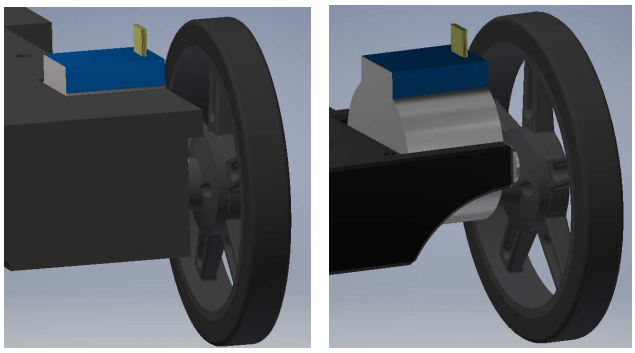
1. táblázat Leggyakoribb platform típusok

A típus kiválasztása után azonban úgy döntöttem, hogy nem kész platformot vásárolunk. Nagyobb feladat, de kifizetődőbb kezdettől fogva olyan platformon dolgozni, amely minden téren a saját igényeinkre van alakítva. Ezen a ponton kezdődött el az első vázlatszerű modellek megalkotása, melyekből később összeállt a Mark I-es. A dolgozat leadásakor a test verziója Mark III.

A specifikációban leírtak alapján az elérhető eszközök figyelembe vételével a 3D nyomtatási technológia került kiválasztásra a test gyártásához. Mivel a SZTAKI rendelkezik FDM nyomtatóval, házon belül gyártható a test. Továbbá egyes kisebb alkatrészeket én is ki tudtam nyomtatni a saját nyomtatómmal.

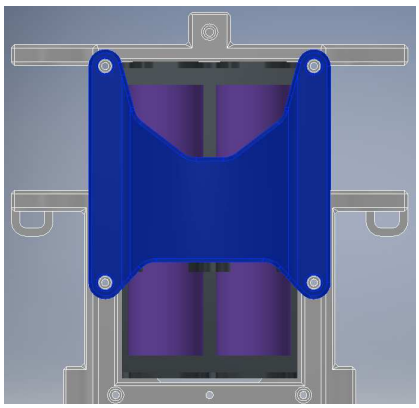
A test tervezése során is fontosnak tartottam, hogy az első ötletek a lehető leghamarabb kerüljenek kinyomtatásra, és tesztelhessem őket a modellezőprogram keretein kívül is. A főbb verziók a legtöbb alkatrész újra tervezését jelentették, de az egyes szinteken kiegészítő alkatrészek is kerültek kinyomtatása. Fontos kialakításbéli előre lepéseket az egyes verziók között itt sorolom fel:

- Az első verziós test teljesen körbe ölelte a motorokat. Egyik korai teszt során azonban észrevettem, hogy a motorok hamar fel tudnak melegedni. A megfelelő hűtés érdekében a második verziótól kezdve a motorok kiálló bakokon kerültek rögzítésre. A 15. ábrán, baloldalon az első jobb oldalon pedig az utóbbi látható.



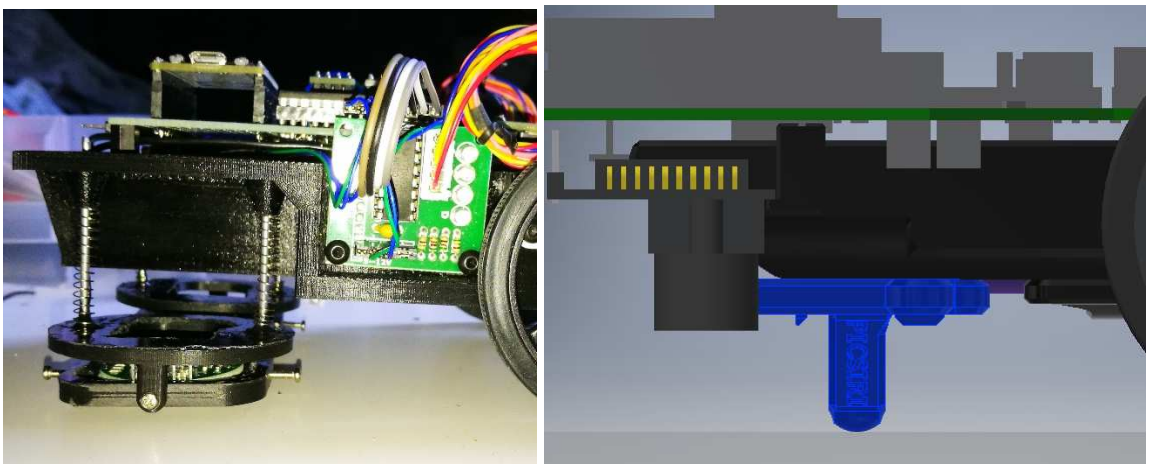
15. ábra Kerék foglalat fejlődése

- Az akkumulátor tartók első ötletem alapján szorító kötéssel voltak rögzítve. Mivel alig tudtam az akkumulátorokat beszerelni, később pedig kiszerelni, a második verziótól kezdve laza foglalatban, csavarozható fedőlappal vannak rögzítve. A konstrukció a 16. ábrán látható. A lila hengerek az akkumulátorok, a kékes színű darab pedig a lecsavarozható fedő.



16. ábra AGV alsó kialakítása

- Az első verzió nem tartalmazott harmadik megtámasztási pontot, mivel ekkor még egy más szenzor rendszeren alapuló megoldásban gondolkoztam. Ez a megoldás gondoskodott volna a megtámasztásról. Ennek hiányában azonban terveznem kellett egy könnyen cserélhető alkatrészt, mivel ez a csúszás miatt folyamatosan kopik és cserélést igényel. A 17. ábrán, baloldalon az első tervek jobb oldalon pedig a végső megoldás látható.

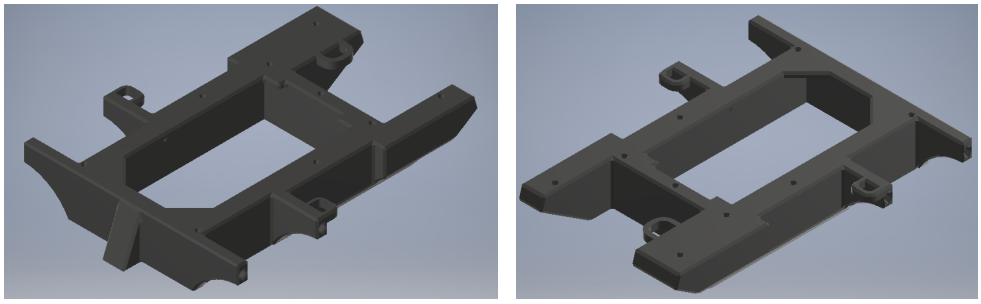


17. ábra Első prototípus és végleges kialakítás hátsó támasza

- A második verzió nem tartalmazott rögzítőpontokat az ütközésgátló szenzorok számára. Ugyanekkor alkalmaztam egy főleg esztétikainak mondható fejlesztést is. A harmadik verzióra kerültek kábel rögzítő fülek a tisztább vezetékezés érdekében. A korábbi 16. ábrán a kábelvezető fülek is láthatóak.

- Folyamatos újításokkal pedig sikerült az utolsó verziót közel fele annyi feature segítségével leírni, mint az elsőt illetve teljesen támaszanyag mentesen nyomtathatóra tervezni. Ezzel a 3D nyomtatás egyik legidőigényesebb munkafolyamatát spóroltam meg, a munkadarab utókezelését. E mellett a keletkező hulladék mennyiségét is csökkentettem evvel.

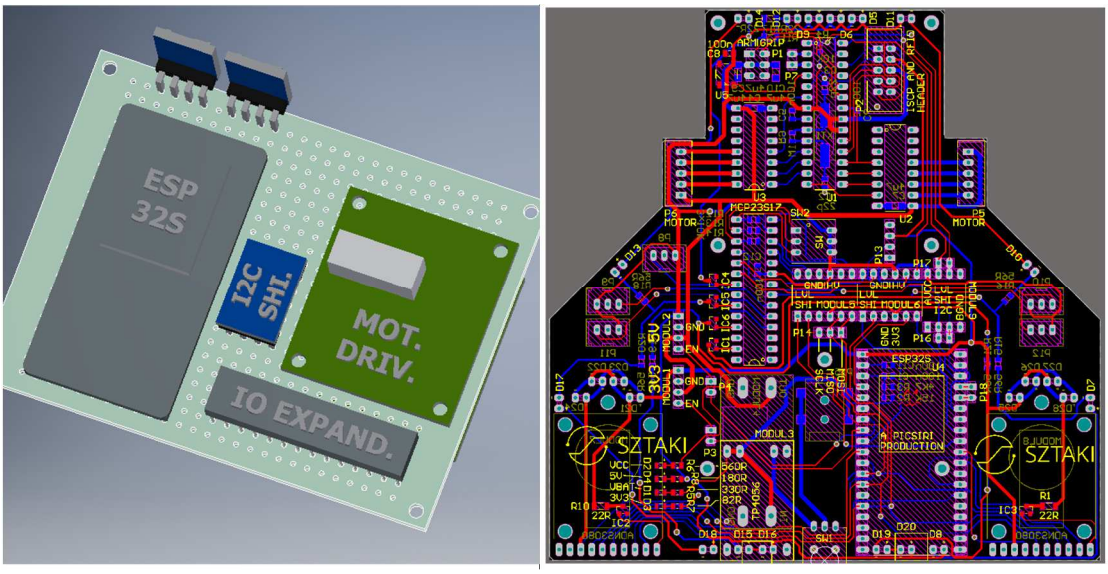
Végül az elkészült test modell önmagában a 18. ábrán látható.



18. ábra Végső test kialakítása

4.2.2. Alaplapi és külső komponensek

Ez a szekció a robot központi elektronikájának alkatrészeit veszi sorra. Ezek az elemek főleg a nyákon helyezkednek el, de némelyik csak vezetékkel csatlakozik rá. A 19. ábrán látható az első, még próbapanelen összerakott alaplap modellje (balra), illetve a leadáskor végleges, Altiummal készült modell (jobbra). A lehető legtöbb alkatrészt igyekeztem egy helyről beszerezni. A választott forrás a HESTORE nevű webáruház. (Hobbi Elektronika Store) Néhány alkatrész azonban Magyarországon nem kapható. Ezeket külföldi webáruházakból rendeltem.



19. ábra Az első és az utolsó alaplap modelljei

A nyákok is hazai cégtől érkeztek, az első verziót a BME Nyák gyártotta, míg a másodikat az SoS PCB kft. Az első prototípus, mint ahogyan a 12. ábrán is látható, forrasztásgátló lakk és beültetési rajz nélkül rendeltem. Ez is a fejlesztési költségek csökkentését szolgálta. A második, utolsó verzió viszont lakkal és rajzzal együtt érkezett.

4.2.2.1. Fő mikrokontroller – ESP32

Ez mikrokontroller kielégíti a specifikációban megszabott feltételeket, jó választás az AGV feldolgozó egységének. A következőkben bemutatom miért. Az eszköz az Espressif cég által fejlesztett ESP32 családba tartozó ESP32D0WDQ6 köré épül (20. ábra baloldal). Ez egy két mikroprocesszorból álló chip. Ebből és további kiegészítő elemekből, mint például a nyák antenna és az EMI pajzs áll a WROOM 32, ami már egy önálló SoC (20. ábra középen). Ez a nyák van ráforrasztva egy ismét másik cég által készített nyárka, melynek a neve NodeMCU ESP32-S (20. ábra jobb oldalon). Ez a nyák tartalmazza többek között az USB-UART chipet, az 5V-3V3 szabályzót, microUSB csatlakozót, a „reset" és „boot" gombokat és a 2x19 db 2,54mm-es raszterű tüskesorokat.

20. ábra ESP32D0WDQ6 (balra) WROOM32 (középen) és ESP32-S (jobbra)

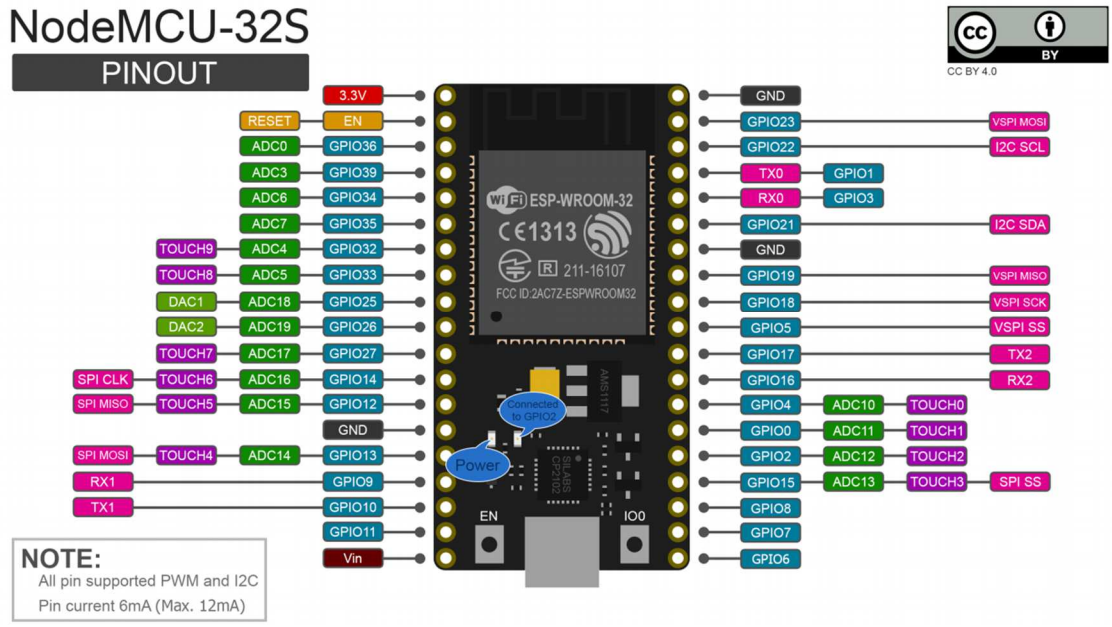
Ezen chip főbb jellemzői az ilyen kisméretű eszközök között szokatlanul magas órajel frekvencia, ami a 240MHz-et is elérheti, a szintén szokatlan kétmagos processzor, illetve a beépített WiFi és BLE. A két fizikai mag neve fő és mellék processzor. A fő processzor alapértelmezett feladata a felhasználói program futtatása, a mellék processzoré pedig a háttérfolyamatok futtatása, vagyis a WiFi és BLE motorháztető alatti folyamatainak kezelése és egyéb feladatok.

Az adattárolásra külső flash memória használható (ESP32D0WDQ6 chipen kívüli), jelen esetben már be van építve a WROOM-32 be, ez 4MB méretű. Az ESP32 belső memóriájának nagyjából fél MB-ja a felhasználó számára fenntartott. Felépítése a mellékletben megtalálható.

Az ESP32 szoftver keretrendszerét, az ESP-IDF-et már fejlesztik pár éve, az internetes közössége is kiépült. Sok információ található róla az interneten, ami a fejlesztés során nagy segítség volt. Ilyen segítségek a következőkben leít néhány fontos információ, melyeket úgy gondolom szükséges tudni, hogy bánni tudjunk az eszközzel. Ezeket

nekem tapasztalat útján sikerült megtanulnom, és bizony nem egyszer estem kétségbe, mikor azt hittem végleg kisütöttem az elektronikát.

A ki és bemenetei 3V3 szintűek és nem 5V toleránsak. Lábkiosztása 21. ábrán látható. Fontos kiemelni, hogy néhány láb kiemelt funkciókra fenntartott, felhasználó által nem használható. Ezek közül néhány a program tároló memóriára vannak kötve, néhány láb pedig csak be- vagy csak kimenetként használható. Ha az előbbiekre bármit kötünk, képtelen lesz a flash memóriáról bootolni az ESP, az utóbbiak pedig szimplán nem fognak az elvárásaink szerint működni.



21. ábra NodeMCU-32S lábkiosztása

Bootoláskor a chip rengeteg módban képes elindulni. A módokat kijelölt lábak induláskori potenciálra helyezésével lehet elérni. Ezek közül az egyik kiemelendő láb a GPIO12. Amennyiben ez a láb lebeg bootoláskor, egy belső ellenállás alacsony szintre húzza a lábat, és a chip a belső feszültségszabályzót 3V3 módba kapcsolja. Ez a belső szabályzó látja el feszültséggel többek között a flash memóriát. Azonban ha a láb magas szintre van húzva bootoláskor, mint ahogy ebben a projektben is, a szabályzó 1V8 szinten indul el, és a flash chip brownoutol, a program pedig nem indul el. Az erre a lábra kötött vonal nem húzható alacsonyra, mivel a kommunikációs buszként van használva, ahol az alapértelmezett érték a magas, ezért a vonal több helyen is magasra van húzva. A megoldás egy belső eFuse kiégetése. Ez nem jelent problémát, mivel a WROOM32 3V3 szintű flash memóriával van szerelve, soha nincs szükség az 1V8 előállítására. A megfelelő eFuse kiégetésével a chip bootoláskor figyelmen kívül hagyja a GPIO12 feszültségszintjét. Ezt a műveletet az ESP-IDF el mellékelt eFuse égető programmal érdemes elvégezni. [21]

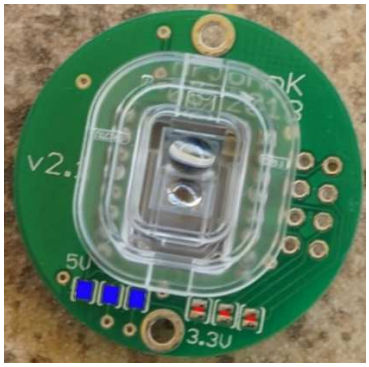
4.2.2.2. Elmozdulást mérő szenzor – ADNS-3080

A felhasznált motor léptetőmotor, amivel pontosan meg lehet adni az elmozdulás mértékét, de a kotyogás és megcsúszás miatt nagyobb távolságokon ez önmagában nem elég pontos. Az esetleges lépés kihagyások, vagy kerék megcsúszások korrigálhatatlan hibát jelentenének. Ezen okok miatt kénytelen voltam valamilyen szenzor használatára. Szintén az AlphaBot2 projekthez visszanyúlva, most is egérben használt elmozdulás szenzorokat választottam kedvező áruk és mechanikai szerelhetőségük miatt. Ez az alkatrész abba a csoportba tartozik, amely nem kapható Magyarországon.

A keresésem eredménye egy ADNS-3080 flow szenzor és egy ADNS-9800 elmozdulás szenzor. Mindkét modul nyákkal és optikával együtt érkezett. Az ADNS-3080 modulon, a 2. táblázat jobb oldalán látható módon optika segítségével meg lett növelve a fókusztávolság, így egy úgynevezett flow szenzort képez. Az ADNS-9800 modulon az optikai egerekből ismerős formájú optika látható. Ugyan azon az elven működik mind a kettő, miszerint az egymás után készített fotókból kiszámítják az előttük lévő felülethez képesti elmozdulást.

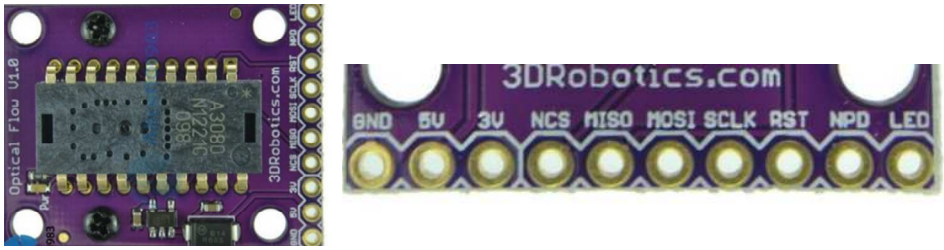
Az előbbi a módosított optikai rendszernek köszönhetően végtelenre állított fókusztávval rendelkezik. Ez azt jelenti, hogy a felület felett nagyobb távolságban elmozdítva is képes elmozdulást mérni (a flow sensor elnevezés erre utal). Az utóbbi azonban az optikai egerekből már jól ismert körülmények közt, csupán 2.5±0.1 [mm] távolságban működik megbízhatóan.

Mindkét modul egy Avago által gyártott IC köré épül, Ezek az ADNS-3800 és az ADNS-9800. A leírt jelölések nem az egész modulra vonatkoznak, csupán a magot képző IC-re, de minden fórumon csupán e szerint hivatkoznak rájuk.

ADNS-9800	ADNS-3080
[22]	
	

2. táblázat Két potenciális elmozdulás mérő szenzor

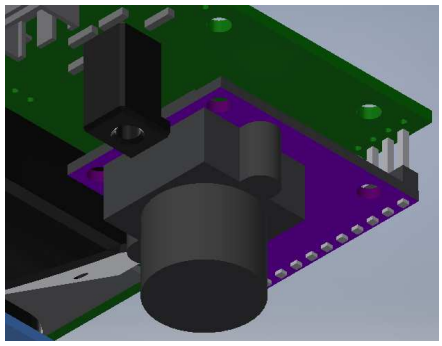
Az első prototípus a 9800-as modullal készült el, aminek a talajon kellett futnia. Ez bonyolult mechanikai felfüggesztést igényelt, aminek az eredménye nagy irányváltási kotyogás lett a pontatlan 3D nyomtatás miatt. Ez a felfüggesztés látható a 17. ábra bal oldalán. Mivel a nyomtatás minőségét nem lehetett növelni és bár maga az IC több generációval újabb és pontosabb, a kotyogásmentes felfüggesztés túl drága lett volna. Ekkor vált a 3080-as a végleges megoldássá. A nyák az IC-vel és annak lábkiosztása a 22. ábrán látható.



22. ábra ADNS-3080 modul és lábkiosztása

A nyák tartalmazza az IC-t, a csatlakozó tüskesort, néhány kiegészítő áramköri elemet és az optikát. Négy furat használható a modul rögzítésére. Az IC 3V3 feszültségszintű, de 5V toleráns. A kommunikáció SPI buszon lehetséges, és az IC regisztereinek írásával és olvasásával történik.

A modul először a 3D nyomtatott testen kapott helyet és vezetékekkel csatlakozott az ESP-re (13. ábra), de a szilárdság növelése és a vezetékezés elhagyása érdekében ezek is a nyáklemezre kerültek. Az előbbi a pontosság növelésében játszik szerepet, az utóbbi a könnyebb, olcsóbb után gyártást segíti elő. A végső megoldás a 23. ábrán látható.



23. ábra ADNS-3080 modul rögzítése

A rögzítését négy darab távtartóval és a nyák peremébe kapaszkodó kis műanyag darabbal oldottam meg. Ez utóbbira azért van szükség, mert az optika miatt nem fér anya az a melletti két távtartó menetére. Ez a modul nem rendelkezik önálló fényforrással, erről külön kellett gondoskodnom. A nyákba kapaszkodó klipsz mellett található négy darab led beforrasztására alkalmas foglalat. Az alaplapon található, LEDek vezérléséért felelős részegységet az ADNS-3080 adatlapjában található ajánlás alapján készítettem el.

A chip képes a lefotózott felület elmozdulásának kiszámolására. A számítás pontossága nagyban függ a felületi minőségtől. A felfedezett alaksajátosságok alapján képes számításokat végezni, így minél több ilyen van, annál pontosabb az eredmény. Képes továbbá saját megvilágításának vezérlésére, ám ehhez szükséges a kettő közé iktatni egy teljesítménynövelő MOSFET-et. A megvilágításra felhasznált LEDek piros színűek, mivel az adatlap alapján ezen a hullámhosszon a leg érzékenyebb a szenzor.

4.2.2.3. Kiegészítő mikrokontroller – ATmega328p

Az első prototípus még e nélkül készült el. A projektbe kerülésének oka hogy az ESP32-n futó szoftver sűrű megszakítása a motorok vezérlése érdekében nem volt elég hatékony, hatalmas overheadet okozott és a mozgás szaggatottá vált. A motor vezérlésének külső mikrokontrollerbe szervezése nagyban megkönnyítette a programozást is, illetve lehetőséget adott, hogy további funkciókat is ide szervezve felhasználhassam az Arduino-s könyvtárakat.

A chip az Atmel által gyártott mára már befutott megaAVR család egyik legsikeresebb tagja, nem utolsó sorban az Arduino elterjedésének köszönhetően. A 328 egy 8 bites Harvard architektúrájú RISC processzorral rendelkezik. Az általam használt 328p ennek egy módosított „picoPower" nevű változata.

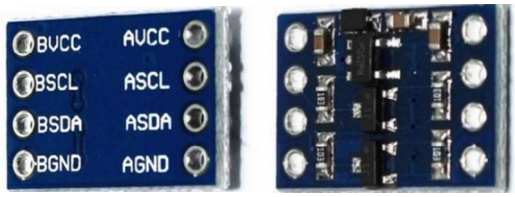
A fejlesztés során egyre több funkciót vett át, míg végül az RFID olvasás, a megfogó kar szervoi és egy WS2812B jelölésű LED vezérlése is erre a mikrokontrollerre került. Ahhoz hogy a mikrokontrollernek nagyobb számítási kapacitása legyen, a 16MHz órajelű üzemet választottam. Ehhez azonban nem elég a 3V3 tápfeszültség, a stabil működéshez 5V kell. Az ESP32 és 328p között I2C kommunikáció van. Mivel az ESP32 3V3 a 328p pedig így 5V feszültségszintű, a két egység közé I2C busz feszültségszintváltó modult helyeztem el.

A mikrokontrollert a NYÁK-on elhelyezett ICSP tüskesoron keresztül lehet programozni. Ehhez egy Arduino Uno-ból készítettem egy programozót. A tüskesorból irányzárasat választottam, így nem lehet rossz irányban csatlakoztatni a programozót, ezzel kizárva a rövidzár általi chip kiégetés lehetőségét. Az RFID olvasó SPI buszon csatlakozik, aminek a kivezetései egyeznek az ICPS kivezetéseivel, így ez a csatlakozó két funkciót lát el. A pinek nagyobbik része közös, de van, amely csak a programozás során, van, amely csak az RFID-vel folytatott kommunikáció során játszik szerepet.

Az IC kapcsolása és a nyák tervezése során az Arduino UNO dizájnjára hagyatkoztam. Az áramkör beindulásához minimális mennyiségű alkatrészt vettem ár a nyílt forrású projektből. Ez a kapcsolási rajz a következő forráson érhető el: [23]

4.2.2.4. I2C jelszintváltó

A korábban említett okok miatt szükség volt egy szintváltó képességre. Hogy kisméretű alkatrészeknek a házon belüli forrasztását minimalizáljam, úgy döntöttem különálló modulként szerelem az alaplapra. Ez a 24. ábrán látható. Bár a kapcsoláshoz nagyjából 20 SMD méretű forrasztás szükséges, ezzel a megoldással mindössze 8 db THD méretű forrasztás is elég.



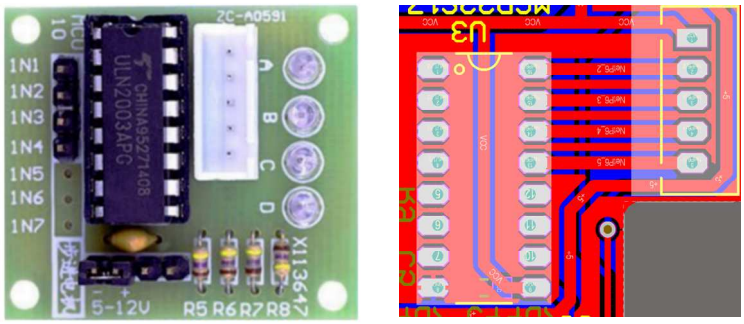
24. ábra I2C jelszintváltó modul

A jelzés nélküli modul, a webáruházban „LS-I2C-2" néven található meg. Tartalmaz 3V3-as 100mA maximális áramú feszültségszabályzót. Nem szabad a 3V3 oldalon táp feszültséget a modulra kötni, az csak áram szolgáltatásra való. Az A jelölésű oldal az 5V-os, a B jelölésű a 3V3.

A kapcsolás csatornánként egy MOSFET-ből és két ellenállásból áll. A kapcsolást a mellékletben részletezem. Úgy illesztettem a modulokat az alaplapra, hogy a feliratok felfele nézzenek az alkatrészek pedig lefele. Ez könnyebb hibakeresést tesz lehetővé, az által hogy a modul melletti teszt csatlakozók számára is feliratként használható.

4.2.2.5. Motor teljesítmény fokozat – ULN2003

A később bemutatott léptetőmotor a mikrokontroller által kiadott maximális áram többszörösét igényli. A kiadott jelet tehát erősíteni kell. Erre a feladatra kapható a motor mellé egy vezérlő, mely a kisáramú mikrokontroller jelből képezi a teljesítmény szintű jelet a motornak. A motor és a vezérlő csomagban kapható és csatlakozóval kapcsolhatók össze. Az ULN2003 modul egyszerű kapcsolással rendelkezik és az első prototípusban felfedezett nagy helyigénye miatt az áramkört később integráltam az alaplapra. A modul és az integrált verzió a 25. ábrán látható.



25. ábra ULN2003 léptetőmotor meghajtó modul és nyákon lévő megvalósítása

A NYÁK egy ULN2003A Darlington array köré épül, ami egymás után fűzött tranzisztorokat jelent. Az IC-n kívül még néhány passzív elemet és LED-eket tartalmaz, melyeket az integrálás során elhagytam. Hátránya más vezérlőkhöz képest hogy 4 GPIO-t foglal motoronként. Kapható olyan vezérlő is, amely I2C buszon csatlakozik, azonban többszöröse árban, így nem azt választottam. Az ULN először GPIO bővítőn keresztül csatlakozott az ESP32-re, majd a 328p bekerülésével elegendő GPIO állt rendelkezésre a bővítő kihagyásához is.

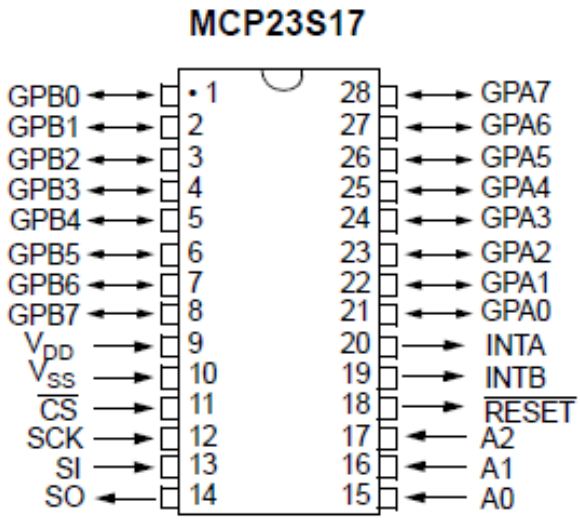
Az áramkör működése egyszerű, a kimenete az unipoláris léptetőmotor öt csatlakozója. Ezekből egy a motor közös kivezetése, ami állandó tápfeszültséget kap. A másik négy vezeték földre húzható a tranzisztorok segítségével, melyeket a négy GPIO vezérel. A mozgás során a tekercseket egymás után kell földre húzni. Erről bővebben a „Mozgás" alfejezetben olvashat a firmware-vel foglalkozó szekcióban.

Az integrált verzióban hely szűke és fogyasztás csökkentése érdekében lekerült minden felesleges alkatrész. Ezek a tüskesorok az ellenállások és a LEDek. SMD kialakításúra cseréltem az IC mellé ajánlott kondenzátort is. A fenti ábrán kiemelve látható az ULN2003 és a motor csatlakozó nyák lenyomata.

4.2.2.6. GPIO bővítő – MCP23S17

Ez az IC oldja meg a korlátozott számú ki és bemenet problémáját. Először a motorok vezérléséhez szükséges extra kimeneteket szolgáltatta, de megmaradt a vezérlés kiszervezése után is. Az utóbbi verzióban a gépjármű világítás és az ütközés érzékelő szenzorok vannak rá kötve. Ezek működéséről bővebben a firmware-el foglalkozó részben lehet olvasni.

SPI buszon csatlakozik a gazda mikrokontrollerhez és további 16 GPIO-t és egyéb funkciókat szolgáltat. Feszültségszintje 5V, így feszültségszintváltón keresztül csatlakozik az ESP-re. Típusszáma: MCP23S17. A lábkiosztása 26. ábrán látható.



26. ábra MCP23S17 lábkiosztása

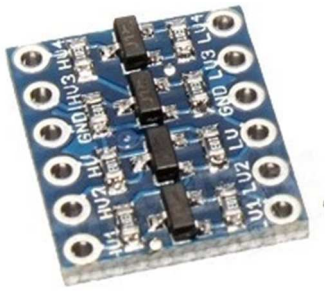
A bővítő jelszintje 5V, és egy feszültségszintváltón keresztül csatlakozik az ESP-hez. A 16 extra GPIO két részre van osztva, A és B portokra. A két porthoz tartozik két programozható interrupt láb, amelyekkel valamely a bemeneten történt változást érzékelhetünk. Ezen a funkcionalitáson alapszik az ütközés elkerülő szenzorok interrupt generálása. Van továbbá miden lábon bekapcsolható beépített felhúzó ellenállás is. Az IC összesen 150mA-t képes nyelni és szolgáltatni, lábanként pedig csupán 20mA-t. A „RESET" lábat az ESP32-re, az „ADDRESS" (A0, A1, A2) lábakat pedig mikrokapcsolókra kötöttem, így az IC címe állítható. A cím hét bit, 0|1|0|0|A2|A1|A0, ahol az Ax az adott cím állító láb logikai szintje. Alapértelmezettként a 0x20 címet használom.

Mint korábban említettem, képes a chip interrupt jel generálására is amennyiben azt bekonfiguráljuk a regisztereiben. Ehhez valamely lábat bemenetre kell állítani és hozzá rendelni egy interrupt módot. Ez lehet felfutó él, lefutó él vagy változás a jelszintben. Az interrupt jelet generáló lábakat visszakötöttem az ESP32-re egy feszültségszintváltón keresztül.

Az IC adatlapja hosszú és kevésbé jól strukturált. A fontosabb információk a fejlesztéshez a következők: Reset után BANK=0 módban indul el. A ki vagy bemenet állító regiszterei ekkor a 0x00 és 0x01 címen érhetők el. A regiszter bitjeinek értéke 1 a bemenet, 0 a kimenet esetén. Az IC minden regisztere a 0x00 és 0x01 kivételével 0x00 értékkel indul. A két kiemelt 0xFF állapotban indul. Miután kimenetre állítottuk a lábat, a 0x14 és 0x15 regiszterekkel állítható hogy a kimenet magas vagy alacsony szinten legyen, 1 a magas, 0 az alacsony szint.

4.2.2.7. Általános jelszintváltó

Mint az előző hasonló feladatot végző modul, ez is a különböző feszültségű modulok közti kommunikáció lebonyolításáért került kiválasztásra. Ez a modul több csatornás, de nem rendelkezik feszültség szabályzóval. A jelzés nélküli modul, a webáruházban „LS-BIDI-4" néven található meg. Kétirányú, négy csatornás. A „H" előjelű oldal a magasabb feszültségű oldal, az „L" jelölésű pedig az alacsonyabb. A 27. ábrán látható.



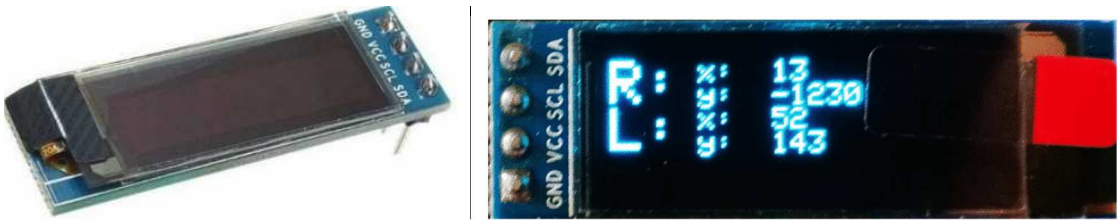
27. ábra Általános jelszintváltó modul

Mind az előző mind ez a modul hasonló tranzisztoros kapcsolással végzi el a feladatát, így a működésének leírása azonos, a mellékletben megtalálható.

Pontos feladata az SPI busz és egyéb vonalak, mint például az MCP és a 328p reset vonalainak a jelszintváltása, így lehetővé teszi az MCP23S17 és ESP32 közötti kommunikációt és az ATmega328p resetelését. Ezen keresztül fut még az MCP interrupt vonala is. Elhelyezési iránya itt is hasonló a korábbihoz. A feliratok felfele, míg az alkatrészek lefele néznek. Az SPI kommunikációért felelős modul mellé pedig tesztpontokat helyeztem el, hasonlóan az I2C modulhoz.

4.2.2.8. OLED kijelző – SSD1306

Az AGV fontos perifériája az OLED kijelző. A felbontása 128x32. I2C buszon csatlakozik az ESP32-re. A belső információk megjelenítése érdekében került a projektbe. Már a fejlesztés során is igen hasznosnak bizonyult a különböző belső állapotok megjelenítésében. Első sorban az ADNS-3080 szenzorok által mért adatokat jelenítettem meg rajta. Ez a 28. ábrán alul látható. A kijelzőt az SSD1306 típusú chip hajtja meg. Monokróm kijelző, nincs árnyalat. Kisebb szövegek, de akár grafikák megjelenítésére is alkalmas. A bolti képe a 28. ábra felső részén látható.



28. ábra I2C OLED kijelző és példa megjelenített információkra

A végső felhasználása az akkumulátor feszültségének megjelenítése, az AGV állapotának és hibaüzeneteinek kijelzése lett volna.

4.2.2.9. Akkumulátor és töltő áramköre – TP4056

A specifikációban leírtak alapján nem volt kérdés hogy a szükséges akkumulátor valamely Lítium technológián alapuló lesz. A két lehetőség a Lítium Ion és a Lítium Polimer. Az előbbi kisebb maximális áram leadására képes, de nagyobb a kapacitás/tömeg aránya. Az utóbbi a nagy áram igényű alkalmazásokban ajánlott. Mivel nem találtam olcsó, sorosan kapcsolt akkumulátorok töltésére való modult, úgy döntöttem párhuzamosan kapcsolt akkumulátorokat alkalmazok. Ez elegendő áramot jelentett még Lítium Ionos akkumulátor esetén is, de a feszültség szintet lekorlátozta mindösszesze 4V2-re. Végül egy igen kedvező áru 2600mAh kapacitású 18650-es akkumulátort választottam.

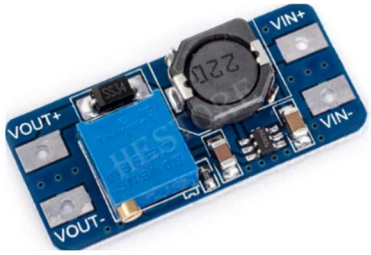
A töltő áramkör szerepe az akkumulátor kezelése. A töltési folyamatot szabályozni kell, különben az akkumulátor sérülhet, szélsőséges esetben ki is gyulladhat. Ennek a feladatnak az elvégzésére a TP4056 IC köré épített modult választottam. Az IC maga egy 1S akkumulátor töltő IC. A modul előnye a kivezetett csatlakozások könnyű hozzáférhetőségé és az USB csatlakozója, melyen keresztül lehet áramot szolgáltatni a modulnak. Fontos hogy a több verziója létezik. Az első prototípusban az első verziót választottam, ám az nem rendelkezik merülésvédelemmel. Ez után a későbbi modellt használtam, ami egy MOSFET-en keresztül csatlakoztatja az akkumulátort és kritikusan alacsony feszültségek esetén megszakítja a kapcsolatot. A túlmerítés hasonló problémákkal járhat, mint a túltöltés. A töltő modul a 29. ábrán látható.

29. ábra TP4056 alapú akkumulátor töltő modul

Maximális töltő árama 1A és jelzőfényekkel jelzi a töltési folyamat végeztét. A jelenlegi dizájn apró hibája hogy ez a visszajelzés nem működik jól. Az alapján dönti ez az IC, hogy végzett-e a töltéssel hogy az akkumulátor által felvett áram lecsökken. Mivel az AGV komponensei még az akkumulátor feltöltése után is jelentős áramot képesek felvenni, a töltő IC nem tudja mikor végzett.

4.2.2.10. Feszültség emelő

Az elsődleges energiaforrások az akkumulátorok, melyek legfeljebb 4V2 leadására képesek. Az 5V-os elektronika és a 6V szinten üzemelő szervo- és léptetőmotoroknak is kevés tehát az akkumulátor szintje. Ennek megoldása érdekében választottam ki ezt a feszültségszintnövelő modult. A jelzés nélküli nyák, a webáruházban „MT3608-2A" néven található meg. Maximális kimeneti árama 2A. Potenciométer segítségével állítható be a kívánt kimeneti feszültség. Képe a 30. ábrán látható.



30. ábra Feszültségszint növelő modul

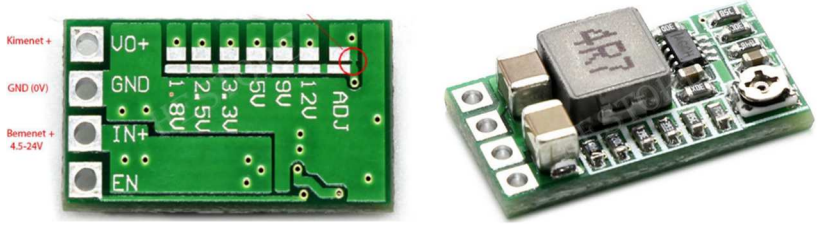
Feladata a nyákra érkező 3V6 és 4V2 közötti akkumulátor feszültség fokozása a feszültségszabályzók és aktuátorok számára. A beállított szint 6V. Minden fedélzeti egység ezen keresztül kapja a tálellátását közvetve vagy közvetlenül.

Az első prototípusban a 3V3 szabályzót e modul kihagyásával kötöttem az akkumulátorra, azonban az akkumulátor merülésével a szabályzónak nem volt elegendő feszültség többlete a 3V3 előállításához így alacsonyabb töltöttségen leállt. Ezt a hibát a második NYÁK verzióban javítottam.

4.2.2.11. Feszültségszabályzó

A panelen a korábbi 4V2 névleges és 6V mellett további vonalakra van szükség. Egészen pontosan 3V3 és 5V-ra. Ezeknek az előállításához kapcsoló üzemű feszültségszintcsökkentő modulokat választottam. A modulok nagyáramúak, magas hatásfokúak és lehetséges rajtuk fix kimeneti üzem beállítása.

Gyártói jelölés: „HW-613”. A szabályzó egy MP2315 típusú változtatható feszültségű feszültségszabályzó IC köré épül. A modul tartalmaz egy potenciométert és egy ellenállás sort is. A 31. ábrán, baloldalon látható módon leválasztva a potenciométert és megfelelő helyen forraszgömböt elhelyezve, fix feszültségű kimenetet kapunk.

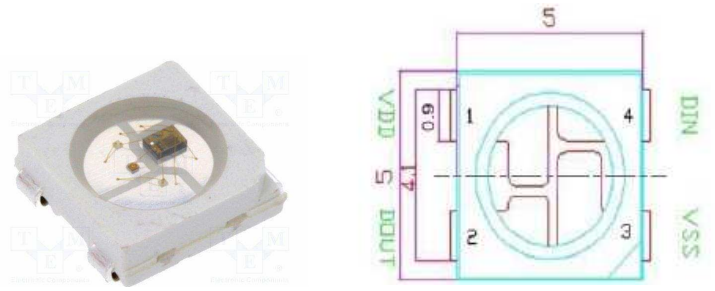


31. ábra Feszültség szabályzó modul

Ezen kívül a szabályzó IC ENABLE lába is ki van vezetve, melyet alacsony szintre húzva lekapcsolható a szabályzó, közel nullára csökkentve a fogyasztását is. Ezeket a lábakat jumperen keresztül kötöttem be az ESP32-be, így lebegve hagyva a belső felhúzó ellenállás bekapcsolva tartja a modult, de akár szoftveres lekapcsolásra is van így lehetőség.

4.2.2.12. RGB LED – WS2812B

A gyors fény visszajelzési igényeknek eleget téve kiválasztottam egy alkalmas LED IC-t. Az eszköz képes 16 millió színes megjelenítésre. Mostanában főleg az RGB világítások elterjedésének és egyedileg címezhető led szalagoknak köszönhetően vált népszerűvé. Típusa WS2812B. Ez egy olyan integráció, amely tartalmaz egy kis számítási képességű IC-t és három alapszínből álló LED-et. Az alábbi 32. ábrán látható a chip és műszaki rajza.



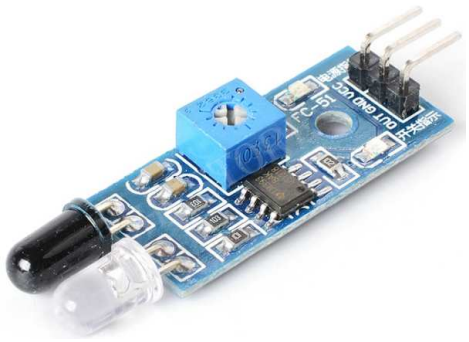
32. ábra WS2812B LED chip és befoglaló méretei

A jelenlegi projektben az alaplapon egy darab ilyen chip található. A színekkel és villogási mintákkal gyors visszajelzést biztosít az AGV állapotáról, a nélkül hogy a kevésbé látható OLED kijelzőre kellene nézni. A minták jelentéséről az „Állapotjelző fények" részben olvashat.

4.2.2.13. Infravörös akadályszenzor

Az előző AGV alapvető funkciója volt az akadály előtti megállás. Ugyan azon az elven alapuló eszközzel oldottam meg jelen fejlesztés során is az érzékelést. Infravörös tartományban dolgozó fényforrás és detektorban végződő modulokat választottam. A nyákon található további elektronika felel az érzékelésért, és az analóg rendszerből egy digitális kimenetet kapunk. A digitális kimenet billenőpontját a rajta található potenciométerrel lehet állítani. Ezzel lényegében az érzékelési távolságot lehet állítani.

Az átbillenés pillanatában érzékeli az MCP IC a változást és interrupt jelet generál az ESP32 számára. Az ESP32 leolvassa az MCP állapotát, amiből kiderül, hogy mely szenzor jelzett. Az ütközés előre jelzésen felül ilyen modul felel a megfogó karban a munkadarab meglétének ellenőrzéséért. A modul a 33. ábrán látható.

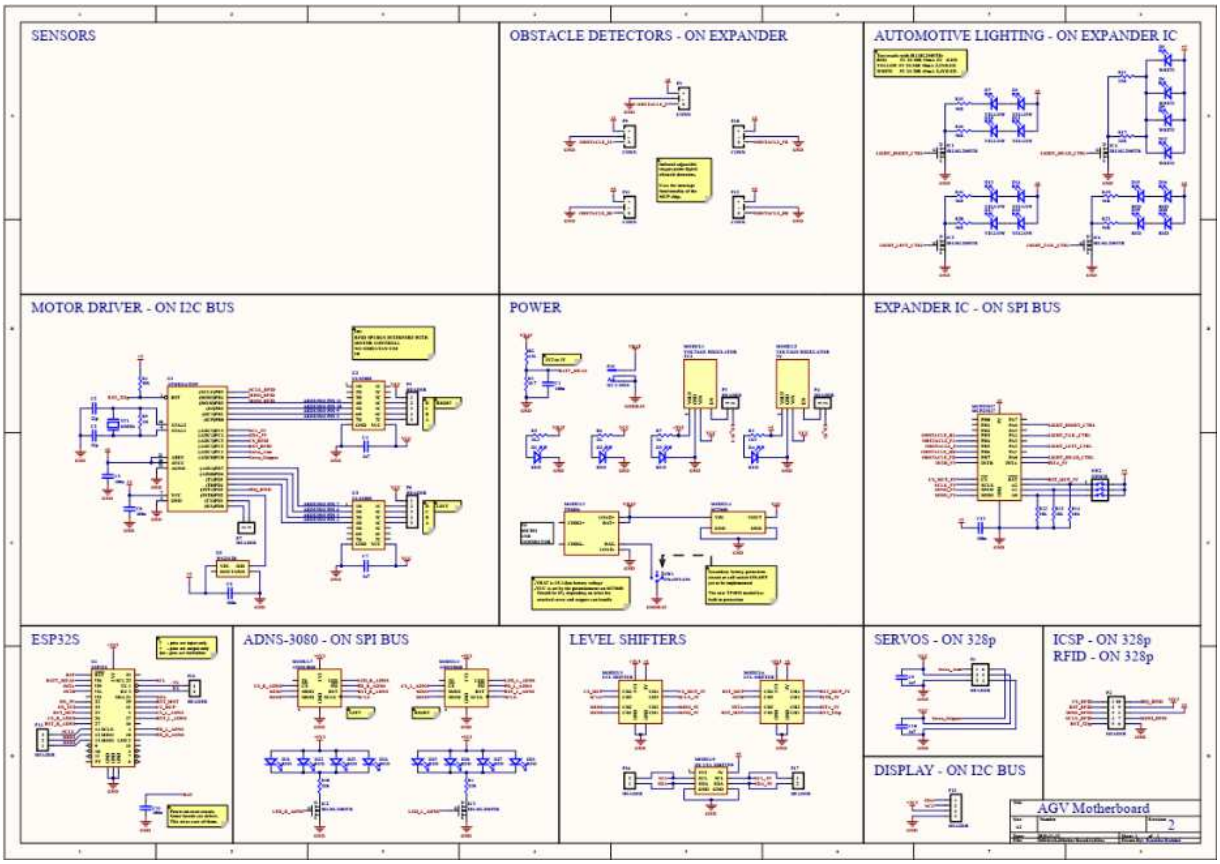


33. ábra IR akadály érzékelő modul

4.2.2.14. Kapcsolási rajz és nyák

Az első prototípus még egy próbalemezen összeforrasztott kapcsolással rendelkezett, és kapcsolási rajz sem készült hozzá. Később az alkatrészek számának növekedésével hatványozottan nőtt a kapcsolatok száma is, kényszerű volt egy program választása.

A feladatnak is része a jó dokumentálás és az összetetteb nyák tervezésének első lépése is a sematikus tervrajz elkészítése. A kapcsolási rajz a már korábban bemutatott Altium nevű programban készült a nyák tervvel együtt. A kapcsolási rajz egyetlen A3 méretű lapon elfért, helyet is hagyva esetleges későbbi bővítéseknek. A kapcsolás nagyalakú PDF-ben megtalálható a külső mellékletek között. A 34. ábra csupán illusztráció, ekkorában nem olvasható a szöveg.



34. ábra Alaplap kapcsolási rajza

A kapcsolási rajzon is funkcionális blokkok szerint csoportosítottam az alkatrészeket. Ezek a vonalakkal elválasztott négyzetek. Ahol értelme volt jelöltem, hogy mely adat buszon található az adott komponens. Ez a felirat a négyzet bal felső sarkában.

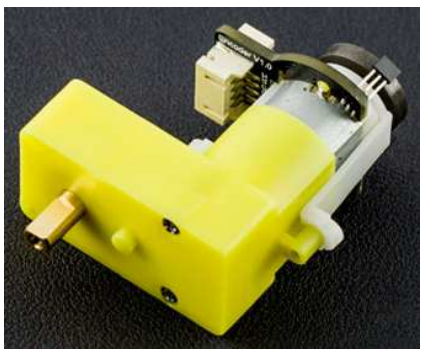
Lehetőség szerint nagyméretű alkatrészeket használtam, hogy a forrasztás könnyű és gyors legyen, ezzel is csökkentve a költségeket. Ez azt eredményezte, hogy az alkatrészek többsége THD és csupán egy kis része SMD. Ez később befolyásolta a kapcsolások kialakítását is. A THD alkatrészek mindkét oldalon igényelnek helyet a nyákon, míg egy SMD alkatrész túloldalán lehet vezeték.

Az alkatrészek nyákra kerülésének okait leírtam a bemutatásukkor. A kapcsolatok kialakításakor tett megfontolások azonban most következnek:

- Az ESP32 lábainál figyelembe vettem a nem használható lábakat, SPI és I2C buszok csatlakoztatását a dedikált lábakra kötöttem, a GPIO lábakat pedig a könnyű vezetékezés megoldása szerint választottam ki. Az ESP rendelkezik belső multiplexerrel, ami képes bármely belső lábat bármely külsőre kötni, de ennek limitált a frekvenciája.
- A 328p és az ULN közötti kapcsolatot úgy alakítottam ki, hogy egy motor egy portról kapjon jelet. Ennek az az oka, hogy az Arduino kód a lábak jelszintváltását sok soros kódból oldja meg. Ez jelen esetben nem megengedhető, ezért a gyors motor forgás érdekében port manipulációt kellett használnom. Ez egy alacsonyabb szintű programozás, mindössze egy CPU ciklus alatt képes megváltoztatni a kimeneti szinteket akár az egész porton. A kiválasztott portok az B és D.
- A szervo motorok azért kerültek a 328p-re, mert nem volt magas szintű kód az ESP32-re megírva, amivel könnyen tudtam volna vezérelni őket. Az rendelkezésre állt viszont az Arduino környezetben. Ugyan ez az oka az RFID olvasó elhelyezésének is.
- Az ütközés érzékelők és világítás vezérlés a GPIO bővítőre került, mivel ezek nem igényelnek nagy frekvenciás vezérlést és sok lábat igényelnek.

4.2.3. MEGHAJTÁS

Az egyik legfontosabb alkatrész a meghajtásért felelős motor. Az AlphaBot2 buktatóiból tanulva valamilyen beépített szabályozhatósággal rendelkező motort kerestem. Az elérhetőségen és költségvetésen alapuló keresés eredménye az alábbi két motor. Hajtóműves DC-motor enkóderrel, illetve hajtóműves léptetőmotor. Ezek a 3. táblázatban láthatók.

DC motor	Léptető motor
	

3. táblázat Potenciális meghajtó motorok

A választás végül a léptetőmotorra esett, két fő indok: harmadannyiba kerül, illetve a programozása is egyszerűbb, nem igényel DC motor szabályozó kört, amivel sok

időt nyertem a firmware írásában. A motor adatai megtalálhatók a szöveges mellékletben.

Az ajánlott üzemmód a fél lépéses, első teszt alkalmával ennek ellenére egész lépéses üzemmódban próbáltam ki. Azt tapasztaltam, hogy megfelelő nyomatékot tud le adni úgy is, a vezérlés pedig egyszerűsödik. A hátrány, ami miatt maradtam az ajánlott fél lépéses üzemmódban, hogy a maximum sebesség csökken, kisebb frekvencián forog el a mágneses mező a rotorhoz képest. Mivel a motor lassú, igyekeztem minden apró előnyt kihasználni a gyorsítás érdekében.

A meghajtáshoz megfelelő kereket is kerestem. A kritériumok a keskeny futófelület, megfelelő átmérő és a kis tömeg voltak. Végül egy modell boltból sikerült szervo kereket szereznem. A kerék a 35. ábrán látható [24]. A motor tengelye és a választott kerék közé 3D nyomtatott adaptert terveztem.

35. ábra Választott kerék

A hajtómű áttételével, motor paramétereivel és a kerék átmérőjével már számolható az egy lépés alatt megtett út. A számítás a következő:

Adatok:

$$(\text{hajtómű áttétel})\ \mathrm{h}=63.68395\ [\text{-}] \tag{4.1}$$

$$\left(\frac{lépés}{rotor\ átfordulás}\right) s = 64\ [-] \tag{4.2}$$

$$(kerék\ átmérő)\ d = 60\ [\mathrm{mm}] \tag{4.3}$$

Számolás:

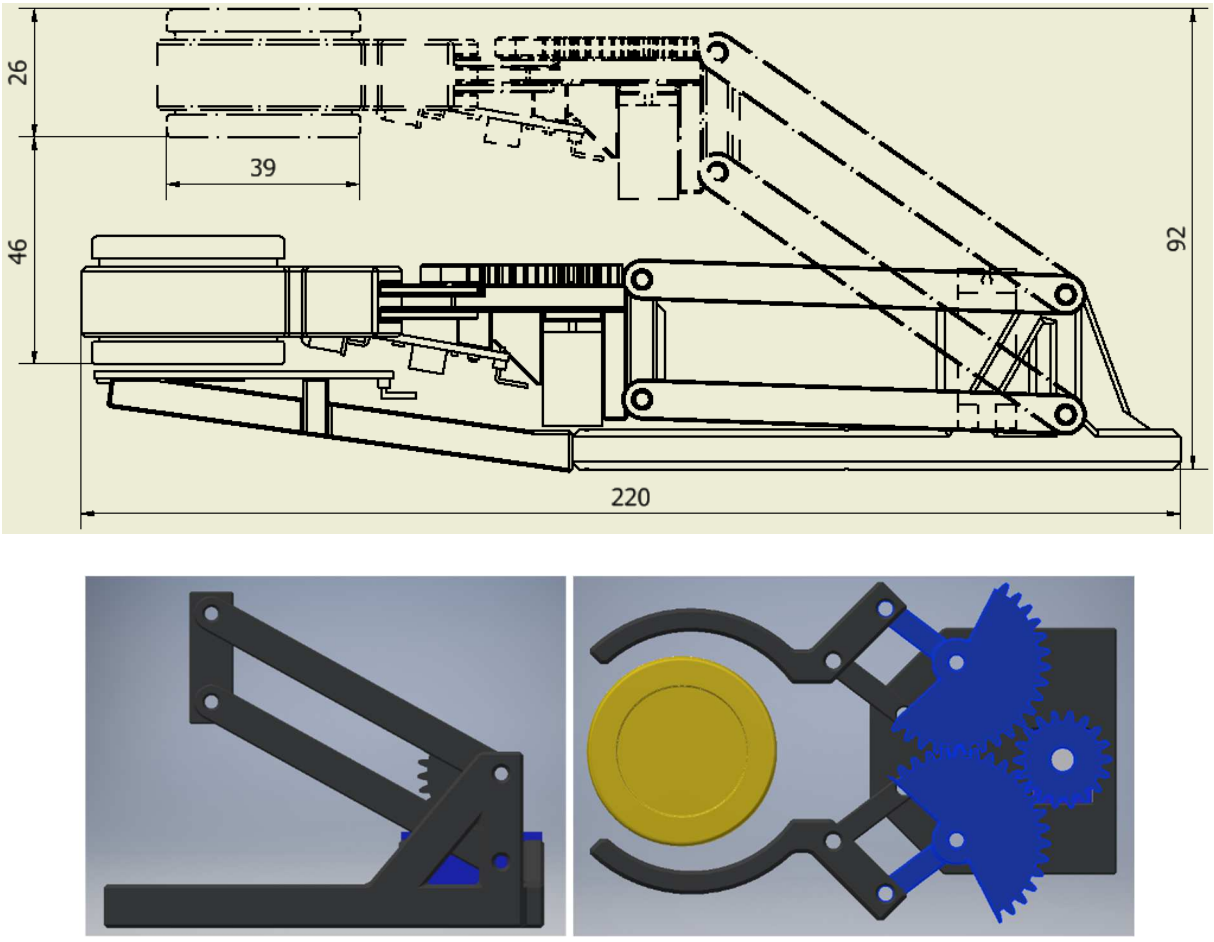
$$\left(\frac{lépés}{csonk\ átfordulás}\right) S = \mathrm{s} * \mathrm{h} = \ 4075.77\ [-] \tag{4.4}$$

$$\left(\frac{elmozdulás}{csonk\ átfordulás}\right)\ k = \mathrm{d} * \pi = 188.5\ [\mathrm{mm}] \tag{4.5}$$

$$\left(\frac{elmozdulás}{lépés}\right) D = \frac{\mathrm{k}}{\mathrm{S}} = 0.04625\ [mm] \tag{4.6}$$

4.2.4. MEGFOGÓ KAR

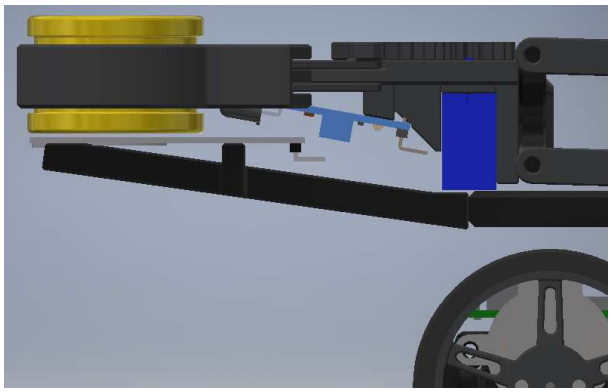
A kar feladata a munkadarab megfogása és tartása szállítás közben. Képesnek kell lennie kompenzálni a magasságkülönbséget a fészkek és az alépítmény között. A kar feladata továbbá az RFID címke olvasása is. Feladatai alapján jól elkülöníthető egységet képez. Ahhoz hogy a fészek fölé tudja emelni és meg tudja fogni munkadarabot, az emelő és megfogó karok is négycsuklós mechanizmuson alapulnak. A kialakítás nagyban alapszik egy internetes oldalon talált megvalósításra [25]. A szemben lévő karok egyenlő hosszúságúak, így emelkedés közben a szemközti részek párhuzamosak maradnak. A szerkezet a 36. ábrán látható. A karok hosszát modellező programban kísérleti úton határoztam meg, mivel megvolt a fészek pontos modellje.



36. ábra Megfogó kar befoglaló méretei és szerkezeti ábrája

Az alsó képek az első prototípust ábrázolják, és a mechanizmust mutatják be, mely nem változott. A felső ábrán a legújabb modell látható a befogó méretekkel, kiemelési magassággal és a munkadarab méreteiv el. A kezdeti kialakítás bővült néhány csavar rögzítő furattal és a 3D nyomtatás megkönnyítő alaksajátosságokkal. A szerelés megkönnyítése érdekében is átalakítottam néhány részt a második verzióra.

Üzemszerű működés esetén a kar alacsonyan van mozgás közben. Fészekre álláskor a kar felemelkedik és amennyiben anyagot vesz fel, a megfogó kinyílik. Az AGV rááll a munkadarabra és a megfogó bezárul. Ez után lehátrál a fészek fölül és leereszkedik a kar. A kar alsó állásában tálcaként is szolgáló RFID olvasó található. Erre eresztve a munkadarabot, leolvasható annak a kódja is. Ez a 37. ábrán látható.



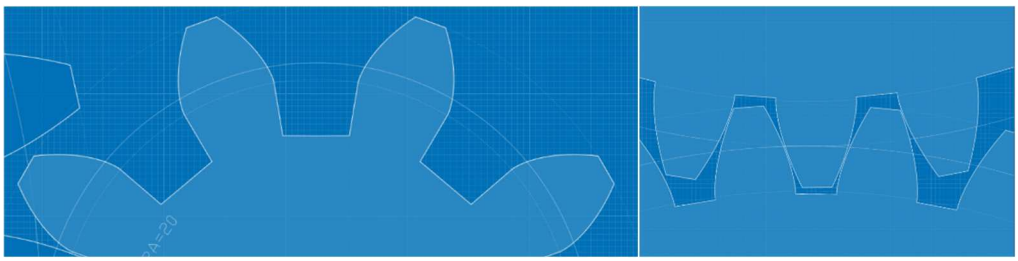
37. ábra RFID olvasó elhelyezése

Az első verzióban még a megfogón volt az RFID olvasó is, azonban a szerkezet tömegének és magasságának csökkentése érdekében az olvasó alulra került. Ezzel elveszett a megfogás pillanatában történő leolvasás lehetősége, azonban sokkal biztosabb lett a szállítás közbeni megfogás és kisebb terhelés kerül a kar emelő szervora és szerkezetre. Ugyanekkor kibővült a megfogó kar egy akadályérzékelő szenzorral. Ez jelzi a munkadarabra álláskor, hogy van-e ott munkadarab.

4.2.4.1. Szervok és fogaskerekek

A kart két szervo motor hajtja, ezek típusa MG90S. Elméletileg 180° elfordulásra képesek, gyakorlatilag azonban az utolsó 10° mind két irányban használhatatlan, mivel itt telítődnek a motorok belső szenzorai és oszcillálnak a parancsba adott szöghelyzet körül.

A szükséges nyomatékról fogaskerék áttételek gondoskodnak. Az emelő kar áttétele 1:4, a megfogó karé 1:2. A fogaskerék profilokat egy online fogaskerék generátorral készítettem [26]. A profilok a 38. ábrán láthatók.

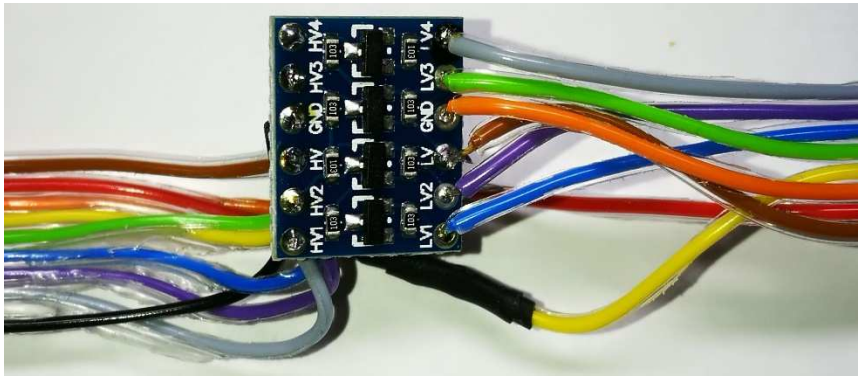


38. ábra Fogaskerekek profiljai

Az áttételeket aszerint választottam, hogy mekkora nyomatékra és mozgási szögre van szüksége az adott rendszernek. Az emelő kar nagyobb terhelést kap, mint a megfogó kar, így itt nagyobb az áttétel. Ez után a négycsuklós mechanizmusok karjainak hosszát úgy igazítottam, hogy a szükséges minimum utat tegyék meg.

4.2.4.2. RFID olvasó – MFRC522

Az RFID tagek olvasására használt modul RFID-RC522 típusú. Az alapját az NXP által gyártott MFRC522 képezi. Ez SPI buszon csatlakozik a mikrokontrollerhez. A kivezetett pinek és funkciójuk a mellékletben lévő táblázatban látható. A maximási adatátviteli sebesség 10Mbit/s, így a maximális órajel 10MHz. A chip és modul mind 3.3V-on működnek és nem 5V toleránsak. A feszültségszint váltást nem tehettem a NYÁK-ra, mivel befolyásolta volna az ICSP működését. Ezt a modul csatlakozójára helyezett kisméretű nyákon lévő feszültségszintváltóval oldottam meg. Ezt a típust használtam az alaplapon is szintváltásra. A nyák a 39. ábrán látható.



39. ábra RFID modul jelszintváltója

Mivel ebből ez az első prototípus, még szabadon vezetékezett módon készült el, de a későbbiekben ez is saját nyáktervet és nyákot kap a könnyebb és biztosabb szerelhetőség érdekében. Szükséges lenne e mellett egyes egyirányú csatornák szintváltása feszültség osztóval, de ez sem könnyen kivitelezhető levegőben lógó alkatrészekkel. Ez is majd csak a készülő nyákon kap helyet.

4.3. Firmware írása

A hardverek letisztázása után a specifikáció szerinti képességekkel kell ellátni azokat. Az alábbi részben a különböző komponensek programozásának folyamatát, illetve a megértéshez minimálisan szükséges ismereteket írtam le. Az összetettebb koncepciók és kisebb rendszer leírások is ebben az alfejezetben vannak. Forráskód nem szerepel a dolgozatban, nagy mérete miatt a szöveges melléklet sem tartalmazza. A teljes forráskód a külső mellékletben (DVD) megtalálható.

Ismét kiemelném azt az oldalt, amely bármiféle programozást lehetővé tett, dokumentációként szolgálva számomra korábban teljesen ismeretlen platformhoz. Ez az ESP-IDF Programming Guide oldala [16]. A szükséges pontokon hivatkozok a dokumentáció egyes részeire.

A fejezet végén található a 328p-re készült kódok bemutatása. Ezt Arduino IDE-ben írtam, és nagyrészt közismert könyvtárakat használtam fel.

4.3.1. ESP32 SZOFTVERES OLDALRÓL

Korábban röviden bemutattam az ESP32-t, de nem beszéltem a szoftveres felépítéséről. A dolgozat egységességének megőrzése miatt ketté választottam. Itt tehát az ESP32 és a rajta futó operációs rendszerről lesz szó, kiegészítve néhány alapvető fogalommal.

Az ESP-IDF egy FreeRTOS nevű, mikrokontrollerekre készült operációs rendszerre épül. Egészen pontosan a 8.2.0 verziójú Xtensa típusú processzorra portolt, kompatibilitás érdekében jelentősen módosított szoftvercsomagra. A módisítás a dupla CPU miatt volt szükséges [27].

FreeRTOS egy az Epsressiftől független projekt. Az Espressif azért vette alapul a termékükhöz, mert a hardver már elég bonyolult ahhoz, hogy megfelelő szoftveres támogatás nélkül senkinek ne érje meg ezt a platformot választani. Ennek a minimalista operációs rendszernek köszönhetően nem kell foglalkoznunk a háttérben futó alapvető folyamatok kezelésével, mint a vezeték nélküli kommunikáció chipjeinek kezelése, vagy olyan magasabb szintű feladatok, mint a feladat ütemezés, időzítés vagy memória kezelés [28].

Ezt kiegészíti még a cég által nyújtott API, ami többek között a WiFi protokoll teljes szoftver csomagját, a Bluetooth kommunikációhoz tartozó RF IC-vel történő kommunikációt és még sok mást tartalmaz. Ezek mind szükségesek, hogy a mikrokontroller a hardvertől kissé eltávolodva, magasabb szintű utasításokkal legyen programozható. Képes továbbá magas szintű debug információkat szolgáltatni egy esetleges crash során. Ezt a funkcionalitást sajnos nem sikerült teljes mértékben kihasználnom, mivel ehhez dedikált hardverre lett volna szükségem, mely csupán a fejlesztés végén érkezett meg hozzám Kínából [29].

Kiemelt fontosságú funkció a „task”, „handler” és „timer”, melyek a FreeRTOS-ből származnak. A legtöbb funkció az ESP-IDF-ben is elérhető. Egy YouTube-on található öt részes videó sorozat jó rálátást ad az RTOS-ek képességeire [30].

Az ESP-IDF telepítése a mellékletben található, de a felhasználásáról röviden itt írok. A három fő feladatot magában foglaló python program az „idf.py” nevet viseli. A három feladat a projekt konfigurálása, erre az „idf.py menuconfig” parancs használható. A projekt buildelése, erre az „idf.py build” parancs használható. Végül pedig az elkészült firmware flashelése, erre az „idf.py flash” parancs való. Utóbbinak legtöbb esetben szüksége van még a COM port paraméterre. Ezen kívül más feladatokra is képes az idf.py. Például soros monitorként is használható. Teljes dokumentáció a korábban említett weblapon érhető el [31].

4.3.2. KOMMUNIKÁCIÓ

Mind a belső, vagyis az AGV komponensei közötti 328p-ESP, ESP-MCP stb., mind a külső, Bluetooth kommunikáció ide tartozik. Az egyes komponensek a következőkben bemutatott egységek segítségével tudnak egymással információt megosztani. A felbontás a használt protokoll szerint történt. Nem részletezem a szabványaikat, mert nem a dolgozat témája és sok jó leírás található az interneten róluk. Ebben a részben csak az ESP32-höz köthető komponenseket írom le.

4.3.2.1. Külső kommunikáció

4.3.2.1.1 WiFi

Az ESP32 rendelkezik beépített 2.4GHz frekvenciájú WiFi adóvevővel. Képes állomás és kliens módban is futni. Nem jutottam el a fejlesztésben e kommunikációs csatorna kiépítéséig, de a Bluetooth szerencsére elegendő volt.

4.3.2.1.2 BLE

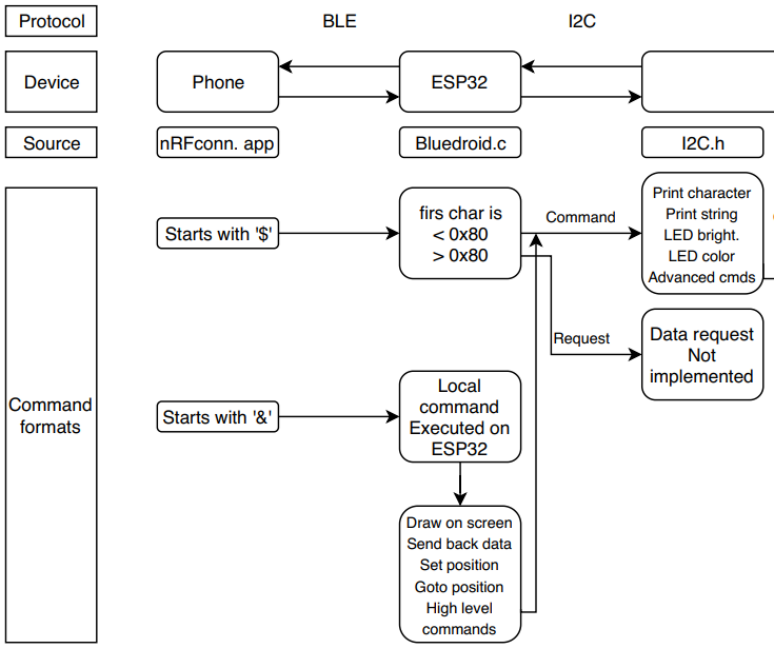
A szabvány fogalmainak és alapvető lehetőségeinek megértéséhez számomra ismét egy YouTube-os lejátszási lista adta a legnagyobb segítséget [32]. Továbbá a fejlesztés és debugolás alatt is nagy segítséget nyújtott számomra az nRF Connect for Mobile nevű Androidos alkalmazás [33]. A komponens teljes megértéséhez a Bluetooth 4.0 vagy másnéven a BLE szabvány alapfogalmainak ismerete szükséges. A következőkben röviden bemutatom a szabványt.

A BLE a jól ismert technológia a Bluetooth Classic leszármazottja, mellyel azonban nem kompatibilis. A szabványban leírt kommunikációs struktúrának köszönhetően a fogyasztása is jóval alacsonyabb elődjénél. Ezt főleg annak köszönheti, hogy míg elődje folyamatos adatkapcsolatot tartott fent, és nagy mennyiségű adatot továbbított, a BLE időszakos kapcsolatokat tart fenn, kisebb csomagokban kommunikál. Ebből az

is származik, hogy a BLE nem alkalmas nagyobb adatok mozgatására, inkább kisebb szenzor adatok időszakos leolvasására.

BLE protokollban az eszközöknek szerepkörei vannak. Ezek a szerepek a „Broadcaster”, „Observer”, „Peripherial” és „Central”. Minden eszköz képes bármennyi szerepet játszani egyidőben. A Broadcaster folyamatosan tesz közzé adatcsomagokat, függetlenül attól, hogy van-e vevő. Az Observer a nélkül veszi fel az adatcsomagokat, hogy jelezné jelenlétét. A Peripherial hírdeti a jenelétét és hogy készen áll a kapcsolódásra. Amennyiben egy Central is jelen van, kapcsolódási kérvényt küldhet a Peripherialnak.

Az alábbiakban tárgyalt kód a forrásban a „Bluedroid” komponensben található meg. A jelenlegi megvalósítás alapján az AGV létrehoz egy szolgáltatás listát, melyben van adat bevételére és adat kijelzésére való szolgáltatás is. Van egy további, amely mindkettőre alkalmas párhuzamosan. Ez gyakorlatilag commandlineként működik. Az ide beküldött szöveges parancsokat az AGV értelmezi, és végre hajtja. A parancsok felépítése a 40. ábrán látható. Az ábra a kommunikáció egy részét mutatja, a teljes blokkdiagram megtalálható a mellékletben, CommandChain.pdf névvel.



40. ábra Parancsok felépítését ábrázoló blokkdiagramm részlet

A jelenlegi állapot szerint amennyiben ’$’ jellek kezdődő parancsot küldünk, azt vagy tovább küldi a 328p számára, vagy adatot kér be attól. Az ábrán látható felirattól eltérően az adat igénylés implementálva van, de nem működik több bájtot igénylő lekérésekkel. A ’$’ parancsokat főleg az I2C busz és az általam megtervezett parancs formátumok tesztelésére használtam. Amennyiben a parancs ’&’ jellel kezdődik, az

ESP belső, magasabb szintű függvényeket hív meg. Ezzel tesztelhettem ezeknek a működését. Ugyanezen függvényeket hívnák az egyéb belső komponensek üzemszerű működés közben. Egy ilyen magas szintű függvény a vészstop is.

A BLE protokoll támogatja a Mesh-t is, ami egy kliensek által fenntartott elosztott hálózat. Nincs központi szerver, közvetlen kapcsolat van minden kliens között. Ez rendkívül hasznos lehet egy raj kiépítése esetén.

4.3.2.2. Belső kommunikáció

SPI buszon csatlakoztattam az MCP23S17-t, illetve ezen a buszon vannak az elmozdulás mérő ADNS-3080 szenzorok is. I2C buszon van az OLED és a 328p. Mindegyikük szerves részét képezi az AGV képességeiknek, így nyilvánvaló a fontosságuk.

4.3.2.2.1 I2C

Az I2C protokollt hardveresen támogatja az ESP32, és alapvető szoftverek is rendelkezésre állnak. Az API dokumentációk átolvasása után könnyen használható függvényeket kapunk [34].

Az alábbiakban részletezett kód a forráskódban „I2C" nevű komponensként szerepel. A kommunikációhoz először egy i2c_config_t struktúrát feltöltünk a kívánt beállításokkal, majd meghívunk két függvényt. Ez a folyamat látható az init_I2C_bus() függvényben. A kommunikáció 1MHz-re van korlátozva, ez talán későbbi ESP-IDF verziókban nagyobb lesz, de egy másik, sokkal alacsonyabb korlátozás ered a 328p hardver felépítéséből. A mikrokontroller legfeljebb 500kHz óra jelet képes feldolgozni. Mivel a buszt csak egy féle képpen lehet konfigurálni és nem praktikus menet közben frekvenciát állítani, a leglassabb buszon lévő eszköz fogja meghatározni az óra jel frekvenciáját.

Megjegyezném, hogy az ESP32 hardveresen képes több I2C busz megvalósítására, és szükség esetén külön választható a kijelző és a 328p busza. A problémát azt jelentheti, ha nagyobb kijelző frissítési frekvenciát szeretnénk elérni. A kijelző egy képkockája 512 bájt, amit a 328p által alacsonyra kényszerített órajel frekvencia miatt még tovább tart kiléptetni a buszra. Ez alatt azonban nem tudunk a motort vezérlő 328p-nek parancsot adni. Mivel nekem nem volt szükségem sok kijelző frissítésre és az időm is szűkös volt, nem bonyolítottam a rendszert tovább ezzel a lépéssel, de a lehetőség adott további fejlesztésekre.

A kommunikáció folyamata jól látható az i2c_write_address_register_data() és társa, az i2c_read_address_register_data() függvényekben. Fel kell tölteni egy i2c_cmd_handle_t struktúrát, mely tartalmazni fogja a kommunikáció minden elemét. Ilyen elemek a start bit, a cím bájt és a különböző adat bájtok. Az első függvény hozzá adja a start bitet a struktúrához. Ez után jön az eszköz címe és így tovább. Miután feltöltöttük a struktúrát, elindítjuk a bitek kiléptetését. Végül töröljük a struktúrát.

A későbbiekben bemutatott magasabb szintű függvények, melyek „írj a kijelzőre szöveget” vagy „motor V-stop” formátumúak, ezekre az alapvető függvényekre alapoznak.

4.3.2.2.2 SPI

Ez a protokoll és a hozzá tartozó szoftveres megvalósítás már jóval összetettebb, mint a korábbi. Itt is mester módban használjuk majd a buszt. Elérhető három vezérlő is, ebből a HSPI-t fogjuk használni. A maradék kettőből használható még egy. A harmadik működése még az IDF-ben nem megoldott. Nem használjuk az opcionális quadwp és quadhd vonalakat [35].

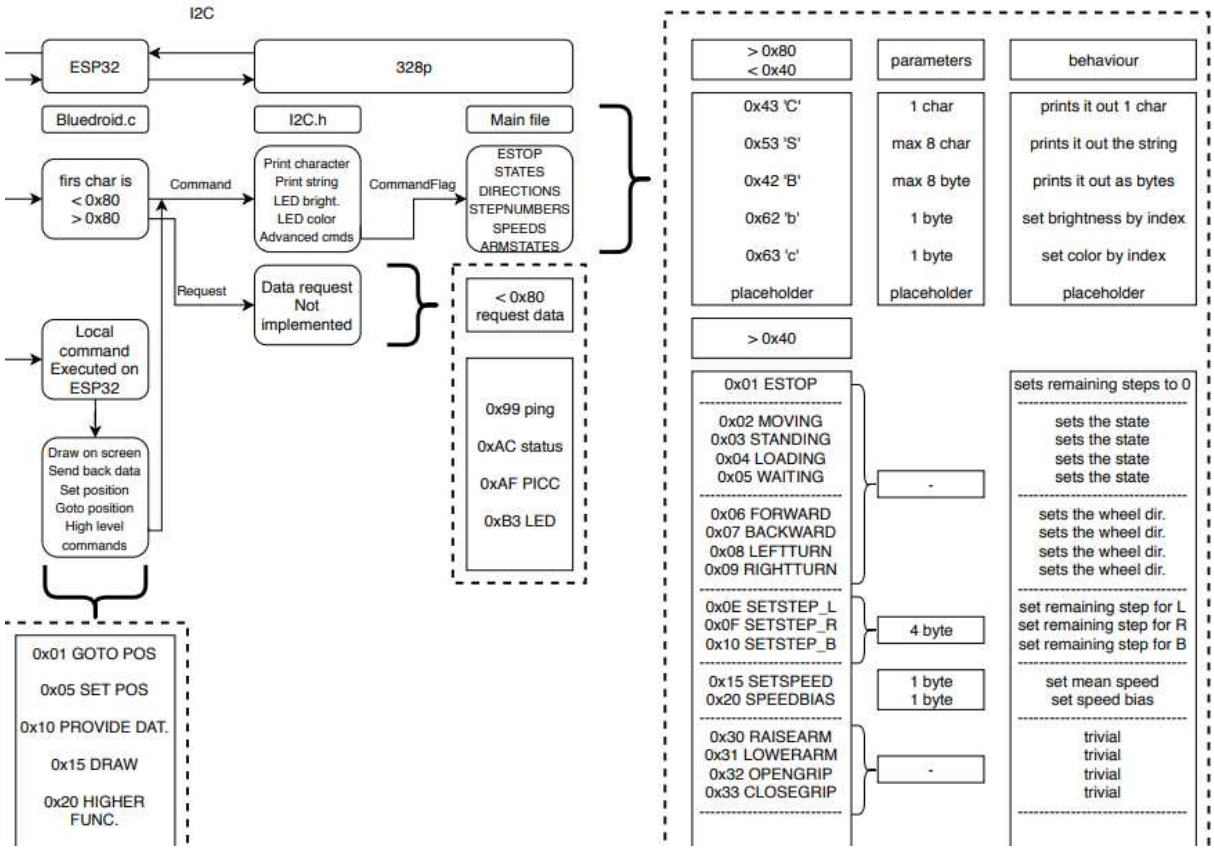
Az itt részletezett kód a forrásban az „SPI” komponensként található meg. A buszon folyó kommunikáció előkészítése több lépéses. Először elő kell készíteni a buszt, ezt az init_SPI_bus() függvényben lehet látni. A menete hasonló a korábban ismertetett I2C busz előkészítéshez. Egy további lépés jelen esetben hogy minden egyes a buszon lévő eszközt is inicializálni kell használat előtt. Az ebben a függvényben elkészített konfiguráció tartalmazza majd az eszközspecifikus beállításokat, mint például órajel, SS vonal, bemeneti késleltetés stb. Ez elkészít egy spi_device_handle_t struktúrát, mely segítségével az adott eszközzel tudunk a későbbiekben kommunikálni. Ebből látszik, hogy ezen a buszon nem korlátoz az egyes eszközök sebessége globálisan.

Az írás és olvasás műveletek egymáshoz képesti időbeni lejátszódása szerint két módot különböztetünk meg. Egyszerre írás és olvasás a full duplex mód. Amennyiben egyszerre vagy csak írás vagy csak olvasás zajlik, half duplex módnak nevezzük. A szoftvercsomag lehetőséget nyújt mindkét üzemmódra. Mivel a half duplex mód nem kompatibilis a DMA-val, melyre később nagy szükségünk lesz, full duplex módot kell alkalmaznunk.

Nem elhanyagolható probléma az SPI buszon az ADNS chipek korlátozása. Mivel igen régi tervezésűek, lassúak. Az IC egy regiszterének megcímezése és az adat rendelkezésre állásához szükséges idő mai mértéket tekintve rengeteg. Mivel az ESP-IDF-et a mai rendszerek figyelembe vételével tervezték, a szoftver nem képes a cím kiküldése és az adat vissza olvasása közben más kódot futtatni. További részletek az ADNS-3080 alatt olvashatók, illetve a részletes megvalósításról a forrásban lévő kommentek nyújtanak további információt.

4.3.2.3. ATmega328p

Az ESP32 és az 328p közötti kommunikáció I2C-n fut, de szükséges volt egy parancs formátum kialakítása is ezen a rétegen felül. Ezt folyamatában alakítottam ki az alapvető bájt átküldésektől felépítve a több bájtos csomagok formátumáig. A korábban bemutatott 40. ábra másik fele a 41. ábrán látható.



41. ábra Parancsok felépítését ábrázoló blokkdiagramm részlet

Az ábrán, jobb oldalon az implementált parancsok listája, paraméterei és hatása látható. Ezeket a buszon bájt sorozatként lehet az eszköznek küldeni, ami megfelelő módon reagál.

Az ESP32-n az alapvető kommunikációért az I2C felel, a magasabb szintű függvényeket az ATMega328p nevű komponens gyűjti. A 328p-n az I2C nevű forrásfájl felel a bejövő adatok pufferelséért. A komponens maga csak egy jelzőt állít be, és a fő programban futó állapotgép hajtja végre a pufferelt parancsot. Ennek köszönhetően kommunikáció szinte aszinkron működik interrupt alapon. Természetesen az IC-nek vannak korlátozásai, melyeket optimalizálással lehet javítani. Ilyen gyengeség az, hogy amennyiben egymás után túl gyorsan küldünk parancsot, az eszköz nem válaszol ACK jellel, amitől az ESP32 hibát dob és újraindul.

4.3.3. ADNS-3080

A specifikáció szerinti újfajta pozicionáló rendszer nem működhet a szoftver nélkül. A szenzor konfigurálása az ESP32 bootolásakor megtörténik. Ekkor a felbontást a maximálisra állítom, ez 1600 cpi. A szenzor a fotókat nagyjából 6000 fps- el rögzíti. Az elmozdulást egy belső regiszterbe gyűjti, mely olvasásakor a tartalom nullázódik. Az

elmozdulás méréséhez tehát annyi a feladat, hogy rendszeresen olvassuk ki a szenzorok regisztereit és konvertáljuk ezt a robot koordináta rendszerébe.

A rendszeres kiolvasásért egy már korábban megismert eszköz, egy task felel. A folyamat rendszeresen lekéri az alapvető adatokat és kiírja őket a kijelzőre. A lekérés folyamata a következő. Mielőtt elmozdulás adatokat olvashatnánk a szenzorból, először státusz bájtot kell kiolvasni. Ebből az IC arra következtet, hogy a következőben szeretnénk az elmozdulás számláló regisztereket olvasni, és előkészíti a tartalmukat. A következőben olvashatunk az adott regiszterből, ami egy előjeles 8 bájtos egész szám lesz. Az IC képes nagyobb adat eltárolására is mint +-127, de egyszerre csak ennyi olvasható ki. Ideális esetben azonban nem szabad hagyni, hogy a tároló felteljen, mert az torzíthatja a pozíció számítás eredményét.

A tárgyalt kód az ADNS3080 forrásfájlokban található meg.

4.3.4. VILÁGÍTÁS

A járműveknek szüksége van világításra, hogy jelezzék „szándékukat" a környezet felé. Nem más a helyzet ez az AGV-k esetén sem. A munkatérben tartózkodó emberek látniuk kell előre, hogy az egység milyen mozgás fog végezni. Ezt fényekkel is lehet jelezni, ahogy én is tettem. A közúti közlekedés szabályai alapján, a jármű elején fehérek, hátulján pirosok, a két oldalán pedig sárga fényű LED-ek találhatók.

A fények MOSFET erősítőn keresztül az MCP23S17 IC-re vannak kötve. A tervek szerint egyenes vonalú mozgáskor a fehér és a piros fények aktívak, forduláskor pedig az adott irány oldalának sárga fényei villognak. A villogásért ismét egy task felel, mely globálisan állítható logikai változókat figyel, és a szerint kapcsolja a fényeket.

4.3.5. ATMEGA328P FIRMWARE

A chip nagy előnye hogy Arduino IDE segítségével programozható. A közösségnek köszönhetően nagy a támogatás, nincs olyan hardware, aminek ne lenne megírva a könyvtára az interneten. Igy volt ez az általam az IC-re kapcsolt összes komponenssel is. A forráskód itt is komponensek szerint van bontva forrás fájlokra.

A fő program gyakorlatilag egy állapotgépet valósít meg. Egyes funkciók csak egyes állapotokban érhetők el. Erre azért volt szükség, mert a komponensek párhuzamos futása kezelhetetlen bugokat okozott, és egyébként sem volt szükség azok párhuzamos üzemeltetésére. Egy ilyen példa hogy az RGB LED vezérlése időzítésen alapszik és egyes szakaszokban kikapcsolja a megszakításokat. Ha eközben a motor meghajtás is működik, a CPU megakad és a program újra indul. A funkciók elérhetősége a 4. táblázatban látható.

STATE	I2C	LED	RFID	MOTOR	SERVO
MOVING	ENABLED	DISABLED	DISABLED	ENABLED	DISABLED
STANDING	ENABLED	ENABLED	DISABLED	DISABLED	ENABLED
LOADING	ENABLED	ENABLED	ENABLED	DISABLED	ENABLED
WAITING	ENABLED	ENABLED	DISABLED	DISABLED	DISABLED

4. táblázat Funkciók elérhetősége állapotok szerint

Az állapotok jelentése:

- MOVING: Ebben az állapotban közlekedik az AGV. A motorok csak és kizárólag ekkor aktívak. Működnek a TIMER0 és TIMER1 időzítő megszakítások. A LED vezérlés csak ekkor van kikapcsolva, mivel itt konfliktust keltene.
- STANDING: Ebben az állapotban vagyunk, mikor a kart emeljük fel a fészek magasságába. Értelem szerűen a szervo vezérlés aktív. Ilyenkor az RFID kikapcsolt állapotban van.
- LOADING: Ugyan az, mint az előző, plusz az RFID vezérlő is aktív. Ez az állapot nem tart tovább, mint 10 másodperc. Az idő letelte után a vezérlés automatikusan STANDING állapotba lép vissza. Erre a motorok megóvása miatt van szükség.
- WAITING: Itt csak a LED vezérlése aktív. Ebben az állapotban a motor és a szervo is kézzel elforgatható, a fogyasztás minimális. így várakozik az egység.

A következőkben a funkciókat részletesebben tárgyalom.

4.3.5.1. Mozgás

A hardver részben már megismert léptetőmotor és annak vezérlő elektronikájának a szoftveres oldaláról szól ez a rész. Bemutatom továbbá a parancsba adott lépésszám kiszámolásának menetét bizonyos szakasz vagy elfordulás megtétele előtt.

Az ajánlott fél lépéses üzemmódot használtam a nagyobb sebesség elérése érdekében. A másik lehetőség az egész lépéses üzemmód lett volna, aminek előnyei az egyszerűbb vezérlőprogram és alacsonyabb fogyasztás. A két üzemmód összehasonlítása az alábbi két táblázatban található. Az egész lépéses módot az 5. táblázat mutatja be, míg a fél lépéses módot az 6. táblázat.

Egész lépés	'A' tekercs	'B' tekercs	'C' tekercs	'D' tekercs
1. lépés	1	0	0	0
2. lépés	0	1	0	0
3. lépés	0	0	1	0
4. lépés	0	0	0	1

5. táblázat Egész lépéses üzemmód

A lépések egymás után következnek időben, a negyedik lépés után ismét az első jön. Ebben az üzemmódban minden 32 lépés alatt fordul körbe a motor rotora. A lépések irányának megfordításával megfordul a forgásirány.

Egész lépés	'A' tekercs	'B' tekercs	'C' tekercs	'D' tekercs
1. lépés	1	0	0	0
2. lépés	1	1	0	0
3. lépés	0	1	0	0
4. lépés	0	1	1	0
5. lépés	0	0	1	0
6. lépés	0	0	1	1
7. lépés	0	0	0	1
8. lépés	1	0	0	1

6. táblázat Féllépéses üzemmód

Ebben az üzemmódban 64 lépés alatt fordul körbe a rotor. A fogyasztás nagyobb, de nő a nyomaték, a határfrekvencia és felbontás is.

A motor megforgatásához az ESP32 bootolása után fel kell venni a kapcsolatot a 328p mikrokontrollerrel I2C buszon. A kontroller parancsokat vár bájt csomagok formájában. A formátumot már bemutattam a korábbi fejezetekben. Az elindulás menete a következő:

- Be kell állítani a kerekekre a szükséges számú lépést.
- Az AGV-t MOVING állapotba kell kapcsolni.
- Meg kell adni a mozgás irányát.

Minden egyes pont egy külön bájtcsomagot jelent. Lehetséges irányok az előre, a hátra és a helyben jobbra vagy balra. A parancs kiadása előtt a geometriai információk alapján közelíti a program a megteendő lépések számát. A kiadott parancs az előírthoz hasonló pályát eredményez, de a folyamatos mechanikai csúszás miatt időnként korrekcióra van szükség. A korrekció egyenes vonalú mozgás esetén a sebesség növelésével vagy csökkentésével lehetséges az egyes oldalakon. Elfordulásnál szintén további lépések megtétele a megoldást, amennyiben a valós elfordulás nem éri el a kívántat.

A (4.6) egyenlet alapján az elmozdulás egy lépésre 0.04625mm. Ennek alapján az elmozdulási szükségelte ismeretében kiszámolható, hogy hány lépés alatt tehető meg a táv.

$$(szükséges\ lépészám)\ l = \frac{\text{elmozdulás [mm]}}{0.04625\ \text{[mm]}} \qquad 4.7$$

Az elfordulás is hasonló módon a korábbi értékből és egy geometriai paraméterből számítható, feltételezve hogy a két motor egyszerre mozdul ellenkező irányba. Az a bizonyos paraméter a kerék futópontjának középpontól vett távolsága. A 3D modell alapján ez:

$$(távolság)\ t = 55.5\ [\text{mm}] \tag{4.8}$$

$$\left(\frac{fok}{lépés}\right)\ \phi = \arctan\left(\frac{\text{D}}{t}\right) = 0.04775[\text{deg}] \tag{4.9}$$

$$(szükséges\ lépészám)\ \varphi = \frac{\text{elferdulás [deg]}}{\phi[deg]} \tag{4.10}$$

90 fokos elfordulás esetén ez:

$$(szükséges\ lépészám)\ \varphi = \frac{90\ [\text{deg}]}{0.04775[deg]} = 1885[-] \tag{4.11}$$

4.3.5.2. Megfogó kar

A kar a korábban bemutatott két szervo segítségével mozog. Ismét megszorítások miatt nem a hivatalos könyvtárat használtam. Mivel a nullás és egyes időzítőket a motorok vezérlésére használtam, kénytelen voltam egy másik, a kettes időzítőn alapuló könyvtárat keresni [36].

A hírtelen rándulások elkerülése érdekében a szervo átállási folyamat lassított. A szögértékeket folyamatosan növelem, késleltetéseket közbe iktatva. Ezeket a paramétereket külön lehet állítani a kódban és egészen finomra hangolható a mozgás.

4.3.5.3. RFID olvasó

A modul működtetéséhez a hivatalos MFRC522 könyvtárat használtam [37]. Az RFID olvasó csak a megfelelő állapotban aktív, a korábban említett ütközés miatt a motor vezérléssel. Ilyenkor az olvasó interrupt jelet küldhet a mikrokontrollernek, jelezve hogy elérhető címke van az olvasási hatótávon belül. A leolvasás végeztével az egyedi azonosítót eltárolható, és a modult deaktiválható. Az eltárolt azonosító innentől kezdve bármikor lekérhető.

4.3.5.4. Állapotjelző fények

A korábban bemutatott WS2812B meghajtásához az Adafruit_NeoPixel könyvtárat használtam [38]. Az állapotokat az alábbi módon rendeltem a fényekhez:

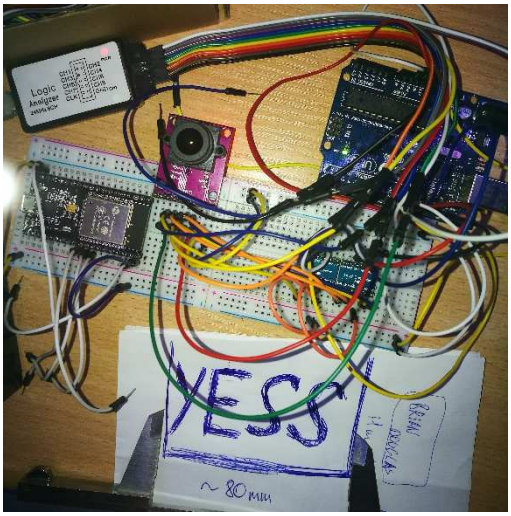
- MOVING állapot közben a led sárga színű
- STANDING állapot közben a led türkiz színű
- LOADING állapot közben a led piros színű
- LOADING állapotban leolvasás esetén a led zöldre vált
- WAITING állapot közben a led kék és pink színek között vált

5. TESZTELÉS

Ebben a fejezetben gyűjtöttem össze a fejlesztés során és a végső prototípuson végzett teszteket és eredményeiket. A fontosabb modulokat külön is teszteltem, hogy megbizonyosodjak, megfelelnek-e majd a célra. Ezek közé tartozik az ADNS szenzor és a léptető motor is. Kissé kilóg a sorból a digitális kommunikáció teszlelése, de a fejlesztés jelentős része telt ezzel, így szerettem volna, ha ez a dolgozatban is megjelenne.

5.1. ADNS első tesztek

A szenzort korábban nem használtam és az interneten is kevés teszt adatot találtam. Ekkor még nem volt adott a lehetőség a szenzor elmozdulás mérési funkciójának tesztelésére. A teszt a szenzor képminőségére irányult. Megfelelő képminőség esetén ugyanis az abszolút pozicionálást is erre lehet alapozni. Mivel ekkor még szinte el se kezdtem az ESP32 kódjának megírását, illetve az Arduinoval is több tapasztalatom volt, a tesztkódot itt írtam meg. A kép megjelenítésére egy C#- os asztali alkalmazást írtam. A teszt elrendezés a 42. ábrán látható.



42. ábra ADNS modul első tesztjének elrendezése

Az ábrán látszik egy próbapanel, rajta egy ADNS modullal és egy feszültségszintváltóval (jobb oldalon a vezetékek alatt). Baloldalon látható az ESP32 az első tesztjeihez szintén próbapanelre rakva. A próbapanel felett látható a logic analyzer és az e teszthez használt Arduino UNO. Alul a következő ábrán lévő kép eredetije látható.

A képet úgy jelenítettem meg, hogy az UNO a szenzort képadatok kibocsájtása üzemmódba állította, a kinyert adatokat pedig továbbította soros porton az asztali számítógépen futó C# alkalmazásnak. Az alkalmazás a bájtsorozatot értelmezve egy 30x30-as képet alkotott az adatokból. Az első képek a kamerából a 43. ábrán láthatók. A kép eredetije a 42. ábrán alul is látszik.

43. ábra ADNS modul által vett kép megjelenítése

A kamera ekkor a leképezett felülettől 210mm távolságra volt, a szöveg szélessége pedig 80mm. Ezekből az adatokból és a kamera felbontásából számítható egy hozzávetőleges szögfelbontás. A pixelek oldal irányában és átló irányában különbözik a felbontás.

$$(távolság)\ t_A = 210\ [\text{mm}] \tag{5.1}$$

$$(szélesség\ )\ a_A = 80\ [\text{mm}] \tag{5.2}$$

$$(átmérő)\ d_A = \frac{a}{\sin(45°)} = 113.18\ [mm] \tag{5.3}$$

$$(látószög\ szél.)\ \alpha 1 = 2 * atan\frac{a_A\,/\,2}{t_A} = 21.57\ [\text{deg}] \tag{5.4}$$

$$(látószög\ átm.)\ \alpha 2 = 2 * atan\frac{d_A\,/\,2}{t_A} = 30.14\ [\text{deg}] \tag{5.5}$$

30x30 pixeles felbontás esetén ezekből a felbontási szögek a következők:

$$(felbontás\ szél.)\ \varphi 1 = \frac{\alpha 1}{30} = 0.719\ [\text{deg}] \tag{5.6}$$

$$(felbontás\ átm.)\ \varphi 2 = \frac{\alpha 2}{30} = 1.005\ [\text{deg}] \tag{5.7}$$

A végső megvalósítás szerint a kamera felfüggesztési magassága 14.5mm. Ez alapján a jól érzékelhető felirat vonal vastagsága:

$$(vonalvas.)\ v = 14.5 * 2 * \text{tg}\frac{\varphi 1 + \varphi 2}{4} = 0.218\ [\text{mm}] \tag{5.8}$$

Egy átlagos inkjet nyomtató felbontása legalább 300dpi [39], amivel a legvékonyabb húzható vonal szélessége 1 inch 300-ad része. Ez 0.085mm. Olyan rajzok nyomtatásához, amely ezzel a felbontással kedvezően felismerhető akár egy irodai nyomtató is használható.

A tesztelés során azt is felfedeztem, hogy a szenzor fényérzékenysége igen alacsony. Megfelelő megvilágítás nélkül semmit nem lát. Pontos mérésekre nem volt eszközöm, így fény intenzitás adatokat nem tudok közölni.

A kamera képes az érzékelt képből minőséget számolni és ezt százalékos értékben adja vissza. A végső összeállításnál 4 LED világítja meg a felületet. Minden más paramétert változatlanul hagyva, egyéb külső fényforrást beiktatva akár 4%-os képminőségjavulás érhető el. Ez alapján a megvilágítás mértéke tovább növelendő.

5.2. Léptető motor első tesztek

A léptető motor kiválasztása első sorban az elérhetőségen és áron alapult, így jogos aggodalom volt, hogy nem fog megfelelni sebességben, nyomatékban vagy más paramétereiben. Ezek közül a sebesség a legkritikusabb, hiszen a motor egy igen nagy hajtóművel van felszerelve, nagy nyomatékot képes leadni.

Ennek megfelelően a tesztelés a motor maximális sebességére irányult. Ezt úgy végeztem, hogy a motort próbapanelen lévő csatlakozóra kötöttem, és egy Arduino UNO-ra írt tesztkóddal különböző sebességekkel hajtottam meg. A tesztet elvégeztem az ajánlott fél- és az egészlépéses üzemmódban is. Az utóbbit annak reményében, hogy nagyobb sebességet érhetek el vele. Először terhelés nélkül teszteltem, később a motort az első prototípus testébe szerelve, a várható terhelést imitálva. Az eredmények a következők:

Az egész lépés az elvárásaim ellenére nem volt gyorsabb. A tesztek alapján pont az ellentéte valósult meg, sebesség csökkenést eredményezett. Ezen felül a terheléssel folytatott teszt során is alul maradt. Az eredmények a 7. táblázatban láthatók.

Üzemmód	Átesési késleltetés	Átesési frekvencia	Motor tengely frekvencia
Fullstep	1750 [us]	571 [Hz]	0.279 [Hz]
Halfstep	800 [us]	1250 [Hz]	0.305 [Hz]

7. táblázat Maximális üzemi frekvencia

Az oszlopok jelentése a következő, a késleltetési idő az, amit az egyes lépések megtétele közé iktattam az Arduino kódban. Ezt addig csökkentettem, amíg a motor még elfordult és nem hagyott ki lépést a túl magas meghajtási frekvencia miatt. Ebből az értékből számíthattam az átesési frekvenciát. Ez az érték az egyes lépésekre vonatkozik. Hogy ebből megkapjuk a motor tengely átfordulási frekvenciáját, el kell osztani a hajtómű és az üzemmód által meghatározott lépésszám szorzatával. A tengely átfordulási frekvenciát beszorozva a kerék kerületével megkapjuk az elméleti maximális sebességet. Ez nagyjából 57.5 [mm/s].

A teszt során észrevettem, hogy nagyobb sebességet lehet elérni amennyiben a sebesség felfutás jellegű, nem azonnal a határfrekvencián kezd. A bemutatott adatok ilyen módon születtek.

Az üzemmód kiválasztása után külön teszteltem a fogyasztást is az alsó és felső határfrekvencián is. Alsó korlát azért van, mert a megvalósításnál további megszorításokat hoznak a 328p időzítői. A Timer0 beállítható legalsó frekvenciája 62 Hz. A fogyasztás a 8. táblázatban látható:

Frekvencia	Fogyasztás
Alsó	250 [mA]
Felső	170 [mA]

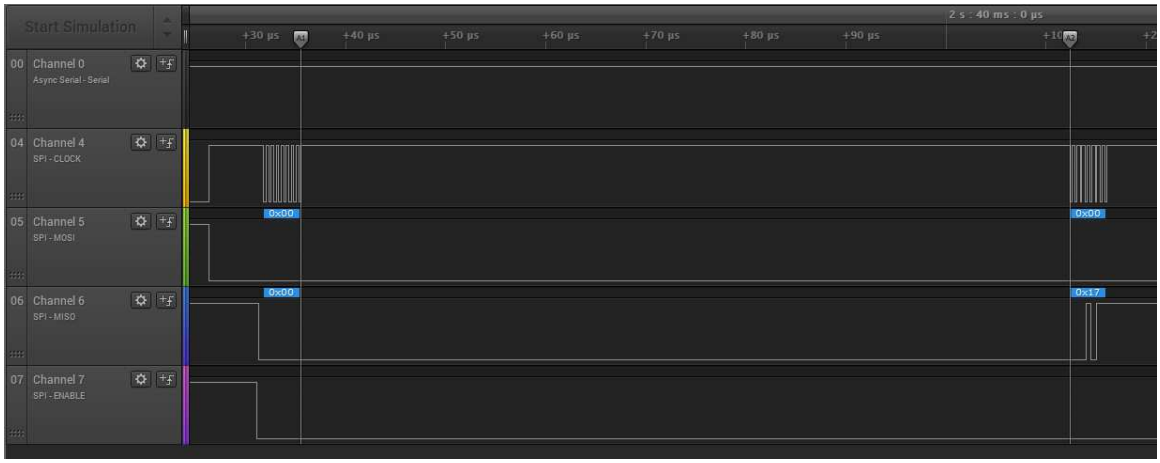
8. táblázat Motor fogyasztása

Ezek alapján a motor fogyasztása kedvezőbb nagyobb sebességeken. Feltételezésem szerint a nagyobb frekvencián jobban érvényesül a Darlington Array IC lábára tett kondenzátor energia visszanyerő szerepe. Lehetséges még az is hogy az egyszerű multiméterem kevésbé képes magasabb frekvenciás áramok mérésére. Az áramot a közös motor tekercsek kivezetésén mértem.

5.3. Logic analyzer tesztek

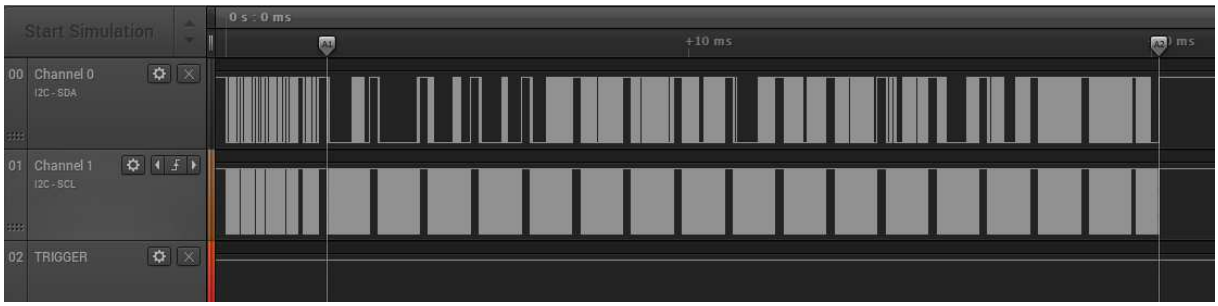
A fejlesztés során sokszor ezzel az eszközzel értem el az áttörést. Az elkészült kódokat először ezzel mértem le. Ellenőriztem, hogy megfelel-e a specifikációnak a mért digitális jelsorozat. A 328p és ESP32 közötti kommunikációban pedig ez után írtam meg a fogadó oldali kódot.

Egyik ilyen teszt az ADNS modulok SPI kommunikációs sebességének mérése volt. Korábban említettem, hogy a chipek régebbi tervezésűek. Szükségük van egy jelentős várakozási időre, hogy az adatokat ki tudják léptetni. A teszt során lassan csökkentettem a várakozási időt, míg el nem jutottam addig a várakozási időig, ami felett még konzisztensen megkaptam minden adatot. Ez a 44. ábrán látható mérési eredmény szerint kicsivel több, mint 75 [µs].



44. ábra ADNS-3080 SPI kommunikációjának mérési eredménye

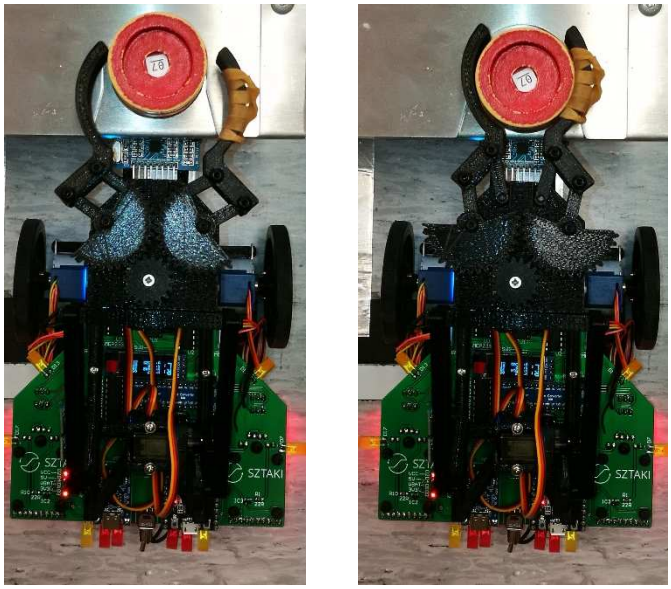
A következő teszt az OLED kijelző és az ESP32 közötti kommunikációra koncentrál. A teszt egyrészt a megjelenítendő kép kiküldési idejének mérésére szolgált másrészt az elején fel kellett építenem a kijelző inicializáló szekvenciát a már működő Arduino könyvtárra alapozva. Ezt egyrészt a kód tanulmányozásával tettem, másrészt a kommunikáció megfigyelésével az alábbi módon. A 45. ábrán látható kommunikációs folyamat eleje egy inicializáló szakasz. Ez az első jelölőig tart. A két jelölő között maga a kép adatainak küldése folyik. Ez a mérés alapján 18 [ms], ami közel 55 képfrissítésre elegendő másodpercenként, feltételezve hogy csak a kijelző használja az I2C buszt.



45. ábra OLED I2C kommunikációja

5.4. Kar tesztelése

Ezen teszt során a kar tervezett működésének ellenőrzése és az előre nem látott hibák felfedése volt a célom. A tesztet a teljes kiépítéssel végeztem, ezért egyúttal ellenőriztem a motoros meghajtás működését is. Igyekeztem optimális beállításokkal dolgozni, de tapasztalataim szerint van jobb paraméter is a karok emelési késleltetéseire. Ezek finomhangolására azonban már nem volt lehetőségem. A működés a 46. ábrán látható.

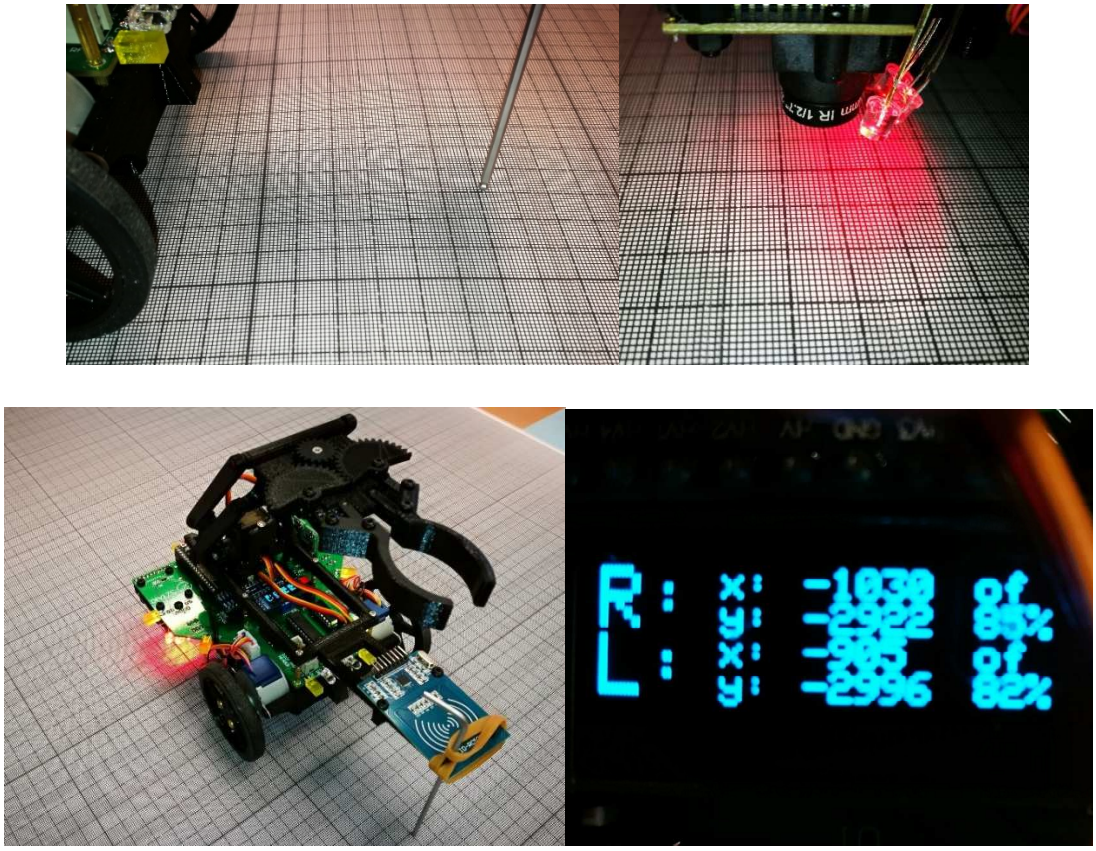


46. ábra Megfogó kar működés közben

A teszt során egy előre beprogramozott mozdulatsort végzett el az egység. Megállítva azt a fészek előtt, felemelte a karját, rágurult a munkadarabra, megfogta majd legurult vele. A feladatot néhány paraméter hangolással többnyire elvégezte. Pár alkalommal a megfogás nem volt elég stabil. Ennek kiküszöbölésére egy postás gumiszalagot tekertem a megfogó pofák egyikére. Ezzel a maradék tesztek hiba nélkül mentek végbe. A megoldást lehet egy rugalmas elem használata vagy a szervok mozgástartományának további finomhangolása. Fontosnak tartom még a munkadarab felvelő mozgás sorozat bővítését is. A munkadarab megfogása után további néhány milliméter elemelkedés a fészektől sokkal biztosabbá tenné a munkadarab felvételt.

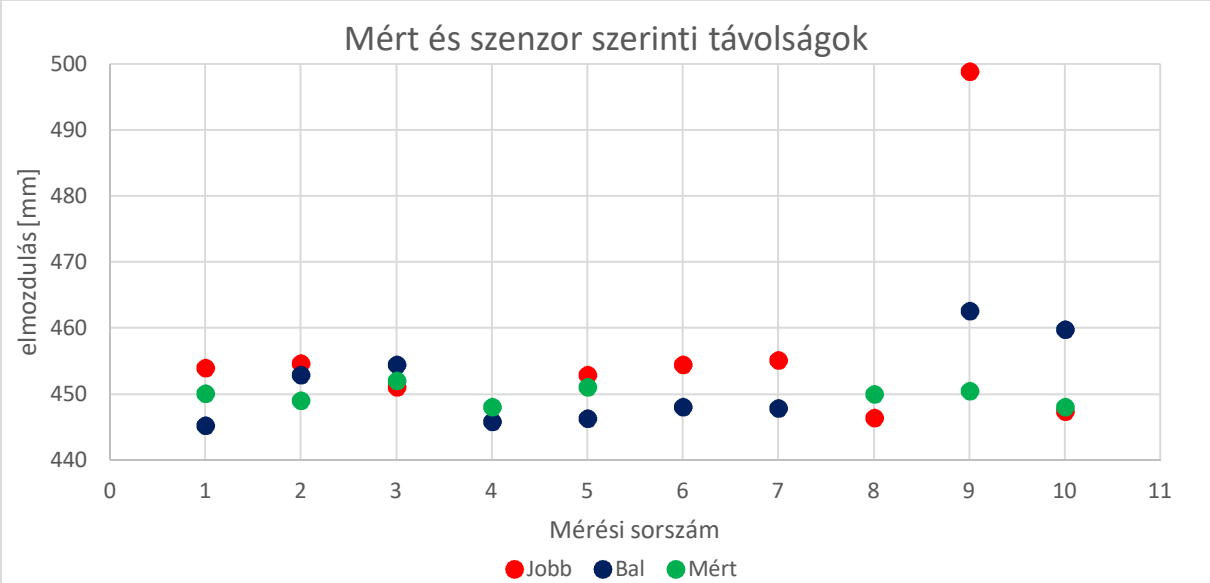
5.5. ADNS-3080 tesztelése

A teszttel a célom a szenzor mérési pontosságának becslése volt. A mérést a legutolsó modellel végeztem a fejlesztési fázis legvégén. A tesztet a 47. ábrán látható módon, milliméterpapíron végeztem. A képminőséget 85% körülire határozták meg a szenzorok a papíron. Ez az ábrán a jobb alsó részben látható. A méréshez adott számú lépést kellett megtennie a motoroknak. A kezdő és a végpontot az ábra bal felső sarkában lévő mutató segítségével olvastam le. Ezt vetettem később össze a szenzorok által mértel.



47. ábra Szenzor tesztelés mérési elrendezése

A tesztelések során 10000 lépést használtam és átlagosan 450 [mm] elmozdulást értem el. A számítás alapján 462.5 [mm] kellett volna legyen, de ez nem veszi figyelembe a csúszásokat. Ebből is látható a mérőszenzor szükségessége. A távolságot átlagosan 36 [s] alatt tette meg. Ez majdnem 13 [mm/s] sebességet jelent. Az alábbi 48. ábrán látható a tesztek eredménye. A mutatóról leolvasott érték a „Mért", a „Jobb" és „Bal" pedig a kettő szenzor által mért mennyiség normalizált értéke. Hogy könnyebben ábrázolni tudjam, minden kijelzőről leolvasott számot elosztottam ugyanazzal az értékkel, amit úgy választottam, hogy a lehető legjobban illeszkedjenek a szenzoros és mért értékek. Ez a szám később akár váltószámnak is használható a szenzor egység és valós távolság közti konverzióban. Jelen esetben ez 7.15 [osztás/mm].



48. ábra Szenzor mérési eredmények

A 9. futam kiemelkedő eltérése eredhet abból, hogy a tesztet a négyzethálóra átlósan végeztem. Ez összezavarhatta a szenzor számoló algoritmusát, így ezt nem veszem bele a pontosság meghatározásába. Ezt kizárva a jobb szenzor eltérése a valós távolságtól -4% és 1% között mozgott. A bal szenzor -2% és 3% között. A hibát ezek alapján 5%-ra becsülöm.

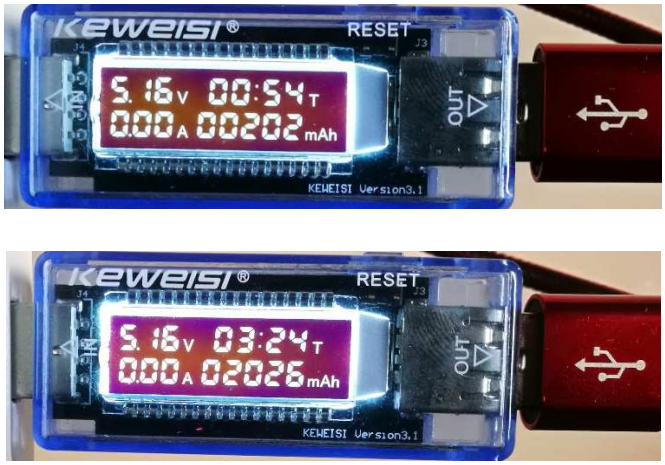
A milliméterpapíros tesztelés mellett kipróbáltam a SmartFactory burkolatját is, hogy milyen képminőséget ér el rajta az ADNS szenzor. A minőség 7% és 30% között ugrált, egyes pontokon akár 40%-ot is elérve. A beprogramozott lépésmennyiség ugyan az volt, mint a korábbi, de elmozdulás mérésére itt nem volt eszközöm. Ezek a 6. és 7. mérési eredmények, ezért nincs ezen oszlopokban mért eredmény feltűntetve.

5.6. Fogyasztás

Ezen teszt segítségével becslést akartam adni az akkumulátor kapacitás és fogyasztás alapján az üzemidőre. A tesztet úgy végeztem, hogy az akkumulátorokat először teljesen feltöltöttem. Egy USB-s áram és feszültség mérő segítségével mértem az akkuk energia felvételét. Ilyen módon az áramfelvételt és a feszültséget is könnyen tudtam mérni, mivel az AGV USB Type C-vel tölthető.

A mérő eszköz a 49. ábrán látszik. Először az üresjárati vagy várakozó állapotban lévő fogyasztás mérését végeztem. Az egységet bekapcsoltam és egy órán át hagytam állni. A fő fogyasztók ilyenkor az ESP32, ezen belül is főleg a vezeték nélküli kommunikációs modulja. Az ATmega328p és a fényforrások. Itt első sorban az ANDS szenzorok megvilágítása. Egy óra elteltével töltőre tettem az egységet és teljesen feltöltöttem. Az eredmény a 49. ábra felső részén látható. Összesen 202mAh-t vett fel az egység.

A második teszt a motorok fogyasztására irányult. Összesen egy órát járattam a motorokat, 10 perc futtatással és 5 perc lehűléssel. Erre azért volt szükség, mert a motorok érinthetően forróvá váltak, a feszültségnövelő pedig érinthetetlen hőmérsékletre melegedett 10 perc után. Az első, sikertelen teszt alkalmával az egység lekapcsolt 15 perc folyamatos motor működés után. A töltés eredménye az ábrán alul látható. Összesen 2026mAh a felvett töltés.



49. ábra USB-s áram feszültség és elektromos töltés mérő

A veszteségeket nem volt lehetőségem mérni így nem tudok velük számolni, ezért mindent veszteségmentesnek veszek. A feszültség végig 5.16V volt. Az első töltés így 1.04Wh energia felvételt jelentett, a második pedig 10.45Wh-t.

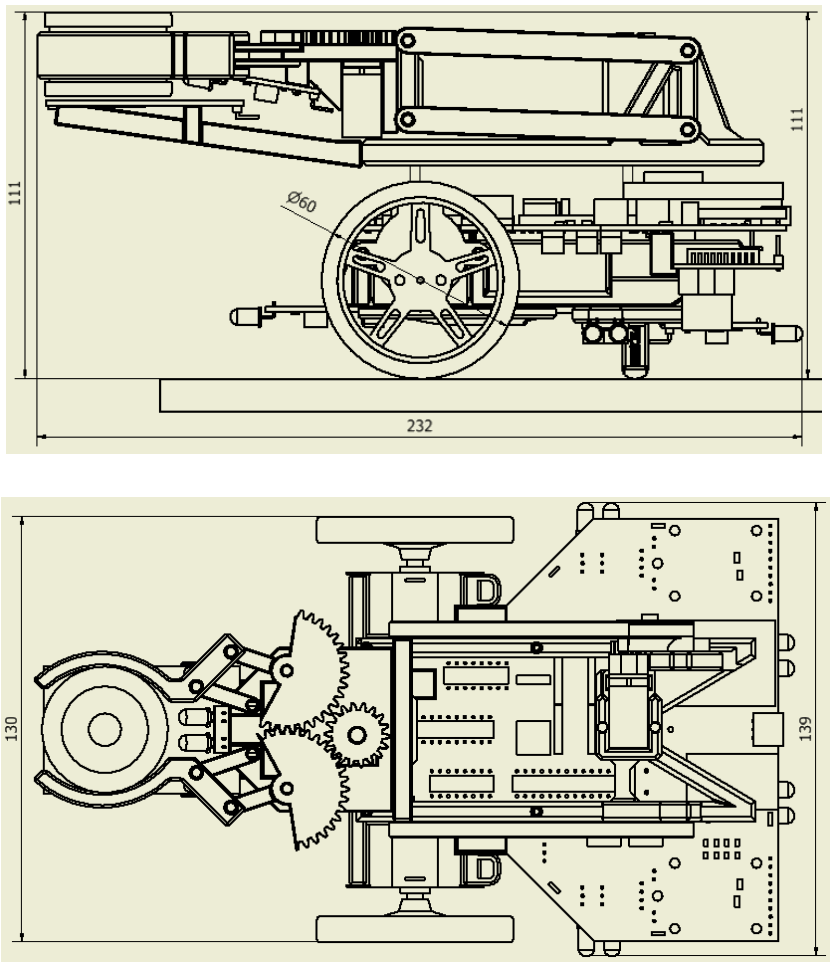
Az akkumulátorok 2600mAh kapacitásúak és párhuzamosan vannak kötve. A középfeszültségük 3.7V. Ez alapján az energia kapacitás becsült értéke 19.2Wh. A fogyasztás alapján nagyjából 1.8h mozgásra és 18.5h várakozásra képes.

6. ÖSSZEFOGLALÁS/EREDMÉNYEK ÉRTÉKELÉSE

A fejlesztés során sikerült a minimális funkcionalitást elérni. Ezeket az eredményeket a következőkben részletezem. A fejezet végén további fejlesztési igényeket tárgyalom és olyan bővítési lehetőségeket, melyek értékes funkciókat szolgáltathatnak.

6.1. Eredmények

A félév során megterveztem és megépítettem több AGV prototípust a kitűzött célok elérése érdekében. Az egyik legfontosabb szempont az ár volt. Az elkészült egység alkatrészlistája és a hozzávetőleges beszerzési áruk alapján a végső ár nagyjából 50.000Ft egységenként, e mellett a legtöbb esetben több alkatrész rendelésé esetén az ár tovább csökkenthető. Ezek alapján a végösszeg megfelelően alacsony. A lista megtalálható a mellékletben Excel fájlként. A végső egység rajza a befoglaló méretekkel az 50. ábrán látható.



50. ábra Végső prototípus befoglaló méretei

A megtervezett megfogó szerkezet hardveresen képes a feladat elvégzésére. Megfelelő magasságba tudja emelni a munkadarabot és képes leemelni azt a fészekről. A működési megvalósítása hangolást igényel, illetve a megfogó szerkezet is fejleszthető a jobb munkadarab tartás érdekében. Az akkumulátor kapacitás és a fogyasztás alapján, feltételezve átlagosan az idő 60%-át mozgással a maradékot pedig várakozással tölti az egység, az üzemidő várhatóan 8.5 óra. Ez megfelel az igényeknek.

Központi feldolgozó egységként az ESP32 megfelelt, a kódot valós időben képes futtatni, a feladatokat ellátni. Ezek közé tartozik a vezeték nélküli kommunikáció kezelése és az alaplapi komponensek vezérlésére. A kijelző vezérléstől a motor vezérlésen át a szenzor adatok bekéréséig.

Kommunikációs téren a Bluetooth 4.0 protokollon futó BLE technológiát tudtam implementálni. A fejlesztés során egy telefonos alkalmazással tudtam irányítani az AGV-t egy virtuális soros porton keresztül, ami ezen a BLE-n alapul. Innen elindulva magas szintű parancsok kialakításával minden szükséges információ átadható és lekérhető ezen a csatornán keresztül. Egy hátránya a protokollnak az, hogy főleg szenzoradatok továbbítására alkották meg. Nem képes például élő képet továbbítani a szenzoroktól, annak nagy sávszélesség igénye miatt.

Kommunikációhoz tartozik még az alaplapon elhelyezett eszközök közti adatcsere is. Ezek az I2C, SPI és OneWire protokollok. A kialakított szoftveres komponenseknek köszönhetően a központi egység minden periféria elérésére és vezérlésére képes.

A munkadarab azonosítására alkalmas tálcaként is szolgáló RFID olvasó is elhelyezésre került az AGV elején. A szenzor a 328p IC-re kötve egy Arduino-s könyvtár felhasználásával képes kiolvasni és lejelenteni a munkadarabban lévő címke tartalmát. A 328p eltárolja, és kérés esetén jelenti az ESP32-nek a munkadarab azonosítóját.

A szenzor képminősége megfelelő ahhoz, hogy egy képfelismerő program bemeneteként szolgáljon. Az abszolút pozicionálás ennek segítségével elvégezhető. E nélkül azonban a pozicionálás nem megfelelően pontos hosszabb szakaszokon történő navigálásra. A szenzor alatti felület megvilágítása is megfelelő, de további fényerő növelés is megfontolandó.

A kijelző képes szöveges elemek és még egy minimális grafika megjelenítésére is. Ez az AGV összetettebb belső állapotainak kijelzésére használható. Az egyszerűbb, de szintén szükséges jelzőfényeket is elhelyeztem az egységen. Ezek az elülső fényszórók, a hátsó lámpák és a kanyarodás jelző fények két oldalon.

Magasabb szintű funkciók megvalósítását a firmware bővítésével értem el. A korábban kiépített alapvető adatcserélő funkciókra alapozva magasabb szintű függvényeket készítettem, melyek képesek egy hívásból elérni hogy az AGV elinduljon, elforduljon adott lépés mennyiséget, vagy vészmegállást hajtson végre, megemelje vagy kinyissa a karját, kiírjon valamit a kijelzőre.

A dokumentációként maga a szakdolgozat, a 3D modellek, a kommenttelt forrás kódok és PDF fájlban mellékelt kapcsolási rajz szolgál. Az megértés az előbbivel, az újra gyárthatóság és további fejlesztések pedig az utóbbiakkal lehetséges.

6.2. Javaslatok / Tanulságok

A fejlesztés során sokszor ütköztem nehézségekbe. Ilyenkor vagy végig vittem a feladatot vagy találtam hatékonyabb és egyszerűbb megoldást. Ezen kívül olyan tanulságokat is szeretnék megosztani, melyek nem köthetők bizonyos alkatrészekhez, inkább az egész fejlesztésre igazak.

A legnagyobb tanulság a projektből hogy a Robotino-t leváltani képes prototípus elkészítése nem egy féléves feladat. A fejlesztés első lépéseit már a félévet megelőző nyár közepén elkezdtem, de így sem tudtam eljutni a szoftveres fejlesztés és teljes hibajavítás végére. Az AGV-re nagy igény van, így a fejlesztést folytatom majd a következő félév során, de ezt már nem egyetemi hallgatóként teszem.

Tanulság még hogy minden funkcionális bővítés után kell egy prototípus, aminek a tesztelése során a felfedezett hibákból egy javított változat gyártható. Minden modellben van hiba, melyek csak a tesztelés során kerülnek elő.

Képességeim alapján az egyik legnehezebb rész a szoftver fejlesztése volt. Az ESP-IDF megismerése és a magas szintű C nyelv szükséglete komoly kihívást jelentettek számomra. Szerencsére az ESP-IDF dokumentációja sok esetben a legapróbb részletekig bemutatta az adott komponenst, de előfordult, hogy alig tartalmazott információt, vagy hiányoztak a legfontosabb részletek a leírásból.

Mivel az ESP-IDF még fejlesztés alatt áll, sok funkció csak a jövőben kerül majd be. Ilyen a NimBLE is, ami egy sokkal modernebb, kisebb memória és processzor igényű megvalósítása a BLE protokollnak. Először ezt próbáltam implementálni, de a dokumentáció szinte nem létezett és a szoftvercsomag is erősen hiányos volt. Kénytelen voltam egy másik, a végleges megoldásban is szereplő csomagra, a Bluedroid-ra alapozni. Ezt érdemes lehet a jövőben a NimBLE-re átalakítani.

6.2.1. TOVÁBB FEJLESZTÉSEK

Ebben a részben azokat a lehetőségeket ismertetem, melyeket nekem már nem volt lehetőségem elvégezni különböző okokból, főleg időhiányból kifolyólag. Egyes pontok létfontosságúak a feladat elvégzéséhez. A következőben felsorolom, majd utána a fontosabbakat részletesen is bemutatom.

- Képfelismerés implementálása a szenzor kumulálódó hibájának kiküszöbölésére
- Legfelső szintű kommunikációs protokoll kiépítése
- Nem csúszó harmadik kerék tervezése

- Ütközés elkerülő funkcionalitás implementálása
- Mesh kommunikáció kiépítése az egységek között
- Kooperációhoz szükséges szoftver komponensek implementálása
- Bluedoid leváltása NimBLE szoftvercsomagra
- Kód optimálás

A fontosabb pontok részletesebben

- A szenzorok által készített képek elemzésével megoldható az abszolút pozicionálás is. A talajra jól megkülönböztethető alakzatokat rajzolva és ezek pontos helyét ismerve a nullpont és a fészekre állások pozíciója is pontosan meghatározható. Jelölő szalagok elhelyezésével és ezek felismerésének képességével pedig akár menet közben is nullázható az összegyűjtött hiba. Ez utóbbi kevésbé célszerű, hiszen az eddigi rendszer is ilyen módon működött és ennek leváltása is célja a projektnek.
- Magas szintű parancs felismerés szükséges a rendszerbe való integrációhoz. Ezek a parancsok a „hozd az adott fészekben lévő munkadarabot az adott pontra” jellegűek. Ehhez még több különböző összetettségű réteg is hiányzik a forráskódból, jelenleg csak a legalapvetőbb funkciók működnek. E mellett még a hardveres kialakítás is kevésnek bizonyulhat.
- A harmadik kerék elhelyezése fontos lehet a hatékonyabb mozgás érdekében. Jelenlegi helyén nagy teher esik rá, és a súrlódás nagyban akadályozza a mozgást. Ennek a javítása kis feladat, ennek ellenére sokat javíthat az egységen.
- Az ütközés elkerülő szenzorok is felkerültek az egységre, de a szoftveres háttér implementálására nem jutott időm. Ez is kis feladat ahhoz képes amekkora előnyt jelenthet.

HIVATKOZÁSOK

[1] „AGV-RENDSZEREK," Gamma digital, http://www.gammadigital.hu/agv-rendszerek/. [Hozzáférés dátuma: 15. 09. 2019.].

[2] Modern Mechanix, „"NO-HANDS" TRAIN," 07. 1958.. http://blog.modernmechanix.com/no-hands-train/. [Hozzáférés dátuma: 02. 09. 2019.].

[3] „Youtube," Intel, 21. 04. 2015.. https://www.youtube.com/watch?v=-KTKg0Y1snQ. [Hozzáférés dátuma: 18. 09. 2019.].

[4] C. Shani, „SAFER HANDLING: AGV Systems," 27. 03. 2019.. https://www.mhisolutionsmag.com/index.php/2019/03/27/safer-handling-agv-systems/. [Hozzáférés dátuma: 09. 10. 2019.].

[5] „What are automated guided vehicles?," 6 River Systems, https://6river.com/what-are-automated-guided-vehicles/. [Hozzáférés dátuma: 18. 09. 2019.].

[6] „Integration of Automated Guided Vehicles," Inser Robótica, https://www.inser-robotica.com/en/news-autonomous-vehicles/. [Hozzáférés dátuma: 23. 09. 2019.].

[7] „Forklift AGVs," SCOTT, 2019. https://www.scottautomation.com/products/forklift-agv/. [Hozzáférés dátuma: 22. 10. 2019.].

[8] O. Adrián, „The autonomous routine AGV SEAT," Icos del motor, 11. 01. 2018.. http://www.locosdelmotor.com/en/2018/01/11/la-rutina-autonoma-de-los-agv-de-seat/. [Hozzáférés dátuma: 27. 09. 2019.].

[9] N. Bartnik, „Industrial Line Follower for Supplying Materials," 15. 09. 2018.. https://create.arduino.cc/projecthub/nikodembartnik/industrial-line-follower-for-supplying-materials-9a35f5. [Hozzáférés dátuma: 20. 11. 2019.].

[10] F. Sam, „Suning launches 'basically unmanned' automated warehouse with plans to use 1,000 robots," Suning Commerce Group, 04. 12. 2017.. https://roboticsandautomationnews.com/2017/12/04/suning-launches-basically-unmanned-automated-warehouse-with-plans-to-use-1000-robots/15272/. [Hozzáférés dátuma: 25. 10. 2019.].

[11] „LASER SCANNING SYSTEMS," CraneTech Solutions, https://cranetechsolutions.com/laser/. [Hozzáférés dátuma: 20. 09. 2019.].

[12] „Bluetooth 5.1," Silicon Labs, 2019. https://www.silabs.com/products/wireless/learning-

center/bluetooth/bluetooth-direction-finding. [Hozzáférés dátuma: 18. 10. 2019.].

[13] „Natural Navigation Automated Guided Vehicles," AGV Network, https://www.agvnetwork.com/natural-navigation-automated-guided-vehicles. [Hozzáférés dátuma: 20. 09. 2019.].

[14] „AGV Comprehensive Catalog PDF," Meiden, 2019. https://www.meidensha.com/catalog/PA52-3139.pdf. [Hozzáférés dátuma: 15. 11. 2019.].

[15] „Build and Flash with Eclipse IDE," Espressif, https://docs.espressif.com/projects/esp-idf/en/latest/get-started/eclipse-setup.html. [Hozzáférés dátuma: 10. 11. 2019.].

[16] „ESP-IDF Programming Guide," Espressif, https://docs.espressif.com/projects/esp-idf/en/latest/index.html. [Hozzáférés dátuma: 25. 09. 2019.].

[17] „Logic Analyzers from Saleae," Saleae, 2019. https://www.saleae.com/. [Hozzáférés dátuma: 17. 10. 2019.].

[18] „AlphaBot2 robot building kit for Arduino," Waveshare, https://www.waveshare.com/alphabot2-ar-acce-pack.htm. [Hozzáférés dátuma: 07. 06. 2019.].

[19] „Robot platformok, Robot KIT-ek," ElekRobot, 2008-2019. http://www.elektrobot.hu/termekek.php?kategoria=353&menu=menu. [Hozzáférés dátuma: 12. 09. 2019.].

[20] „6WD Curious chassis," Kit4Curious, http://kit4curious.com/6wd-curious-chassis/. [Hozzáférés dátuma: 11. 05. 2019.].

[21] „GitHub/espefuse," Espressif, 2019. https://github.com/espressif/esptool/wiki/espefuse. [Hozzáférés dátuma: 12. 11. 2019.].

[22] JACK Enterprises, „ADNS-9800 Laser Motion Sensor," tinde, 2013. https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/. [Hozzáférés dátuma: 11. 09. 2019.].

[23] „Arduino UNO Reference Design," Arduino, https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf. [Hozzáférés dátuma: 18. 10. 2019.].

[24] „Feetech Kerék 360 fokos micro robot szervókra," SG Modell, 2019. http://www.sgmodell.hu/szervokarok/2479-feetech-kerek-360-fokos-micro-robot-szervokra-1db-9g-szervora.html?search_query=kerek&results=255. [Hozzáférés dátuma: 28. 10. 2019.].

[25] A. Gandolfo, „Pinza robot,” 19. 01. 2017.. https://www.adrirobot.it/pinza_robot/pinza_robot.htm. [Hozzáférés dátuma: 05. 10. 2019.].

[26] Abel Vincze, „Gear Generator,” Iparigrafika Ltd., 2014-2019. https://geargenerator.com/. [Hozzáférés dátuma: 08. 09. 2019.].

[27] „FreeRTOS Additions,” Espressif, https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/freertos_additions.html. [Hozzáférés dátuma: 15. 09. 2019.].

[28] „FreeRTOS,” Real Time Engineers Ltd., https://www.freertos.org/. [Hozzáférés dátuma: 22. 08. 2019.].

[29] „JTAG Debugging,” Espressif, 2019. https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/jtag-debugging/. [Hozzáférés dátuma: 18. 10. 2019.].

[30] RenesasPresents, „YouTube,” 19. 02. 2018.. https://www.youtube.com/watch?v=ECEvUEkSSLg. [Hozzáférés dátuma: 11. 10. 2019.].

[31] „Build System,” Espressif, https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/build-system.html#idf-py. [Hozzáférés dátuma: 25. 09. 2019.].

[32] „YouTube,” Ellisys, 06. 08. 2019.. https://www.youtube.com/playlist?list=PLYj4Cw17Aw7ypuXt7mDFWAyy6P661TD48. [Hozzáférés dátuma: 22. 09. 2019.].

[33] „Google Play/nRF Connect for Mobile,” Nordic Semiconductor ASA, 2019. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp. [Hozzáférés dátuma: 21. 09. 2019.].

[34] „I2C,” Espressif, 2019. https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/i2c.html. [Hozzáférés dátuma: 12. 10. 2019.].

[35] „SPI,” Espressif, 2019. https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/spi_master.html. [Hozzáférés dátuma: 15. 10. 2019.].

[36] N. Bontrager, „GitHub/ServoTimer2,” 2017.. https://github.com/nabontra/ServoTimer2. [Hozzáférés dátuma: 25. 10. 2019.].

[37] M. Balboa, „GitHub/rfid,” 2019. https://github.com/miguelbalboa/rfid. [Hozzáférés dátuma: 03. 11. 2019.].

[38] „GitHub/Adafruit_NeoPixel,” Adafruit, 2019. https://github.com/adafruit/Adafruit_NeoPixel. [Hozzáférés dátuma: 05. 10. 2019.].

[39] „OKI's Technology Guide to Inkjet Printing," OKI Printing Solutions, 2009. https://web.archive.org/web/20090815080542/http://www.askoki.co.uk/encyclo/printertech/inkjet.asp. [Hozzáférés dátuma: 20. 11. 2019.].

7. SUMMARY

In my thesis, I had to upgrade a unit in a Cyber-Physical Sample Production System. The unit is an AGV (Automated Guided Vehicle), and its task is to move work pieces between different holders in the area of this mini-factory. This Industry 4.0 system is located in a laboratory in SZTAKI and it is called Smart Factory.

The task was previously carried out by bigger and more complex units, so called Robotino-s. However, these were old, expensive, difficult to maintain and became obsolete. Therefore a new, smaller, cheaper and simpler alternative was required to fulfil their purpose.

During my thesis, I designed the body and supplementary parts for the AGV. They are 3D printable and easy to assemble. The manipulator arm was also designed considering FDM (Fused Deposition Modelling) 3D printing technology. The main electronics are housed on a PCB (Printed Circuit Board) that was also custom made for this prototype. The body and the PCB design had a few iterations during the semester, and each one was tested and upgraded upon. Many design elements were scraped during the development of the hardware side and of the software side as well. To leverage the hardware, I also had to write a firmware. This code stack is responsible for the behavior of the AGV. It mainly runs on the ESP32, the central computing unit of the AGV, but there is a co-microcontroller built in on the PCB as well, which also has its own firmware. The whole unit contains the following main components. Firstly, the main board that contains all the electronics, the ESP32, the ATmega328p, the ADNS-3080 sensors, level shifters, voltage converters, GPIO expander and supplementary components. The second main part is the 3D printed body with the stepper motors and the batteries. Finally, the manipulator arm with its gripper, fixed to the upper part of the AGV and connected with screws and servo connectors.

The AGV proved to be sufficient during the testing, although there are still ways to improve. The final version is still not ready for duty, and needs to have proper sensor calibration and more advanced firmware before being able to replace the Robotino-s. The later one includes the implementation of high level functions that are based on the already written low level ones.

Keywords: *AGV, SmartFactory, ESP-IDF, PCB, FDM, Firmware*

8. MELLÉKLETEK

8.1. Eclipse

8.1.1. ESP-IDF TELEPÍTÉSE

Az ESP-IDF ez alapján a leírás alapján könnyen telepíthető, a lényeget lépésekbe szedem.

https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html#step-1-install-prerequisites

1. Telepítő letöltése: https://dl.espressif.com/dl/esp-idf-tools-setup-2.0.exe
2. Szükség van Pythonra és Gitre a működéshez, ezeket vagy megadjuk, hogy hol található a gépen, vagy telepít nekünk a telepítő.
3. A projekt során a 3.2.2-es verziójú IDF-et használtam.
4. Ha hibát jelez a telepítő, olvasd el a log fájlt. A telepítő nekem két alkalommal is képtelen volt kicsomagolni a letöltött zip fájlt, ami az IDF-et tartalmazza. Ezt manuálisan érdemes megoldani. A .esperiff mappa releases almappájában található a zip fájl. Ez után érdemes törölni a .esperiff mappát, úgy sejtem ettől akad el a telepítő második futtatásra.
5. Kicsomagolás után a telepítőt újra indítva adjuk meg a megfelelő elérési útvonalakat és a verzió választó oldalon ne letöltést válasszuk, hanem adjuk meg a kicsomagolt zip mappa helyét.
6. A telepítés végén az ESP-IDF cmd-t futtatva, ha az utolsó sorok között a „Done! You can now compile ESP-IDF projects.” felirat is látható akkor sikeres a telepítés.

Fontos megjegyezni, hogy a létrehozott ESP-IDF Command Prompt parancsikon a rendszer változókat is szerkeszti. A szerkesztés tapasztalataim szerint ideiglenes, de a számunkra fontos módosításokat felül írja legközelebbi újra indításig. Ez azzal jár hogy a következőkben elvégzett módosításokat ideiglenesen visszaállítja. Ha Eclipse-t használunk, ne futtassuk az ESP-IDF Command Prompt parancsikont!

8.1.2. ECLIPSE KONFIGURÁLÁSA

A második rész már nehezebb. A következő leírást több forrásból raktam össze, melyek egyenként nem működtek nekem.

Kolban – ESP32 könyv 402. oldal – Programming using Eclipse

https://www.youtube.com/watch?v=H07DjpTtk_k

https://docs.espressif.com/projects/esp-idf/en/latest/get-started-legacy/eclipse-setup.html

1. Telepítő letöltése: https://www.eclipse.org/downloads/
2. Az Eclipse Javat igényel.
3. Megjelenő felületen a C/C++ opciót válasszuk.
4. Első indulásnál válaszunk egy projekt könyvtárat.
5. Másoljuk át az ESP-IDF telepítési könyvtár „examples\get-started" mappájából a „hello_word" mappát a projekt mappába.
6. Ha elindult a program, „File" menü „Import…" gombot megnyomva válasszuk a „C/C++" alatt az „Existing Code as Makefile Project" opciót.
7. A projekt név ugyan az legyen, mint a korábban bemásolt mappa (hello_word) Ez később is fontos lesz, mert nem egyszerű új projektet kezdeni a semmiből, sok fájlt kell generálni, ezért a bevett szokás az, hogy egy üres projektet másolunk be a projekt könyvtárba, átnevezzük és abból kezdünk el dolgozni. (link to template: https://github.com/espressif/esp-idf-template)
8. „Existing Code Location" alatt válasszuk ki azt a mappát, ami a „Makefile" fájlt tartalmazza. Ez általában a „main" mappát is tartalmazza.
9. „Languages" alatt válasszuk ki mind a C mind a C++ opciót.
10. „Toolchain" alatt a „Cross GCC" lehetőséget válasszuk.
11. „Finish" gomb után a „Project Explorer"-ben a frissen importált projekt „Properties" ablakát megnyitva több változtatást is el kell végezni.
12. „C/C++ Build" alatt „Enviroment": Adjunk hozzá egy bejegyzést „BATCH_BUILD" névvel és 1 értékkel.
13. Ugyanitt adjunk hozzá a „PATH" változó végéhez egy „;"-t és a „xtensa-esp32-elf" fordító elérési útvonalát. Ez nálam a „Telepítési könyvtár \.espressif\tools\xtensa-esp32-elf\1.22.0-80-g6c4433a5-5.2.0\xtensa-esp32-elf\bin" alatt található.
14. „C/C++ Build" alatt a „Preprocessor Include …" almenüben a „Providers" fülön a „CDT Cross GCC Built-in Compiler Settings"-et kiválasztva írjuk át a „Command to get compiler specs" szövegmezőt erre: „xtensa-esp32-elf-gcc ${FLAGS} -std=c++11 -E -P -v -dD "${INPUTS}""
15. Ugyanitt a „CDT GCC Build Output Parser"-t kiválasztva írjuk át a „Compiler command pattern" szövegmezőt erre: „xtensa-esp32-elf-(gcc|g\+\+|c\+\+|cc|cpp|clang)"
16. „C/C++ General" „Indexer" almenüben jelöljük be az „Enable project specific settings" jelölőnégyzet.
17. Vegyük ki a pipát az „Allow heuristic resolution of includes" mellől.
18. „C/C++ Build" „Behavior" fülén jelöljük be az „Enable parallel build" jelölőt.

19. Ezen felül minden egyes include-nak külön meg kell adni a tartalmazó mappáját a fejlesztői környezetnek. Ez opcionális, mert a fordító e nélkül is megtalálja a szükséges fejléc fájlokat. Ezt a „C/C++ General" „Paths and Symbols" almenüben lehet megtenni.
20. Zárjuk be a „Properties" ablakot.
21. Jobb kattintással a projektre a „Project Explorer" fülön válasszuk a „Build Targets" almenüből a „Create…" lehetőséget.
22. „Target name" legyen „build".
23. „Build Command" alatt vegyük ki a pipát.
24. „Build Command" legyen „python Telepítési könyvtár \esp-idf-v3.2.2\tools\idf.py"
25. „OK"
26. Jobb kattintással a projektre a „Project Explorer" fülön válasszuk a „Build Targets" almenüből a „Create…" lehetőséget.
27. „Target name" legyen „flash".
28. „Build Target" alatt vegyük ki a pipát.
29. „Build Target" legyen „-p COM10 flash" ahol COM10 a csatlakoztatott ESP32
30. „Build Command" alatt vegyük ki a pipát.
31. „Build Command" legyen „python Telepítési könyvtár \esp-idf-v3.2.2\tools\idf.py"
32. „OK"

Később az így megjelent gombokkal lehet buildelni és flashelni. Ezen kívül a rendszerváltozókat is módosítani kell. Ahhoz hogy bármely konzolból használhassuk az idf.py programot, így az Eclipse-ből is, hozzá kell adnunk a rendszerváltozókhoz.

1. Hogy megszerezzük a szükséges bejegyzések sorát, legegyszerűbb, ha futtatjuk a korábban említett „ESP-IDF Command Prompt" parancsikont.
2. Másoljuk ki a „Setting IDF_PATH…" sortól kezdődően a „Checking if Python…" sorig a megjelenő szöveget és tegyük el egy szövegszerkesztőbe.
3. Nyissuk meg a Windows Enviroment Variables ablakát.
4. Hozzuk létre az „IDF_PATH" változót és adjuk a szövegszerkesztőbe másolt megfelelő sort.
5. Szerkesszük a „PATH" változót és adjuk hozzá a végéhez „;" elválasztó karakterrel az összes további szövegszerkesztőben lévő sort.
6. Ezek után menjünk vissza az Eclipse projekt Properties ablakába, ahol a „C/C++ Build" alatt „Enviroment" almenüben a „PATH" változó végéhez adjuk hozzá a „;${PATH}" szöveget.

8.1.3. BUILD ÉS FLASH

Van még egy további lényeges funkció, melyet nem lehetett az Eclipse alá szervezni, ez a menuconfig. Navigáljunk el konzollal abba a mappába ahol a Makefile található és futtassuk az „idf.py menuconfig" parancsot. Az alapbeállítások használhatók, de néhány beállítás elég beszédes, ezek tetszés szerint állíthatók. Mentés és kilépés. Ez generálni fog egy fájlt, ami szükséges a build során. Visszatérve az Eclipse-be, kattintsunk duplán a build gombra a „Build Targets" alatt. Ha végzett a build kattintsunk a flash gombra. Amint megjelenik a konzolban a „Serial port COMxx" felirat nyomjuk meg és tartsuk lenyomva az ESP32-n a flash gombot. Amint további sorok jelennek meg, elengedhetjük a gombot. Feltöltés után opcionálisan soros portot nyitva láthatjuk az első programunkat futni az ESP32 platformon. Amennyiben nincs kikapcsolva, az alapbeállítás szerint az RTOS indulásakor rengeteg információ jelenik mega konzolon.

8.2. Hardveres információk

Alábbi táblázatban látható az alkatrészlista, a szükséges darabszámok és egy becslés az egység anyagköltségeire. A külső mellékletben megtalálható Excel formátumban is további oszlopokkal. Az itteni változtat vágva van hogy kiférjen és olvasható is legyen.

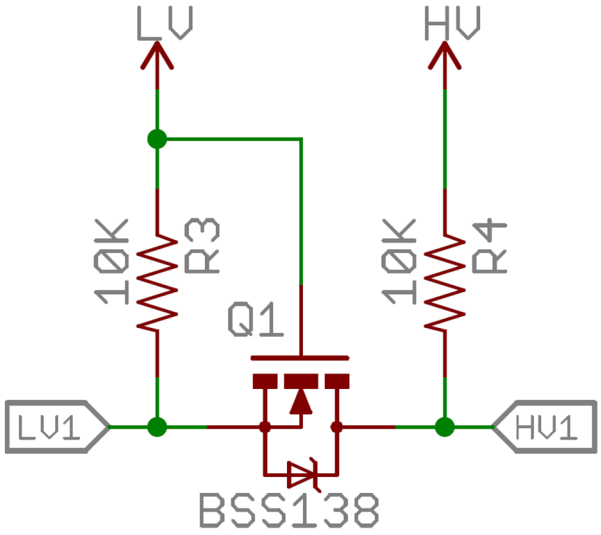
Comment	Description	Footprint	Quant	Designator	Unit Price	Price
1K	SMD Resistor	0805R	1	R7	2.8	2.8
1K2	SMD Resistor	0805R	1	R5	2.8	2.8
1K5	SMD Resistor	0805R	1	R8	2.8	2.8
1M	SMD Resistor	0805R	1	R9	2.8	2.8
2K	SMD Resistor	0805R	1	R6	7.3	7.3
4K7	SMD Resistor	0805R	1	R3	2.8	2.8
4u7	SMD Capacitor Ceramic	1206C	4	C4, C7, C9, C10	22.9	91.6
10K	SMD Resistor	0805R	4	R4, R12, R13, R14	4	16
15K	SMD Resistor	0805R	1	R2	3	3
16MHz		HC49SM	1	XT1	119	119
22p	SMD Capacitor Ceramic	0805C	2	C2, C3	9.7	19.4
22R	SMD Resistor	1206R	4	R1, R10, R11, R17	2.9	11.6
56R	SMD Resistor	0805R	6	R15, R16, R18, R19, R20, R21	2.8	16.8
100n	SMD Capacitor Ceramic	0805C	6	C1, C5, C6, C8, C11, C12	6	36
ADNS3080		ADNS3080	2	MODUL7, MODUL8	2620	5240
ATMEGA328P		DIP28	1	U1	902	902
CONN	OBSTACLE DETECTOR CONNECTOR	HEADER_1x3_2.54_LOCK	5	P8, P9, P10, P11, P12	11	55
DC CONN	SMD DC BARREL PLUG CONNECTOR	DC barrel plug SMD	1	P15	239	239
ESP32S	ESP32 NodeMCU Microcontroller	ESP32S	1	U4	3670	3670
HEADER		HEADER_1x2_2.54	6	P3, P4, P7, P16, P17, P18		0
HEADER		HEADER_1x3_2.54	1	P14		0
HEADER		HEADER_1x4_2.54	1	P13		0
HEADER		HEADER_1x5_2.54_LOCK	2	P5, P6	16	32
HEADER		HEADER_2x3_2.54	1	P1	27.4	27.4
HEADER		HEADER_2x5_2.54_LOCK	1	P2	377	377
SHIFTER		I2C LVL SHI	1	MODUL9	378	378
IRLML2803TR	SOT23 N-MOSFET	SOT23	6	IC1, IC2, IC3, IC4, IC5, IC6	40.6	243.6
LVL SHIFTER		LVL SHI	2	MODUL5, MODUL6	349	698
MCP23S17		DIP28	1	MCP23S17	684	684
MT3608		STEP UP	1	MODUL4	273	273
ON-OFF-ON		SWITCH 3STATE	1	SW1	208	208
RED	TH LED	LED_5mm	12	9, D20, D21, D22, D23, D24, D25, D	52	624
RED	SMD LED	0805L	4	D1, D2, D3, D4	16.7	66.8
SWICH		DIL6	1	SW2	228	228
TP4056		TP4056	1	MODUL3	613.5	613.5
ULN2003	Darlington array	DIP16	2	U2, U3	85.9	171.8
REGULATOR	Voltage regulator module	HEADER_1x4_2.54	2	MODUL1, MODUL2	372	744
WHITE	TH LED	LED_5mm	4	D5, D6, D9, D12	50.8	203.2
WS2812B		5050SMD	1	U5	107	107
YELLOW	TH LED	LED_5mm	8	7, D8, D10, D11, D13, D14, D17, D1	44.6	356.8
			NON			
HEADER	Single row female headers	1x40	3		113	339
HEADER	Single row male headers	1x40	1		24.8	24.8
OBSTACKLE	Obstackle detector module	-	5		174	870
BATT	Lilon battery	-	2		1257	2514
HOLDER	Lilon battery holder	-	2		195	390
CONN	Barrel plug connector	-	1		32.4	32.4
CONN	2x5 locked ribbon connector	2x5 lock	1		25.4	25.4
CABLE	Ribbon cable	-	1		304	304
HOLDER	DIP16 IC holder	DIP16	2		15.2	30.4
HOLDER	DIP28 IC holder	DIP28	2		17.8	35.6
SPACER	M2 spacer 10mm long	-	5		31	155
SPACER	M2 spacer 3mm long	-	11		18.5	203.5
SERVO	13g 180deg servo	-	2		2413	4826
RFID	RFID reader module	-	1		2165	2165
SCREW	Package (not economical)	-	1		1530	1530
SCREW	Piece by piece (economical)	-	???			
SCREW	M2x10	-	???			
MOTOR	Uniporar stepper motor	-	2		690	1380
WHEEL	Wheel for servo (mounted on stepper)	-	2		590	1180
JUMPER	Jumper connector	1x2	2		4.5	9
						32490.9
PCB	Motherboard	-	1			14000
BODY PARTS	3D printed body parts	-	1			5000
						51490.9

8.2.1. ESP32 MEMÓRIA FELÉPÍTÉSE

- 448 KB ROM bootolásra és alapvető funkciókra
- 520 KB SRAM adatoknak és végrehajtásnak
- 8 KB SRAM az RTC-ben, ennek a neve RTC FAST memória és használható adattárolásra; a fő CPU használja az RTC boot során a mélyalvás módból
- 8 KB SRAM az RTC-ben, ennek a neve RTC SLOW memória és a mellék CPU használja a mélyalvás során.
- 1024 bit eFuse: 256 bitet a rendszer használ (MAC cím és chip beállításra) a maradék 768 bit a felhasználó számára fenntartott, használható flash titkosításra és chip ID beégetésére.

8.2.2. FESZÜLTSÉGSZINTVÁLTÓ MŰKÖDÉSE

A kapcsolás négy ugyan olyan kapcsolásból épül fel. A kapcsolás az alábbi ábrán látható. A működés a következő. Az alacsony oldal, vagyis LV1 magasba húzása esetén a MOSFET lezár és az R4 jelölésű ellenálláson keresztül magas szintre áll be a HV1 vonal. Alacsonyba húzás esetén a MOSFET kinyit és HV1 is alacsony szintre kerül. Amennyiben HV1 magas, a MOSFET ismét lezár és az LV1 oldal LV szintre kerül. Amennyiben HV1 alacsony, a MOSFET-ben található diódán keresztül LV1 is alacsony szintre kerül.



8.2.3. LÉPTETŐMOTOR ADATAI

- Típusjelzés: 28BYJ-48
- Névleges fesz.: 5V
- Fázisszám: 4

- Tekercselés: Unipolar
- Hajtómű áttétele 1/64 (1/ 63.68395 ez alapján: https://www.makerguides.com/28byj-48-stepper-motor-arduino-tutorial/)
- Ajánlott üzemmód: Fél lépés
- Lépés/fordulat: 64 (Ajánlott üzemmódban)
- Sebesség: 15RPM@5V; 24RPM@12V (https://medium.com/jungletronics/28byj-48-stepper-motor-peak-rpm-658eae6afe2f)

8.2.4. MRFC522 PINOUT

Név a modulon	Funkció
SDA	Not Slave Select
SCK	SPI Clock
MOSI	SPI MOSI
MISO	SPI MISO
IRQ	Iterrupt request, adatlapban további részletek
GND	GND
RST	Hard reset és power down, aktív alacsony, minimális fogyasztás
3.3V	Táp