

Robotikus kézfej protézis motorvezérlésének tervezése

Szentmiklósi Máté

Óbudai Egyetem

Bánki Donát Gépész és Biztonságtechnikai Mérnöki Kar, BSc 6. évfolyam

Konzulens: Dr. habil Laufer Edit, egyetemi docens

1. Bevezetés	3
2. Történeti áttekintés	4
3. Architektúrák	5
4. A mikrovezérlők.....	8
4.1 Regiszterek	9
4.2 Memória felépítés	10
4.3 Portok, Port kezelés	11
5. Motorvezérlés	14
6. Rendszerterv.....	16
7. A programkód felépítése	18
8. A kód részletezése	22
9. Eredmények értékelése, továbbfejlesztési lehetőségek.....	30
10. Felhasznált irodalom	31

1. Bevezetés

Amikor látszólag egymástól különböző tudományterületek találkoznak, keletkezik köztük egy híd, ami összekapcsolja őket. A biológiát és a mechatronikát összekapcsoló híd nem más, mint a biomechatronika. Ez a komplex tudományterület tartalmazza a mechatronika építőköveit (gépészet, elektronika, informatika) és a biológia tudásanyagát. Célja, hogy olyan ember számára is normális életet tudjon biztosítani, akiknek egy baleset vagy születési rendellenesség ezt nem tette lehetővé. A biomechanikus eszközök az alkalmazások széles skáláját ölelik fel, a végtagprotézisek fejlesztésétől a légzőrendszer támogatásán keresztül, a látás, valamint a szív- és érrendszer problémáinak műszaki megoldásáig.[9]

Számomra régóta fontos ez a tudományág, ezért csatlakoztam az Óbudai Egyetem Bánki Karán működő Mesterséges Intelligencia Műhely Biomechatronika munkacsoportjához, ahol a csapattagokkal egy mesterséges intelligencia vezérelt alkarprotézisen kezdtünk dolgozni. Az én feladatom a motorok vezérlése lett, amit elkezdtem kidolgozni. Sajnálatos módon a csoport aktivitása időközben alábbhagyott, de az ott kitalált ötleteket tovább vittem és elkészítettem a kéz motorvezérlését.

Dolgozatomban egy kézfej protézis motorvezérlésének a tervezését mutatom be. A dolgozat elején a mikrokontrollerek felépítését tekintem át, mivel ezek szerves részét képezik a protézisnek. Ezután a motorvezérlés alapfogalmainak ismertetése következik, majd az általam fejlesztett programot mutatom be. A kitűzött cél, hogy a program képes legyen úgy vezérelni a motorokat, hogy azok alkalmasak legyenek egy igazi kéz mozgását utánozni. Ez alatt azt értem, hogy a gyorsítás és lassítás szakasznak legyen felfutási ideje, valamint a program futási ideje minél kisebb legyen, hogy ezzel is a valódi, emberi mozgásra hasonlítson.

2. Történeti áttekintés

Napjainkban szinte minden területre beszivárgott a robotika, ez alól az egészségügy sem kivétel. Ma már vannak olyan robotok, akik képesek egy embert megműteni és vannak olyan robotikai eszközök is, amik képesek elvesztett végtagokat pótolni. Ugyanakkor a protézisek nem mai találmányok. A régészeti leletek alapján már az ókori Egyiptomban is használtak protéziseket. Patológus kutatók találtak egy múmiát, aminek a nagylábujja mesterséges volt. Ez a művégtag bőrből és fából készült és lehetővé tette a mezítlábbal való járást, és az Egyiptomban akkortájt használatos szandálok viseletét.[20] Természetesen nem ez az egyetlen eset. A mesterséges végtagok végig kísérik a történelmünket. Találni olyan Római feljegyzést, ami szerint egy római tábornok, bizonyos Marcus Sergius, akinek levágták a jobbkezét, az egyik hadjárat során, ezért fémből készíttetett egyet, ami képes volt tartani a pajzsát és így visszatérhetett a csatamezőre. Később, a középkorban is használták ezt a megoldást a lovagok. A 17. században is alkalmaztak vaskezeket protézis gyanánt, de nem csak kezet, hanem kampót is használtak kézpótlásnak Henri de Tonti volt az a személy, akinek egy kampóprotézise is volt. Az igazi fejlődést a 20. század hozta el. Egy olasz orvos feljegyzéseiben említ egy karját vesztett személyt, aki a protézisével képes volt a kalapját megemelni ajtót nyitni, sőt még aláírni is. Az igazi áttörést az hozta, hogy a második világháború után a NASA elkezdett pénzt fektetni a művégtagok kutatásába.

A 90-es években jelent meg az első mikrokontrollerrel vezérelt műláb, ami természetesebbé tette a járást. 1998-ban adták ki az első olyan térdfeletti protézist, ami hidraulikát és pneumatikát, valamint mikrovezérlőt tartalmazott, ami elősegítette az emberi mozgáshoz még inkább hasonlító megvalósítást. Napjainkban már folynak olyan kutatások is, amik úgynevezett neurorobotic protézisek. Az ilyen fajta műkezek képesek visszacsatolást küldeni az agynak, aminek köszönhetően az érintés is lehetségessé válik.

3. Architektúrák

Dolgozatom elsődleges célja a motorvezérlésre alkalmas program fejlesztése, amihez elengedhetetlen az architektúra, vagyis a számítógép felépítésének ismerete. Gép és gép között is igen nagy eltérés lehetséges, ezért érdemesnek tartom néhány szót ejteni a lehetséges architektúrákról.

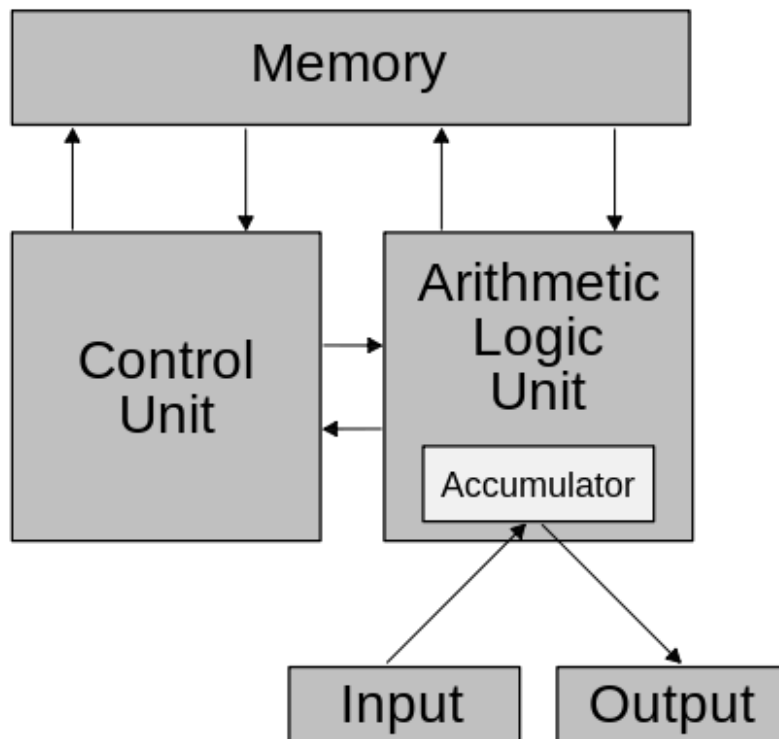
A két leggyakrabban alkalmazott változat a Neumann és a Harvard architektúra. Az alábbiakban ennek a két architektúrának a felépítését mutatom be.

A Neumann architektúra felépítése az alábbi felsorolásban látható, blokk vázlatát pedig az 1. ábra szemlélteti. Ez a felépítés egyszerűbb kialakítású, mint a Harvard architektúra.

A Neumann architektúra elemei:

- központi vezérlőegység (CU)
- Aritmetikai és logikai egység (ALU)
- Memóriák: RAM, ROM, FLASH
- általános felhasználású be-, és kimenetek (GPIO)

- szabványos interfészek

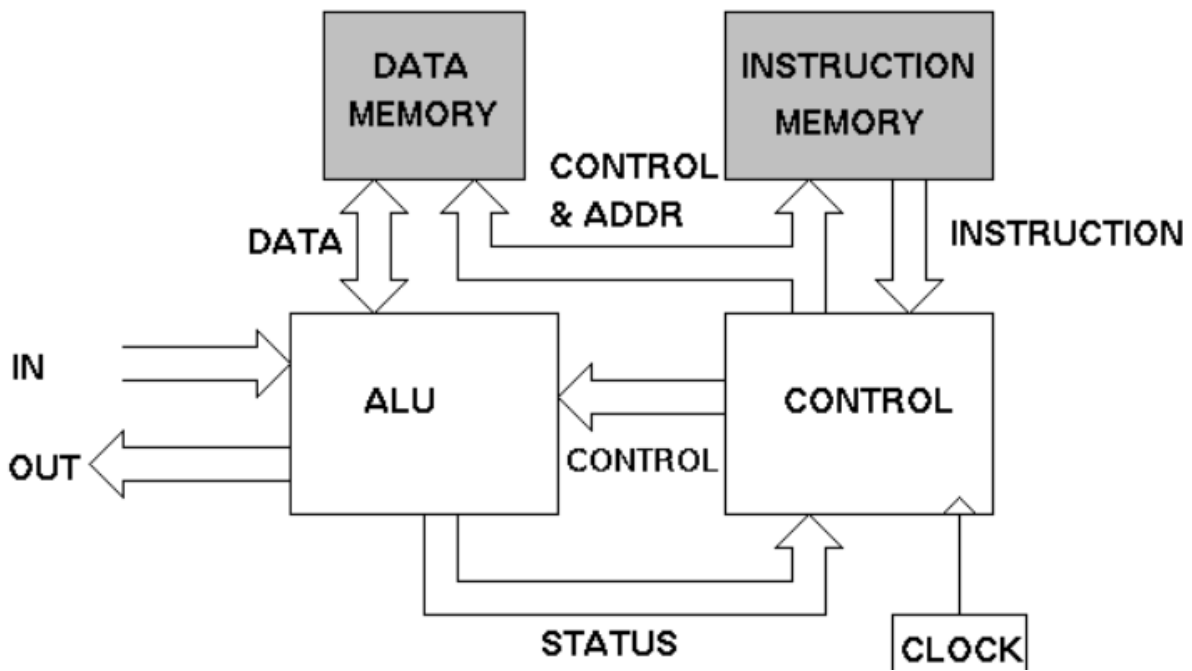


1. ábra Neumann architektúra blokk vázlata[17]

Az ilyen fajta számítógépek egy adatsínt használnak, ami azt eredményezi, hogy az aritmetikai és logikai műveletek egyidejűleg nem végezhetők az adat kiolvasással. Továbbá az adatok és a program tárolása egy helyen, a RAM-ban történik.

A Harvard architektúra felépítését tekintve ugyanazokat az elemeket tartalmazza, mint a fent említett, viszont itt az adat és a program memória fizikailag elkülönül. A memóriákhoz különálló utak vannak biztosítva, így a teljesítménye nő. A címezést tekintve a Harvard

architektúra külön címzi az adat-, és külön a program memóriát. Az architektúra blokk vázlata a 2. ábrán látható.



2. ábra Harvard architektúra blokk vázlata[12]

4. A mikrovezérlők

Mindenképpen szót kell ejteni a mikrovezérlőkről is, hiszen a protézisben ez fogja vezérelni a motorokat. Ide érkeznek be az adatok, itt kerülnek feldolgozásra, és innen küldi tovább az információt a motoroknak. Mondhatni ez lesz az agya a protézisnek. A mikrokontroller vagy más néven mikrovezérlő egy nagy integráltságú áramkör, mely utasításokat olvas be és hajt végre. Felépítésüket tekintve általában Harvard architektúrájúak annak köszönhetően, hogy ha az adatmemória és a program memória egymástól elkülönítve található, és különálló elérési útvonallal rendelkezik. Ebben az esetben elmondható, hogy a mikrovezérlő nagyobb teljesítményre képes.

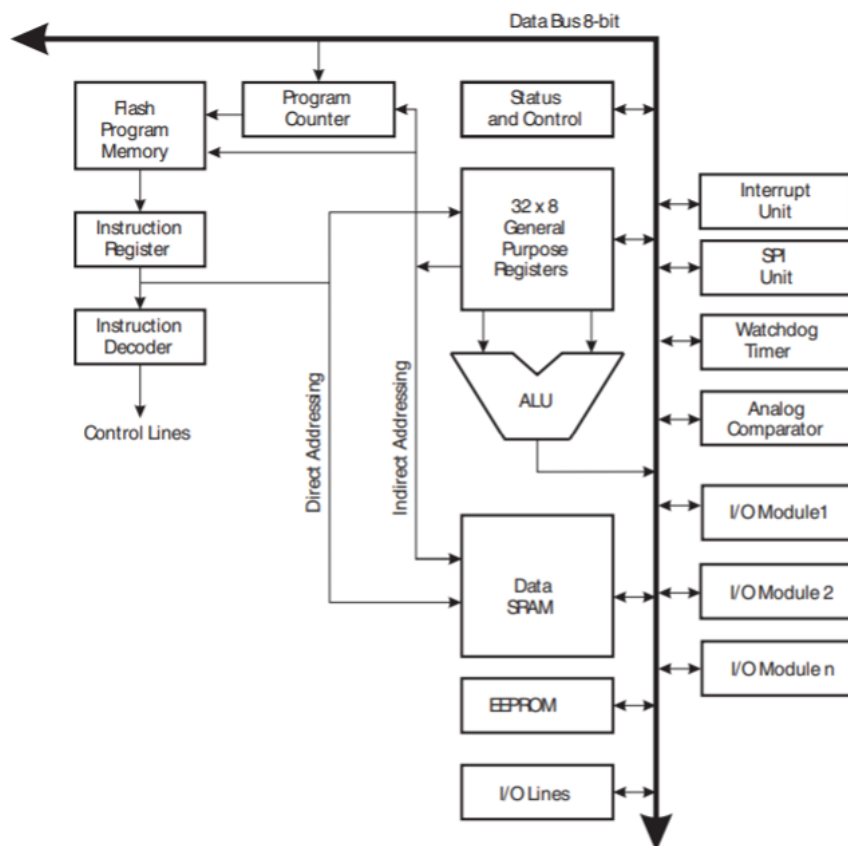
Általánosságban elmondható, hogy egy mikrovezérlő kvázi egy csökkentett utasítás készlettel rendelkező egy tokba integrált mikroszámítógép. Felhasználásukat tekintve általában vezérlési feladatokat látnak el, és leginkább a beágyazott rendszereknél használják.

A mikrokontrollerekből találhatunk 8,16,32 bites kivitelűt. Egy mikrovezérlő tartalmazhat processzormagot, programmemóriát, szabadon hozzáférhető memóriát (RAM), adatmemóriát (EEPROM), órajel generátort, digitális ki-bemeneteket, soros portokat, analóg bemeneteket, PWM kimeneteket, időzítőket, számlálókat, eseményfigyelőket, megszakítási lehetőségeket, megbízhatósági funkciókat (brown-out reset, watchdog timer stb)

A projektben egy ATmel ATmega 2560-as mikrokontrollert használtam. Így az alábbiakban az ATmel családba tartozó ATmega 2560 segítségével mutatom be egy mikrovezérlő főbb részeit, a dolgozat szempontjából kevésbé fontos elemeket nem részletezem.

Ez a verzió egy 8 bites 256kb FLASH memóriával rendelkező Harvard architektúrás mikrokontroller, felépítését a 3. ábra szemlélteti. Fő egységei a következők:

- aritmetikai és logikai egység (ALU) amely az operandusokkal végzi a műveleteket
- általános felhasználású regiszter úgynevezett General Purpose Registers, melyek 8 bitesek és a RAM terület első 32 bájtyát adják.
- DATA SRAM (4/8 kbyte) amelyet egy 0x100 címtől a felhasználó programozhat[5]
- 256kb FLASH memória, ahová az adott felhasználó programjai kerülnek[5]



3. ábra ATmega CPU magja [5]

4.1 Regiszterek

A regiszter olyan tárolóegység, amely gyors hozzáférést biztosít a CPU-nak írásra és olvasásra. Általában egy gépi szó feldolgozására alkalmas egyidőben, és ezek az adatok ideiglenes tartalmúak. Elhelyezkedését nézve az SRAM-ban található. Két fő részre osztható:

- általános felhasználású regiszter
- speciális felhasználású regiszter

Az általános felhasználású regiszterek, mint ahogy a nevük is mutatja, általános célokra vannak fent tartva.

A mikrokontrollerben 32 darab 8 bites általános regiszter található, ebből:

- Egy 8 bites kimeneti operandus és egy 8 bites eredménybemenet [5]
- Két 8 bites kimeneti operandus és egy 8 bites eredménybemenet[5]
- Két 8 bites kimeneti operandus és egy 16 bites eredménybemenet[5]
- Egy 16 bites kimeneti operandus és egy 16 bites eredménybemenet[5]

A regiszter fájlban működő utasítások többsége közvetlen hozzáféréssel rendelkezik az összes regiszterhez, és többségük egyciklusú utasítás. Minden regiszterhez adatmemória-címet is hozzárendelnek, amely mindjárt a felhasználói adattér első 32 helyére van leképezve.

A speciális felhasználású regiszterek többek között a következők lehetnek: veremmutató, utasítás mutató, programszámláló, státusz regiszter és kontrol regiszter.

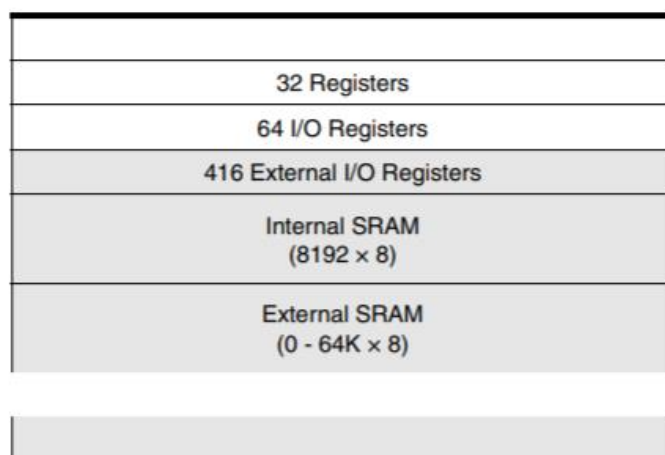
A veremmutató-regiszter mindig a verem tetejére mutat. A verem elsősorban ideiglenes adatok -, helyi változók tárolására és visszatérési címek megőrzésére szolgál. A verem a magasabb memóiahelyekről az alacsonyabb memóiahelyek felé növekszik. A veremmutató arra az adat-SRAM veremterületre mutat, ahol a szubrutin és a megszakítási verem található. A programnak meg kell határoznia az SRAM-ban lévő veremterületet, mielőtt bármilyen szubrutinhívást végrehajtana, vagy megszakításokat engedélyezne.[5]

Az utasításmutató/ programszámláló a soron következő utasítás címét tartalmazza. Értékét a dekóder adja az utasítás dekódolása után vagy értéket adhat a szubrutinhívás, megszakítási rutin hívás, elágazó utasítás.

4.2 Memória felépítés

A memóriát két fő részre osztjuk: adatmemóriára és a programmemóriára. Ezen kívül EEPROM memóriával is rendelkezik az a mikrokontroller, amelyet használtam. Mindhárom memória terület lineáris és szabályos.

Adatmemóriában az első 32 hely a Regiszterfájl, a következő 64 a szabványos I/O memóriát tartalmazza, majd a kiterjesztett I/O memória következik 416 helyen keresztül, a következő 8192 helyet pedig a felhasználói programok használhatják. A verem terület az SRAM terület végétől építkezik visszafelé, ahogy ez a 4. ábrán is látható.



4. ábra Adat memória felépítése [5]

Az általam használt Atmega 2560 256 Kb-át újraírható Flash memóriát tartalmaz a programtároláshoz. Mivel minden AVR utasítás 16 bites vagy 32 bites, a Flash $128K \times 16$ -ként van megszervezve. A szoftverbiztonság érdekében a Flash program memóriaterülete két részre oszlik, a Boot Program szakaszra és az Alkalmazásprogram részre. A Flash memória legalább 10 000 írási/törlési ciklust képes elviselni.

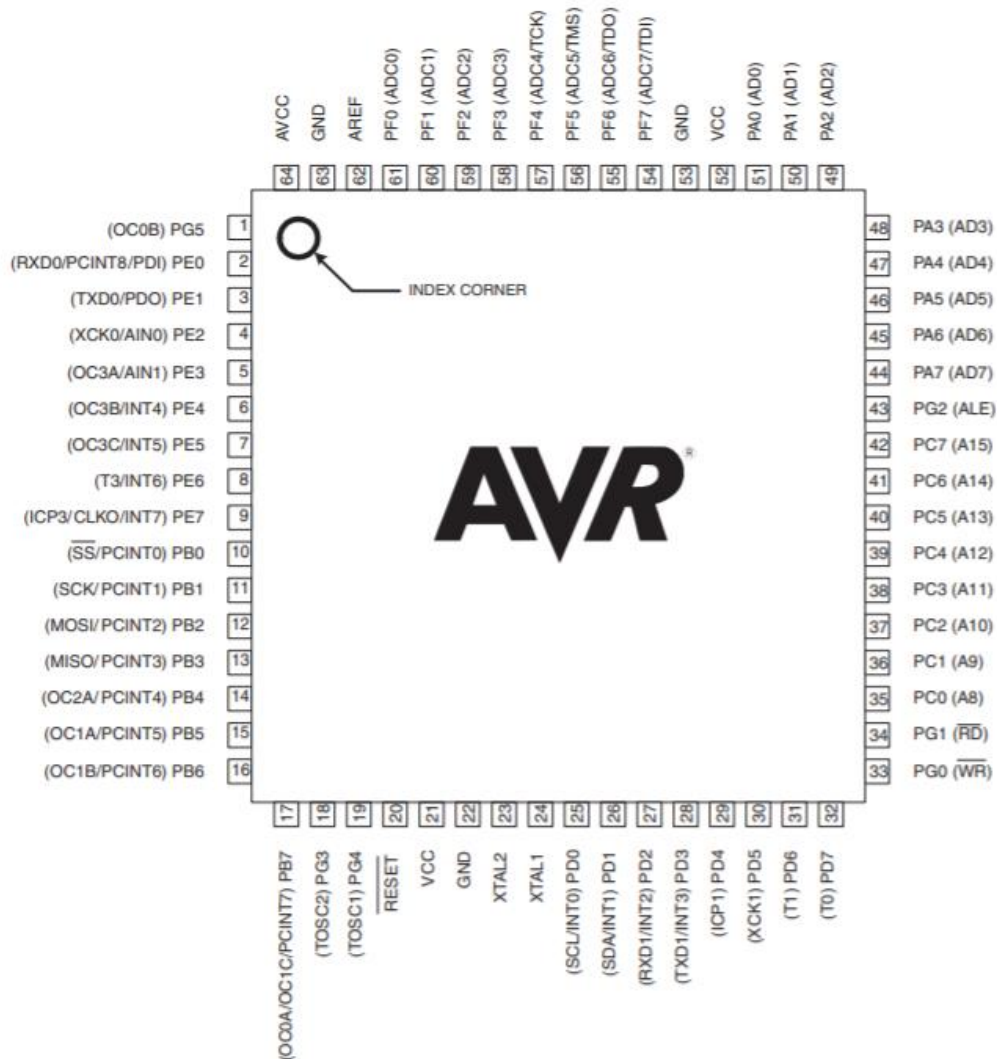
4.3 Portok, Port kezelés

A mikrokontrollerek különféle kimeneti és bemeneti portokkal rendelkeznek attól függően, hogy melyik gyártó melyik típusát választja a felhasználó. Én egy ATmega 2560 típusú AVR mikrovezérlőt használtam, ami 86 I/O lábbal rendelkezik.

Ezek a pinok 11 portra lettek szétosztva A-tól K-ig vannak csoportosítva. Némely csoport között alig találni eltérést, míg mások között jelentős az eltérés. De nem csak I/O portok találhatók itt, hanem a működésért felelős pinok is, mint például az 1. táblázatban felsoroltak:

Láb neve	Feladata
VCC	pozitív tápfeszültség
GND	földelés
RESET	reset bemenet
XTAL 1-2	külső kvarc ki-, és bemenet
AVCC	analóg rész pozitív tápfeszültsége
AREF	analóg rész föld

1. táblázat Lábak megnevezése és hozzá tartozó funkciója [3]



5. ábra ATmega 128 lábkiosztása [5]

Mint az 5. ábrán is látható, némely láb mellett zárójelben, egy úgynevezett másodlagos funkció található, melyek közül a legfontosabbakat a 2. táblázatban ismertetem.

Megnevezés	feladata
RXTx/TXDx	kommunikációs port
AINx	ADC analógbemenetei
OCx	kimenet összehasonlítása és PWM kimenet, időzítő/számláló vagy pin megszakító
SCL/SDA	busz adatvonal TWI
INTx	külső megszakító bemenet
TCK,TMS,TDI,TDO	JTAG programozó adatvonalai
MOSI,MISO,SCK,SS	SPI busz adatvonalai
ADx	külső memória interfész cím és adat bitje

2. táblázat Lábak másodlagos funkciói

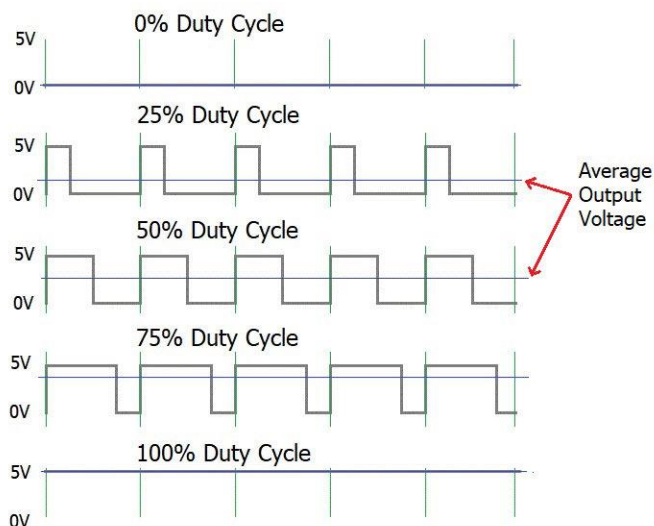
Ahhoz, hogy a ki-,és bemenetek funkciói közötti váltást szoftveresen tudjuk megoldani, először is egy Header fájl, vagyis az Arduino.h meghívásával kell kezdeni, ami meg fogja hívni az avr/io.h-t. Ez a fájl tartalmazza az AVR belső működésének megfelelően definiált kifejezéseket, vagyis a ki-, és bemenet beállítását, melyet a portnál a DDRx szó segítségével lehet beállítani, ahol az x a port nevét takarja. Például a D port kimenet beállítását a `DDRD |= 0b1111000` paranccsal lehet megoldani. Ha az első 4 bit logikai 1-es és a második 4 bit logikai 0, akkor kimenetre; ha fordítva, akkor bementre állítottuk az adott portot. [3]

5. Motorvezérlés

A vezérlés olyan jellemzően nyitott hatásláncú irányítás, amelyben a beavatkozás jellemző értékektől függetlenül külső jel vagy jelek hatására jön létre, és a hatása nem hat vissza a vezérlő berendezésre.

Az egyenáramú motorok sebességét a feszültséggel, a nyomatékukat az áramerősséggel, az irányukat pedig a polaritás váltással lehet kontrolálni. A motorokra eső feszültség nagyságát a PWM jel kitöltési tényezője adja meg. A PWM jel egy olyan nem szabályos négyszög jel, aminek az aktív és inaktív tartományait mi adhatjuk meg egy hányados segítségével.

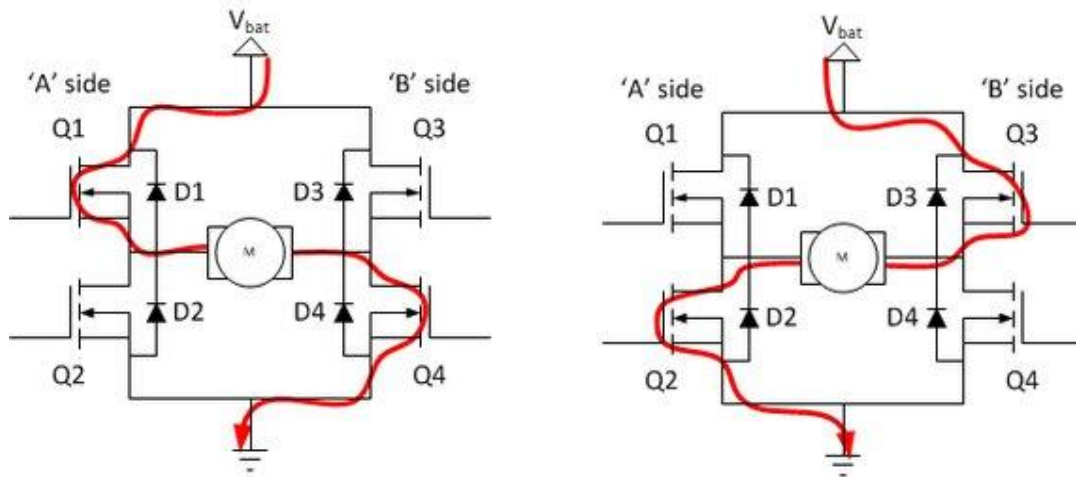
Ennek az aránynak köszönhetően a feszültség átlaga szabályozható, ami azt eredményezi, hogy



6. ábra PWM jel szemléltetése[19]

egy 0-5V-os feszültség szintet kedvünk szerint változtathatunk. A mikrokontrolleren ezt az arányt 0-255-ig választhatjuk meg, ahol a 255 a 100%-os kitöltés.

A polaritásváltás egy külső kapcsolással oldható meg, úgynevezett H-híd kapcsolással, ahol négy kapcsoló van vezérelve egy mikrokontrollerrel. A kapcsolók lehetnek tranzisztorok, relék, MOSFET-ek, vagy motormeghajtó IC. A H-híd működését a 7. ábra szemlélteti.



7. ábra H-híd működése[14]

Ha a Q1-es kapcsoló aktív, akkor a Q4-esnek is aktívnak kell lennie és így balról jobbra folyik az áram. Ha Q2 aktív, akkor annak a párja a Q3 és így megfordul a folyásirány és ezzel együtt a forgásirány is.

A program írása során a fentiekben ismertetett elméleti követelményeket vettem figyelembe.

6. Rendszerterv

A célkitűzésben szereplő program feladata, hogy 5db DC motort vezéreljen a lehető legrövidebb futási idő alatt, és mivel egy protézisnek készül, ezért a motorok gyorsítása és lassítása minél inkább természetes kell, hogy legyen. Fontos követelmény az is, hogy a motorok egyszerre és külön is működtethetők legyenek, akár különböző forgásiránnyal, vagy csoportosítva, tehát két motor órajárásnak megfelelően forog, három másik, órajárásnak ellentétesen. Azt a feladatot is el kell látnia, hogy a motorok nem ugyanannyit fordulnak, tehát az egyik később, a másik előbb áll meg, mint a többi. Ezekre a feladatokra azért kell, megfeleljen mert nem sima micro servo motorokat használok mivel sok helyet foglalnának.

Egy könnyen elkészíthető kézprotézis általában mikró szervomotorokkal készül. Ennek egyik hátránya, hogy sok helyet foglal. Erre megoldást jelenthet, ha a szervomotorok helyett DC motort és potenciómétert használunk. Ez több problémát is felvet, mint pl.: a fogaskerék áttétel, illetve a kábelkezelés, de összességében nézve egy helytakarékosabb megoldásnak látszik a többséghez képest, a forgási szögek intervallumai pedig az általunk tervezett fogaskerekektől függenek.

Annak érdekében, hogy egy sima egyenáramú motor képes legyen a pozicionálásra, szükség van többek között egy mikrokontrollerre, illetve egy motorvezérlésre.

A dolgozatomban egy ilyen felépítésű vezérlést hoztam létre. Törekedtem arra, hogy n db motorral, illetve potencióméterrel működjön. Az egyetlen korlátozó tényező a portok száma. Cél egy olyan program létrehozása volt, ami képes „egyszerre” több motort a megfelelő irányba, a megfelelő ideig működtetni.

A fent említett elemek közül választottam alkatrészeket, de ezek az építőelemek a jelenlegi információknak megfelelően lettek kiválasztva, ami azt jelenti, hogy a megvalósítás közben felmerülő problémákra válaszul más elemeket kell választani. Potméternek egy Piher PT15 NV 10K-t választottam. A vele szemben támasztott elsődleges követelmény a méretre vonatkozott, ez a darab egy 15mm-es külső átmérővel bír, ami éppen megfelelő. Motornak egy 8520 Coreless Motor tűnt a leginkább megfelelőnek, ami méreteit tekintve igen kicsi, átmérője mindössze 8mm; hossza 20mm. A rákapcsolható feszültség DC 1,5-4,2 V. A választás többek között a nagy fordulatszáma miatt esett rá, ami az 50000 fordulat/percet is elérheti. A következő elemek a fogaskerekek, amelyeknél a fogaskerék áttételek a legfontosabb jellemzők közé tartoznak. A fogaskerekek áttételek a következők: 11:61; 12:48; 13:47; 13:42, melyekkel képes

egy 259 fordulat/perc és 1Nm forgatónyomatékot 1 fordulat/percre és 259 Nm forgatónyomatékra átalakítani.

A következő részekben először a feladat matematikai hátterét ismertetem, majd az általam fejlesztett program részeit mutatom be.

7. A programkód felépítése

Az elméleti háttér és a rendszerterv ismertetése után rátérek a TDK fő részét képező feladatomra, a vezérlőprogramra.

Mint a bevezetésben már említettem, fő feladatom egy alkarprotézis motorvezérlésének megvalósítása volt.

A PWM jel 0 és 255 közötti értékvételét nem egy lineáris egyenlettel, hanem egy exponenciális egyenlettel oldottam meg, ezzel emberibbé téve az ujjak mozgását.

Az egyenlet a következő:

$$t = \frac{max-min}{n} \quad (1)$$

ahol max a maximális futási idő, és $max \neq 0$, min a kezdési idő, $n \in \mathbb{N}$, $t \in \mathbb{R}^+$

$$f = min + t * x \quad (2)$$

ahol $x \in \mathbb{N}$ és $x \rightarrow n$, $f \in \mathbb{R}^+$

$$y_1 = \frac{1}{1+e^{-c_0*(f-c_1)}} \quad (3)$$

$$y_2 = \frac{1}{1+e^{-c_{02}*(f-c_2)}} \quad (4)$$

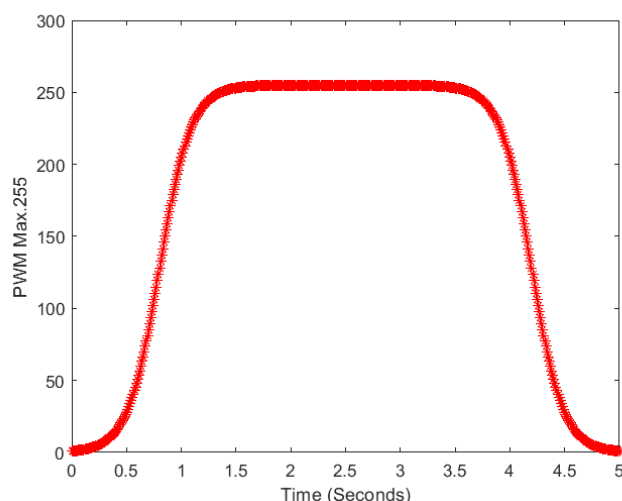
ahol $c_0, c_{02} \in \mathbb{Z}^+$ ők felelnek a lassulás és gyorsuló szakasz meredekségéért

$c_1, c_2 \in \mathbb{R}^+$ és $c_1 < c_2$ ezek az értékek adják meg a szakasz középpontot.

$$y = |y_1 - y_2| * 255 \quad (5)$$

ahol $y \in \mathbb{R}^+$. Az $y_1; y_2$ behelyettesítve a következő egyenletet kapjuk.

$$y = \left| \frac{1}{1+e^{-c_0*(f-c_1)}} - \frac{1}{1+e^{-c_{02}*(f-c_2)}} \right| * 255 \quad (6)$$



8. ábra y ábrázolása

A 8. ábrán jól látható, hogy 3 különálló részre oszthatjuk a mozgást, az első szakasz a gyorsulás, majd egy egyenletes mozgás következik, ezek után pedig a lassítás jön. A programban ezeket a szakaszokat 2 metódussal oldottam meg. A gyorsuló szakasz, ahol a fent említett egyenletek segítségével 0 és 255 közötti PWM kitöltési tényezőnek megfelelő feszültséget ad ki az adott port(ok).

Ezen szakasz vége 255-ös kitöltési érték, ami már az egyenletes mozgó szakasz is, így itt a ciklusvázoló értékét mindig visszaállítottam eggyel, ezáltal a ciklusból csak egy feltétel segítségével lehetett kilépni. A második szakasz a lassítás, ahol az adott portot/portokat a (6)-os képletnek megfelelően lassítom le.

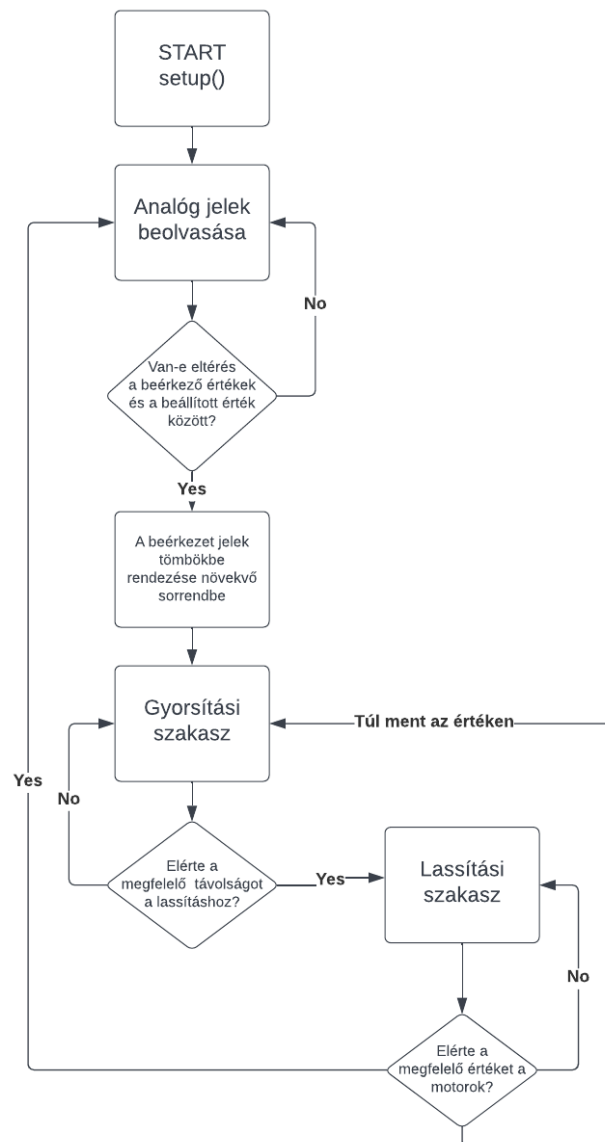
Mivel az ujjak helyzetét a program a potenciométerek segítségével kapja meg, így az analóg bemenetek folytonos figyelésével tudjuk, hogy pontosan mikor ér a helyére az ujj.

Viszont amikor egyszerre szerettem volna megnézni az összes ujj helyzetét és ezek változását feljegyezni, a program (vagy a hardver) nem volt képes ezt megtenni. De nem is kell az összes analóg jelét folyamatosan monitorozni, mivel elég, ha csak azt a portot figyeljük amelyik a legkevésbé tér el a kívánt jeltől, mert mindegyik motor legalább annyit fordulni fog, mint a megfigyelt. Ennek a módszernek elengedhetetlen része, hogy összegyűjtjük azokat a beérkező jeleket, amelyek eltérnek az általunk megadott jeltől, majd növekvő sorrendbe rendezzük.

Ezek alapján a program felosztható 3 főbb részre és több kiegészítő metódusra, amiről a későbbiekben szó lesz:

1. rész analóg jelek beolvasása majd ezek rendezése.

2. rész a kapott értékeknek megfelelően gyorsító és állandó sebességet végző rész. Amint a legrövidebb utat megtevő ujj megközelíti a cél helyét, jön a 3. rész
3. rész a lassítás, ahol lelassítjuk a megfelelő ujjat/ujjakat. Mivel a megállás időbe telik, úgy ahogy a 8.ábrán is látszik, ezért elő fordulhat az a helyzet, hogy akad olyan ujj, amely a legrövidebb értékhez közel áll és belesik a lassításba, ami azt jelenti, hogy nem várhatja meg azt, míg megáll az első, aztán ő. Éppen ezért egy késleltető funkciót vezettem be, erről később részletesebben beszélek.



9. ábra a program leegyszerűsített blokk vázlata

Mint látható, a kód egyszerű felépítésű, de ennek működéséhez a kiegészítő metódusok elengedhetetlenek. A következő fejezetben a fő, illetve mellék metódusokat fogom bővebben kifejteni.

8. A kód részletezése

A programot a header fájlok meghívásával kezdem. Meghívtam a `stdio.h`, `math.h` fájlokat. Majd a `define`-ok következnek: a motorvezérlő, a H híd-vezérlő és az analóg bemenetek kaptak „becenevet”. Ezek után létrehoztam néhány globális változót azért, hogy minden metódus könnyen hozzáférjen és módosíthassa. Többek között itt találhatók azok az értékek, amelyek szerint szeretnénk mozogni az ujjakat, és az aktuális értékek is, ahol jelenleg áll a feszültségosztó.

Az analóg jelek feldolgozását a `potmeter()` metódus végzi, amelyben egy „`analogRead()`” parancs segítségével olvasom be az aktuális értéket, majd csupán a könnyebb kezelhetőség kedvéért egy „`map()`” függvény segítségével áttanszformálok egy 0 és 270 közötti értékre. Az intervallum megválasztása annak köszönhető, hogy a potenciométer, amit használtam, ekkora szögben képes fordulni. Annak érdekében, hogy mindegyik analóg portot olvasni tudjuk egy `switch case` elágazás alkalmazására van szükség. Az 10. ábrán látható a fent említett program részlet, a `case` címeket az analóg porthoz „tartozó” digitális port sorszáma adja.

```

37
38 static int potmeter(byte a){
39     int pot;
40     switch(a) {
41     case 2: pot=analogRead(mpoti); break;
42     case 3: pot=analogRead(kpoti); break;
43     case 4: pot=analogRead(gpoti); break;
44     case 5: pot=analogRead(kipoti); break;
45     case 6: pot=analogRead(hpoti); break;
46
47     }
48     pot=map(pot,0,1023,0,270);
49     return pot;
50 }

```

10. ábra potenciométert beolvasó kód rész

Miután az előző programegységből megkaptuk a motorok állását, már össze tudjuk vetni az általunk kívánt értékkel. Ez történik a `loop` segédmetódusban, amely a megadott értékekkel történő összehasonlítást végzi egy `for` ciklus segítségével, egyesével végig nézve a beérkező és feldolgozott jeleket. Az összehasonlítás során az adott értéktől való eltérést vesszük figyelembe. Ha az eltérés nulla, a beérkező jel megegyezik a kívánt értékkel, ha eltérés tapasztalható, akkor tudjuk, hogy nincs a megfelelő helyen az ujj. A feltételnél azért nem nulla áll, hogy a

feszültségosztón történt esetlegesen pontatlan beállítások is kezelhetők legyenek egy tűrést megengedve. Ha az eltérés a határon belül van, akkor úgy kezeli, hogy a helyén van az ujj, viszont, ha nem, elindul a gyorsító metódus, ami az egyik fő metódus, a sebmot(), ahol a motorok gyorsítása történik.

```

556 int TZtomb[5]={tz,tz1,tz2,tz3,tz4};
557 void loop() {
558   byte var=0;
559   for(int i=2;i<7;i++)
560   {
561     if(abs(potmeter(i)-TZtomb[i-2])>=kulombseg)
562     {
563       var++;
564       sebmot();
565     }
566     else if(var==0)
567       Serial.println("ok");
568   }
569 }

```

11. ábra beérkező és megadott értéket összehasonlító kód rész

A „sebmot()” elnevezésű részben először is meghívásra kerül két másik metódus: először az „elso()” elnevezésű, ahol a „potmeter()” metódus segítségével átírjuk azon globális változókat, melyek a potenciométer állását tárolják; a második metódus a „tomb()”, itt történik a jelek válogatása/szortírozása. Első lépésben azt nézi meg ez a rész, hogy az általunk kívánt érték és a valós érték különbsége negatív vagy pozitív, hiszen ez alapján tudjuk vezérelni a motorok forgásirányát. Ha pozitív, akkor órajárásnak megfelelően forog, ha negatív, akkor ellentétesen. Ezt a műveletet két lépésből oldottam meg, először megszámláltam, hogy hány pozitív és negatív érték van, majd a kapott értékeknek megfelelően létrehoztam egy-egy tömböt, amelyekben eltároltam a különbségeket. Majd a tömbök tartalmát egy buborék rendezés segítségével növekvő sorrendbe rendezem. Erre azért van szükség, mert a program írásakor először az adott metódus egy időben egy portot kezel, és ezt a metódust hívtam meg a loop függvényben minden egyes porthoz. Ez azt eredményezte, hogy egyszerre csak egy motor működött annak ellenére, hogy egyik globális érték sem egyezett. Később felismertem, hogy a mozgó ujjak minimum annyi utat megtesznek, amennyit a legkevesebb utat megtevő ujj, ezért volt szükség a tömbök létrehozására, illetve azok rendezésére. Azonban a két különálló tömb segítségével nem biztosítható a megfelelő működés, mivel a kezelésük azt fogja eredményezni, hogy két időben és orientációban elkülöníthető mozgás jön létre. Tehát egyes motorok az órajárásnak megfelelően mozognak, a maradék pedig órajárással ellentétesen. Annak érdekében, hogy ezt kiküszöböljük, az értékeket a tomb() metódus egyik része segítségével egy

közös tömbbe kell összefuttatni a tömbökben szereplő értékek abszolút értékét figyelembe véve, de a közös tömbbe már előjelesen kerülnek be ezek az értékek.

```

91 void tomb(int c[]) {
92     int A[5]={memory1,memory2,memory3,memory4,memory5};
93     int B[5]={tz,tz1,tz2,tz3,tz4};
94     db=0;
95     db2=0;
96     int minusz=kulombseg*-1;
97
98     for(int i=0;i<5;i++){
99         if((A[i]-B[i])>=kulombseg){
100             db++;
101         }
102         else if((A[i]-B[i])<=minusz){
103             db2++;
104         }
105     }
106     db3=0;
107     db3=db+db2;
108     int c[db];
109     int d[db2];
110     for(int i=0;i<db3;i++){
111         if((A[i]-B[i])>=kulombseg){
112             c[i]=A[i]-B[i];
113         }
114         else if((A[i]-B[i])<=minusz){

```

12. ábra negatív és pozitív értéket számláló rész

Az eltérések megfelelő sorrendbe rendezése után a hozzájuk tartozó kimenetek előállítása következik. Ezt úgy oldottam meg, hogy egy for ciklus és öt elágazás segítségével egy tömbbe gyűjtöttem a különbségekhez tartozó portokat (ez lett a porttomb[]). Az elágazások feltételei igen egyszerűek : HA a különbség egyenlő egy globális változópár különbségével, akkor a párhoz tartozó port érték bekerül a porttomb-be. A globális változópár egy potenciométer érték és egy úticél érték, a hozzájuk tartozó kimenetet már a potmeter() metódusnál kialakítottam.

```

363
364 int zh=C4[0];
365 byte porttomb[db3];
366 for(int i=0;i<db3;i++)
367 {porttomb[i]=0;}
368 for(int i=0;i<db3;i++)
369 {
370     if(C4[i]==(memory1-tz))
371     {porttomb[i]=2; C2[i]=memory1;C3[i]=tz;}
372     else if(C4[i]==(memory2-tz1))
373     {porttomb[i]=3;C2[i]=memory2;C3[i]=tz1;}
374     else if(C4[i]==(memory3-tz2))
375     {porttomb[i]=4;C2[i]=memory3;C3[i]=tz2;}
376     else if(C4[i]==(memory4-tz3))
377     {porttomb[i]=5;C2[i]=memory4;C3[i]=tz3;}
378     else if(C4[i]==(memory5-tz4))
379     {porttomb[i]=6;C2[i]=memory5;C3[i]=tz4;}
380 }
381

```

13. ábra "porttomb" és az beérkezett értékek össze hangolása

Ha két vagy több globális pár különbsége megegyezik, akkor a fent említett port tömb értékei megegyeznek, ami azt fogja eredményezni, hogy néhány kimenet nem fog működni. A probléma kiküszöbölésére a következő algoritmust dolgoztam ki. Két egymásba ágyazott for ciklust alkalmaztam. A külső ciklus ciklusváltozójának értéke a portok számával egyezik meg, azaz 2- 6-ig megy, míg a belső a porttomb méretének megfelelően pl.: egy ötelemű tömb esetén 0-4-ig. Majd egy feltétel minden egyes értéket összevet a külső ciklus változójának értékével. Ha minden jól megy, alaphoz egyszer meg kell, hogy egyezzen a tömbérték és a ciklusváltozó értéke, ezt az értéket egy változóban tárolom és az 1-től vett eltérést figyelem. Ha a változó kisebb, mint 1, akkor hiányzik az adott ciklusérték, ha nagyobb, mint 1, akkor abból az értékből van több pl.: a külső ciklus (a) értéke 2. A belső ciklus végigmegy és összehasonlítja az értékeket. A zárójelben jelöltem azt az értéket, amelyik kéne, hogy legyen (3.táblázat).

	2
	2
	2
	5
	5
	2
	3
	4
	5
	6

14. ábra hibás és helyes értékek

Jól látható, hogy az első port többször szerepel, ami azt jelenti, hogy megvan az egyik érték, amit többször olvasott be a gép. A belső ciklus ciklusváltozójának értékét egy tömbbe

lelmentem, mert ez fogja megmondani, hogy hol van a többlet, majd csökkenteni kell eggyel annak a változónak az értékét, amelyik segítségével tudjuk, hogy többlet van. A csökkentésre azért van szükség, mert már tudjuk, hogy melyik érték van meg többször, de ezen kívül még lehet több ilyen eset. A következő ciklus (b) az összehasonlító érték 3 és mivel rosszul lett beolvasva az érték, ezért a változó nullát fog adni, ami azt jelenti, hogy ez az egyik érték, ami hiányzik. Ezt az értéket is le kell menteni egy tömbbe, hogy tudjuk, hogy milyen értéket kell majd beírni.

a,

2 = 2
2 = 2(3)
2 = 2(4)
2 = 5
2 = 5(6)

b,

3 = 2
3 = 2(3)
3 = 2(4)
3 = 5
3 = 5(6)

3. táblázat hibajavító algoritmus működését szemlélteti

A következő ciklusok a fent leírtak alapján végzik a dolgukat.

```

437
438     if(ertektomb[0]>1&&msc>=1&&hely!=0&&ert!=0)
439     {
440         for(int i=0;i<hely;i++)
441         {
442             for(int j=0;j<ert;j++){
443                 if(ertektomb[j]<7&&ertektomb[j]>1&&helytomb[i]<5){
444                     byte ft=helytomb[i];
445                     porttomb[ft]=ertektomb[i];
446                 }
447             }
448         }
449     }
450 }
451

```

15. ábra "porttomb" értékeit korrigáló rész

Az értékeket és a helyeket tároló tömbök nagyságát a fent említett módhoz hasonlóan oldottam meg. Annak érdekében, hogy a „porttomb” értékei valós adatokat mutassanak, egy for ciklus segítségével a „helytomb” értékének megfelelő helyre az érték tömb értékét beírtam.

Sajnos ez az algoritmus csak teljes tömböknél működik, azaz egy akkora férőhelyes tömbnél, ahány port van deklarálva. Előfordulhat azonban olyan eset, hogy csak 2 vagy 3 ujj akar ugyanarra az értékre menni. Ezt a következőképpen lehet megoldani egy az előzőhöz nagyon hasonló megoldással. Itt a for ciklusok a tömbök értékein mennek végig. Az egyik a

„helytomb” értékein, a másik a „porttomb” értékei szerint. A ciklus belsejében egy feltétel található, ami összehasonlítja „potmeter” metódus segítségével az érték tömb és a porttomb értékeit. Amelyik megegyezik, azt kell kicserélni valamelyik portomb értékével azok közül, amelyek többször szerepelnek. A tömbbe mindegy, hogy melyik helyére lesz átírva hisz az úti cél egyezik.

A következő lépés a H-hídat működtető pinek kezelése. Ezt úgy oldottam meg, hogy létrehoztam egy logikai értékeket tartalmazó tömböt majd egy porttomb-höz hasonló tömböt is deklaráltam, ami a H-híd port-értékeket megfelelő sorrendben tárolja. Az előbb említett tömbben a sorba rendezést a porttomb sorrend kialakító fázisnál alkalmazott metódushoz hasonló módszerrel oldottam meg, de itt nem a különbségeket, hanem már a végleges tömböt használtam. Egy for ciklusba ágyazott switch case elágazást használva feltöltöttem a logikai értékekkel a H-híd érték tömböt aszerint, hogy ha nullánál nagyobb, akkor 1 kerül a tömbbe, ha kisebb, akkor 0. Ezt követően egy for ciklus segítségével aktívvá tettem ezeket a pineket.

```

477 }
478
479 for(int i=0;i<db3;i++)//Híd tömb sorrendbe helyezése
480 {
481     if(porttomb[i]==2)
482     {Hhdtomb[i]=elorem;Hhdtomb[i+1]=hatram;}
483     else if(porttomb[i]==3)
484     {Hhdtomb[i]=elorek;Hhdtomb[i+1]=hatrak;}
485     else if(porttomb[i]==4)
486     {Hhdtomb[i]=eloreg;Hhdtomb[i+1]=hatrag;}
487     else if(porttomb[i]==5)
488     {Hhdtomb[i]=eloreki;Hhdtomb[i+1]=hatraki;}
489     else if(porttomb[i]==6)
490     {Hhdtomb[i]=eloreh;Hhdtomb[i+1]=hatrah;}
491 }
492
518
519 byte p=0;
520 for(int jk=0;jk<db3;jk++)//Híd porttomb érték adása
521 {
522     if(C4[jk]>0)
523     {
524         rt[p]=1;
525         p++;
526         rt[p]=0;
527         p++;
528     }
529     else if(C4[jk]<0)
530     {
531         rt[p]=0;
532         p++;
533         rt[p]=1;
534         p++;
535     }
536 }
537
538

```

16. ábra H-híd tömb érték sorba rendezés

A portok előkészítése után a következő lépés a motorgyorsítás, amely a „A program kód felépítése” részben tárgyalt függvény alapján működik. A változók deklarálása után egy for ciklus segítségével a (6)-os képletnek megfelelően egytől 456-ig gyorsul. A felső értékhatár azért 456, mert ezt az értéket behelyettesítve a képlet eredménye 255 lesz, ami a maximálisan előállítható érték. Ezt követően egy if feltétel segítségével megszakítottam a ciklus futását. Az if elágazás feltétele, hogy a futó pont legyen 455 és a porttomb első tagjának az analóg bemenet értéke legyen a kívánt célérték $\pm$ az az érték amennyit forgat a motorlassulás alatt. Ha ezek a feltételek teljesülnek, akkor a lassítás fázis következik, ha nem, akkor a futó pont értéke legyen 454, ami azt jelenti, hogy a kitöltési tényező 255.

A következő nagy rész a programban a lassítás. Ebbe a szakaszba belépve először deklarálni kell azokat a változókat, amik szükségesek a változók a lassuláshoz. Ezek után egy ciklus és egy feltétel segítségével megszámlálom, hogy hány olyan érték van, aminek az értéke nagyobb, mint a lassuláshoz szükséges érték. Ez alapján tudjuk, hogy melyik PWM jelet adó kimenet fog változni és mely kimenetek nem. A lassítás megkezdése után két opció lehetséges:

a) azonos helyről megegyező helyre kívánnak jutni, b) kis eltérés mutatkozik a kívánt célhelyek között és/vagy a kiindulási hely között.

Az „a” eshetőség az egyszerűbb így azzal kezdem. Itt egy for ciklus végigmegy a megállni kívánó portokon és az egyenletnek megfelelően megállnak. Amelyik pin nem szeretne megállni annak a kitöltési tényezője 255.

A „b” eshetőségnél a mechanikából ismeretes $\frac{a}{2}t^2 = s$ képletet alkalmazzuk a t értéke egy „millis()” függvény segítségével megmérhető, hogy mennyi ideig fut a lassítás fő ciklusa és az s , ami a különbség lesz. Ezek után „a”-ra kell rendezni a képletet, hogy ki lehessen számolni azt az értéket, amennyivel több kell, legyen az y értéke. Azért nem delay-t használok, mert az az egész programot várakoztatja x idővel, de így csak néhány ciklus ideig lesz feltartóztatva a program futása.

A gyorsítási szakaszhoz hasonlóan itt is található kilépési feltétel, amelyet három elágazás segítségével adok meg. Az első az az eset, amikor a ciklus lefutott, de nem érte el a rendeltetési helyet. Ilyenkor újra lemérjük a potméterek állását és újra rendezzük a tömböt, majd ismét elkezdjük a lassítást. A második eset, amikor valamilyen oknál fogva a potenciométer túlfordul. Ekkor meghívásra kerül a gyorsító szakasz, mivel már nincs a kívánt érték közelében. A harmadik eset az, amikor még nem futott le a ciklus, de már a helyén van az ujj. Ilyenkor csak a futó pont értéke lesz maximális.

9. Eredmények értékelése, továbbfejlesztési lehetőségek

A feladattal kapcsolatos legfontosabb célkitűzéseim a következők voltak: a program futása gyors legyen; a motorok gyorsítása és lassítása természetes legyen; képes kell, legyen a motorokat egyszerre és külön működtetni, akár különböző forgás iránnyal; csoportokra bontottan is képes legyen a motorokat működtetni, ha kell ellentétes orientációval; a motorok nem megegyező ideg való működtetése. Ezen célok közül a program a legtöbbet sikeresen teljesíti. A program futása az elvártaknak megfelelő. A gyorsítás és lassítás természetes hatása a fizikai kéz hiányában nem eldönthető. A motorok egyszerre és a külön működtetését a vártaknak megfelelően végzi. A csoportos mozgatásnál azonban még tapasztalhatók hibák. Az egyik csoport sikeres célba jutása után a „portomb”-ben nem megengedett értékek jelennek meg. Feltételezhetőleg a tomb() metódusban keresendő a hiba. Mivel ott történik a sorba rendezés és a válogatás. Feltételezéseim szerint a hibaokozásban segédkezik a globális tömb mivel fix nagyságú. Úgy vélem, hogy az egyik ciklusváltozó értéke nagyobb, mint az egyik tömb mérete és ez okozhatja a hibát.

A fentiekben ismertetett program működőképes, de természetesen számos továbbfejlesztése lehetséges, amelyeket a jövőben tervezek megvalósítani. Elsősorban a hatékonysági kérdések továbbfejlesztése szükséges a programkód áttekinthetőbb, jobban struktúrált megvalósításával. Komoly hátrány volt a fejlesztés során, hogy a kézfej fizikai megvalósítása híján csak a szoftveres rész tesztelésére nyílt lehetőség. A protézis fizikai megvalósítása után biztosan további fejlesztés válik szükségessé. Korlátokba ütköztem a rendelkezésre álló eszközök tekintetében is, mivel nem állt rendelkezésemre annyi motor, potenciométer, illetve fogaskerék, amivel szimulálni tudnám a valós működést.

Az értékelés írása pillanatában a program memória gondokkal küszködik számomra még ismeretlen okból kifolyólag az egyik pointer nem a megfelelő memória címre mutat, így a program megáll.

A fentiekben említett problémák megoldása után a fő továbbfejlesztési irány egy olyan kézprotézis fejlesztése, amely az embertől érkező jelek észlelése, feldolgozása alapján vezérelhető. Ennek megvalósításához egy EMG jel feldolgozására képes metódust vagy osztályt kell létrehozni. Továbbá egy fizikai ház megléte után lehetne egy olyan kódrészletet fejleszteni, ahol egy inverz Jacobi-mátrix segítségével sebességet tudna számolni és ennek megfelelően végezné a lassítást.

10. Felhasznált irodalom

- [1]Zalotay Péter: DigitálisteknikaiI BMF Kandó Kálmán Villamosmérnöki Kar Elektronikus jegyzet, 2011, p.27-42
- [2]Dr. Bencsik Attila - Felker Péter: Digitálisteknikai ÓE BGK, 2010, p.171-178
- [3]Sándor Tamás : Programozás II ÓE-KVK, 2018, p.15-25
- [4]Dr.Bencsik Attila - Dr.Harkay Gábor: Irányítástechnika ÓE BGK, 2010, p.11-12
- [5] Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V Datasheet, https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf [2022.03.02.]
- [6] Bill Giovino Take Advantage of a 16-bit Microcontroller's Performance and Low Power2019-07-24 <https://www.digikey.hu/en/articles/take-advantage-of-16-bit-mcu-performance-and-low-power> [2022.03.10.]
- [7] RoboworldHobbielektronika DC motor vezérlés Arduinoval - L298N-PWM- H-híd 2017.12.02. <https://hobbielektronikabolt.blogspot.com/2017/12/DC-motor-L298N-PWM.html>[2022.03.12.]
- [8] Regiszter (számítástechnika), [https://hu.wikipedia.org/wiki/Regiszter_\(számítástechnika\)](https://hu.wikipedia.org/wiki/Regiszter_(számítástechnika)), [2022.03.06.]
- [9]Prosthesis, <https://en.wikipedia.org/wiki/Prosthesis> [2022.04.07.]
- [10]Biomechatronics, <https://en.wikipedia.org/wiki/Biomechatronics> [2022.04.10.]
- [11] Paul D. Marasco, Jacqueline S., Hebert Jonathon, W. Sensinger, Dylan T. Beckler, Zachary C. Thumser, Ahmed W. Shehata, Heather E. Williams, Kathleen R. Wilson, Neurorobotic fusion of prosthetic touch, kinesthesia, and movement in bionic upper limbs promotes intrinsic brain behaviors, 2021.09.01. <https://consultqd.clevelandclinic.org/bionic-arms-neurorobotic-sensory-motor-fusion-restores-natural-behaviors/> [2022.04.03.]
- [12]Harvard-architektúra, <https://hu.wikipedia.org/wiki/Harvard-architektúra> [2022.03.01.]

- [13]millis(),<https://www.arduino.cc/reference/en/language/functions/time/millis/>
[2022.01.22.]
- [14]What is an „H-bridge”, <https://www.quora.com/What-is-an-“H-bridge”> [2022.04.04.]
- [15]Horváth Márk, Interfészek, előadás jegyzetek, Óbudai Egyetem, 2022
- [16]Nagy István, Mobilrobotok, előadás jegyzet, Óbudai Egyetem, 2022
- [17] Neumann-architektúra, <https://hu.wikipedia.org/wiki/Neumann-architektúra> [2022.02.28.]
- [18] Sarayut_Pongsanthia [S-curve-motio-profile](https://github.com/aodreds/S-curve-motio-profile) 2019.05.17. <https://github.com/aodreds/S-curve-motio-profile> [2021.12.21.]
- [19] Aswinth Raj What is PWM: Pulse Width Modulation
2018.09.18.[https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-](https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation)
modulation[2022.04.12.]
- [20]Finch, Jacqueline 2011 The ancient origins of prosthetic medicine
[https://www.thelancet.com/journals/lancet/article/PIIS0140-6736\(11\)60190-6/fulltext](https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(11)60190-6/fulltext)
[2022.03.23.]