Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Komáromi Sándor

ROS ALAPÚ PÓKSZERŰ ROBOT FEJLESZTÉSE

KONZULENS

Nagy Ákos

BUDAPEST, 2022

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Komáromi Sándor**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 05. 29

.....
Komáromi Sándor

Összefoglaló

A szakdolgozat tárgya egy – más szakdolgozatok készítése során kifejlesztett – pókszerű járó robot lényeges továbbfejlesztése mind mechanikai, mind vezérlő algoritmus és kód szempontjából.

A fejlesztendő robot mechanikája gyenge volt, több helyen eltört, és a mozgató szervók sem bírták az utólag beépített LIDAR szenzor által megnövekedett tömeg mozgatását. Megoldandó feladatnak tűztem ki egy olyan új járási algoritmus kifejlesztését, amivel a robot – orientációjától függetlenül – bármely irányba haladni képes, és az orientációját is tudja változtatni. Továbbá azt is, hogy az új járási algoritmus vezérelhető legyen ROS platformon keresztül. A négylábú robot járása bonyolultabb a hat- vagy nyolclábúaknál, mivel egyszerre csak egy lábát tudja felemelni, és közben a tömegközéppontjának a másik három láb által alkotott háromszögön belül kell maradnia.

A robot mechanikáját teljes mértékben újraterveztem, erősebb illesztéseket és fém betéteket alkalmaztam a csuklóknál, valamint erősebb szervókat használtam. A teljes mechanika 3D nyomtatással készült. A járási algoritmus megalkotása során azt részekre bontottam, és külön-külön dolgoztam ki. Az algoritmus először meghatározza a lépéstávolságot a mozgás irányába, majd a lábak lépéseinek sorrendjét, s végül az lábak új pozícióját is. Az algoritmus működését a fizikai robot használata előtt egy – erre a célra fejlesztett – szimulátorprogrammal ellenőriztem.

A robot összeszerelése problémamentes volt, a csuklók holtjátéka minimálissá vált, a réz betétek tökéletesen beváltak, az egész mechanika megfelelően működik. A sikeres – szimulátorban történt – próbák és a robot mechanikájának kalibrálása után a robot és az új járási algoritmus a célkitűzéseknek megfelelően működik. A ROS platformmal történő illesztést megvalósítottam, így irányíthatóvá is vált ezen keresztül.

Az új mechanika tervezésének és megépítésének tanulsága az, hogy az akkumulátort szerencsésebb lett volna a jobb tömegeloszlás miatt a robot geometriai középpontjában rögzíteni. Ez sokat segített volna a robot járási algoritmusának működésében, mely miután nem vizsgálta annak tömegközéppontjának pozíciója, emiatt kissé dülöngélve jár. A ROS illesztésnél a pozíció-orientációt alapú vezérlést alkalmaztam.

Abstract

The topic of my thesis is a significant development of a Four-Legged Spider Robot which pervious version has been developed by other students. My goal is also to improve mechanical parameters of robot and the algorithm of program code.

The mechanics of the robot to be developed were weak, it broked in several places. The servos could not managed the increased weight of the retrofitted LIDAR sensor. My goal is to develop a new moving algorithm that can move the robot in any direction, regardless of its orientation, and can change its orientation too. Furthermore, the new walking algorithm has to be controlled via the ROS platform. The walking of a four-legged robot is more complicated than that of a six- or eight-legged one, because it can only lift one leg at a time, while its center of gravity must remain within the triangle formed by the other three legs.

I completely redesigned the robot's mechanics, using stronger joints and metal inserts at the joints, and using stronger servos. All of mechanics elements are made with 3D printing. During the creation of the walking algorithm, I divided it into parts and worked it out step-by-step. The algorithm determines first the step distance in the direction of movement, then the sequence of steps of the legs, and finally the new position of the legs. Before using the physical robot, I checked the operation of the algorithm with a simulator program developed for this purpose.

The assembly of the robot was trouble-free, the backlash of the joints was kept to a minimum, the copper inserts proved to be perfect, the whole mechanics worked properly. After successful tests in a simulator and calibration of the robot's mechanics, the robot and the new walking algorithm work as intended. I implemented the integration with the ROS platform, so the robot became controllable via it.

This would have helped a lot in the operation of the robot's walking algorithm, which, after I didn't examine the position of its center of gravity, makes it a bit tumbly. For ROS fitting, I used position orientation-based control.

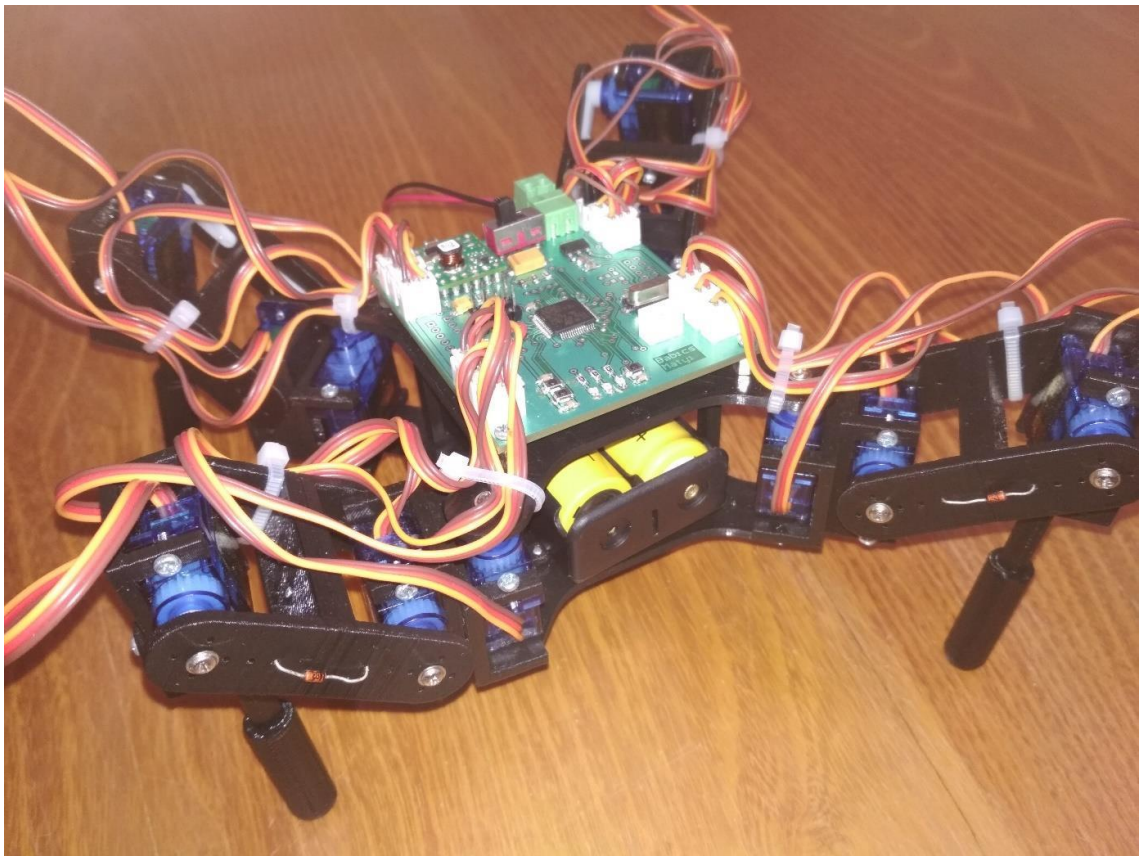
1 Bevezetés

Manapság egyre inkább előtérbe kerül a mobil robotok fejlesztése, felhasználása, napjainkban ezek elsődlegesen az iparban jelennek meg, de egyéb más alkalmazásterületük is feltörekvőben van. A mobil robotok egyik speciális területe a járó robotok, melyek felépítése bonyolultabb a hagyományos kerékmeghajtású robotoknál, viszont sokkal alkalmasabbak a talaj egyenetlenségeinek kompenzálására mozgásuk közben. Ezen robotok elsődleges feladata általában a környezetük feltérképezése, és az abban való problémamentes navigáció megvalósítása. Eme robotok mechanikai kialakításából következik az, hogy képesek többirányú mozgás megvalósítására, vagyis könnyedén tudnak előre, hátra, oldalra, fel- és lelépkedni.

A szakdolgozatom megvalósítása során két másik hallgató munkáját folytattam. Elsődleges feladatom az önálló laboratóriumom [1] során tervezett robot háromdimenziós (3D) nyomtatással készített alkatrészeinek legyártása és összeszerelése volt. Továbbá célkitűzésem egy olyan új járási algoritmus elkészítését, mely megvalósítja a fent említett mozgási sémát, miközben a robot orientációjának változtatásáért is felel. S végül, de nem utolsó sorban az új mechanikával rendelkező robot integrálása is elvárás volt a részemről a Robot Operating System (ROS) [2] interfész alá.

1.1 Pókszerű járó robot

A robot készítése Babits Mátyás munkájával kezdődött. [3] Ő egy négylábú robotot alkotott meg, mely jellegét tekintve pókszerű volt. A robot négy lába egyenként három-három csuklóból állt, olyan elrendezésben, mint a pókok, rákok, vagy a hangyák lábai. A robot alkatrészei 3D nyomtatással készültek. Lábanként három hajlási pont biztosítja az elegendő szabadsági fokot ahhoz, hogy egy-egy láb minden olyan pozíciót fel tudjon venni, amelyet a járási algoritmus megkövetel. A Mátyás által készített robot rendelkezett egy olyan beágyazott szoftverrel (firmware), mely be tudta mutatni a robot járási képességeit. Kezelte a szervomotorokat, valamint a lábakra kiadott végpontokból kiszámolta az inverz geometriai algoritmus segítségével a motorok szögét. A robot mozgása limitált volt; tudott előre menni, valamint jobbra és balra forogni.



1. ábra: Babits Mátyás által készített pókszerű robot.

Ezt követően Massár Lóránt Mátyás fejlesztette tovább a robotot [4], akinek elsődleges célja a robot okosítása volt, ezért olyan szenzorokkal szerelte fel azt, melyek térképezési és kommunikációs feladatokat láttak el. Az okosítás jegyében Lóránt ellátta a robotot egy Light Detection and Ranging (LIDAR) egységgel, illetve egy Internal Measurement Unit (IMU) egységgel is. Előbbi egy olyan lézer alapú távolságérzékelő, mely 360°-ban meghatározza a robottól a külvilág távolságát [5], míg utóbbi a robot térbeli mozgásáról és elhelyezkedéséről biztosít pontos információkat [6]. A robot egy ESP-01 wifi modullal is gazdagodott, mely a távoli vezérlés megvalósítását biztosító kommunikációért felelt, s egyben lehetővé tette a robot irányítását a ROS platformon keresztül. A ROSSerial node segítségével kiépítette a robottal való kommunikációt [7], és a RVIZ [8]valamint a Hector SLAM [9]segítségével környezet feltérképező algoritmust hozott létre.

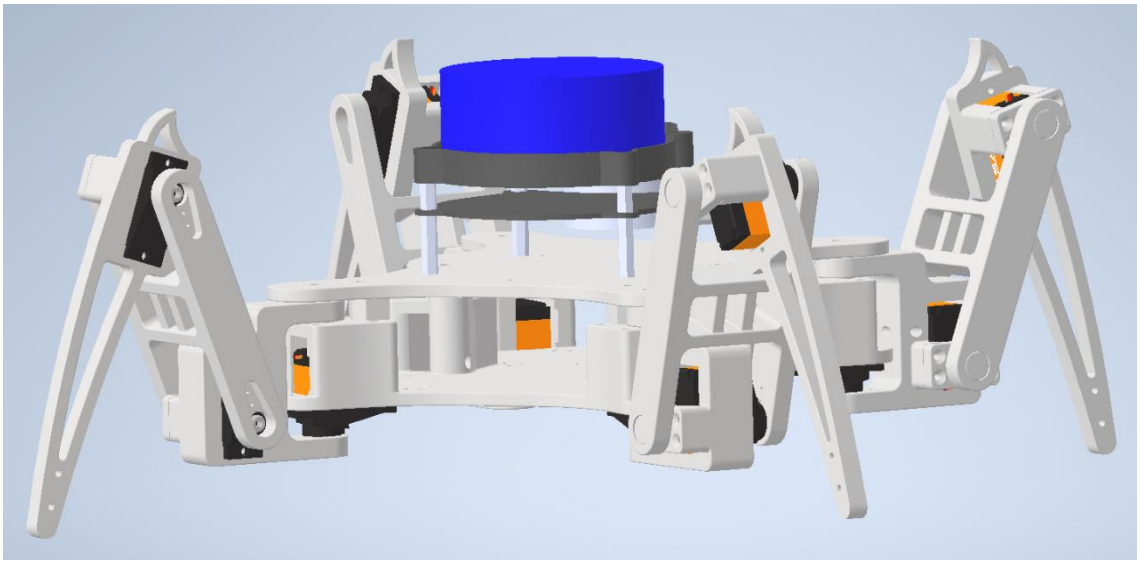
2. ábra: Massár Lóránt Mátyás által készített pókszerű robot.

1.2 Új mechanikai modell

Az önálló laboratóriumom kezdetekor, mikor kézhez kaptam a Lóránt által készített robotot, elsődlegesen a robot mechanikai hiányosságai voltak szembetűnőek. A robot több helyen el volt törve, és a szervomotorjai is nehezen bírták a LIDAR által megnövelt terhet. Ezért mechanikailag teljesen újraterveztem a robotot, erősebb illesztéseket használtam a csuklóknál, valamint a szerelhetőség megtartása érdekében réz betéteket (inserteket) használtam a csavarok rögzítésére. A szervokarok rögzítését megerősítettem, új, erősebb szervomotorokat választottam a robothoz, valamint további két új lábbal is kiegészíthetővé tettem a robot modelljét. Az újratervezés során fontos célkitűzésem volt az, hogy a robot lábainak kinematikai kialakítását ne változtassam meg, azaz a robot lábainak mozgási algoritmusa ugyanolyan maradjon, mint az eredetileg volt, csak paraméterei módosuljanak.

Az önálló laboratóriumom során a robot összeszerelésére már nem maradt időm, így csak annak megtervezése történt meg, melyet a 3. ábra mutat be. Ezen kívül

kiválasztottam a robot új meghajtó motorját, valamint az inverz kinematikai modell paramétereit is meghatároztam, melyeket a 2.1 pontban részletezek.



3. ábra: Önálló laborom alatt tervezett robot 3D-s terve. [1]

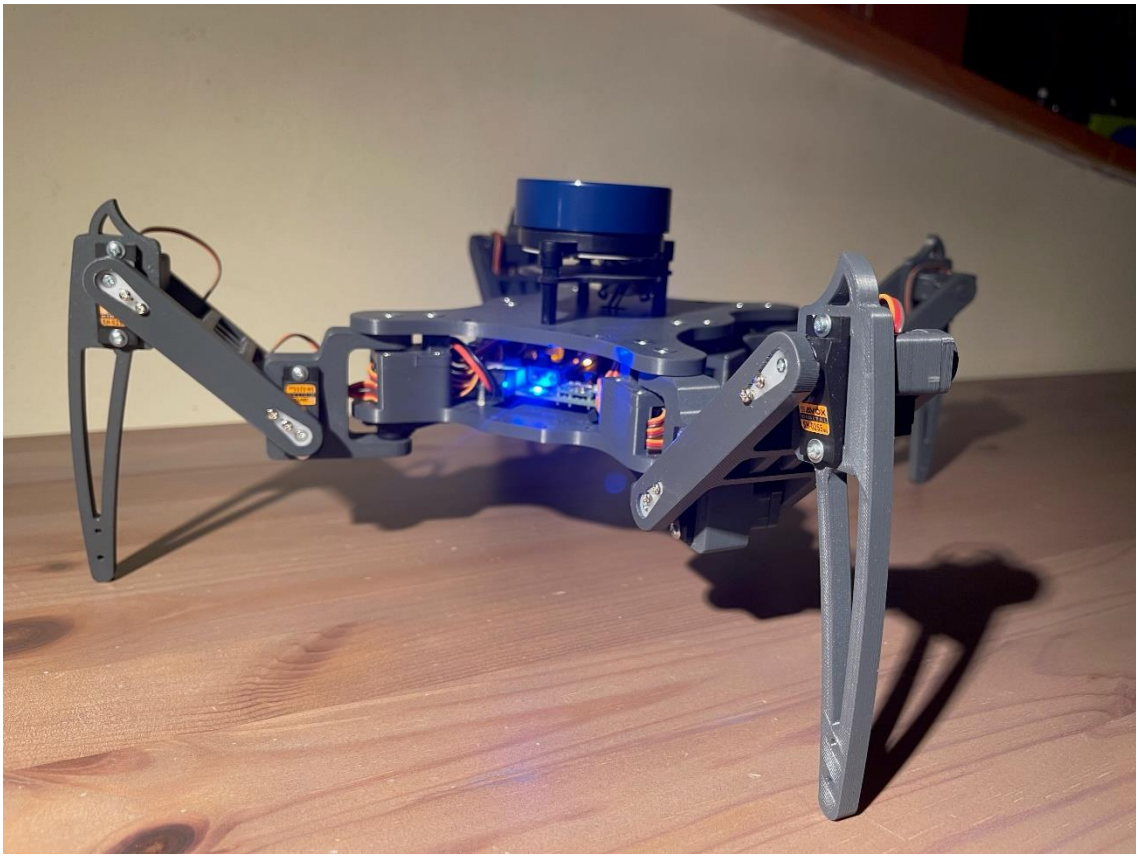
1.3 Célkitűzések

Célom az volt, hogy a robotot elődjeim munkájának megtartásával tudjam továbbfejleszteni, kiegészíteni. Továbbá elkészítsem a robot új mechanikai vázát, valamint új járási algoritmust tervezzek, mely alkalmas többirányú mozgás megvalósítására, mely nagyban kiterjesztené a robot alkalmazási lehetőségeit. Szeretném Lóránt munkáját - melyet az 1.1 pontban összefoglaltam - újrahasznosítani az általam tervezett roboton. Annak érdekében, hogy a fent említett céljaimat meg tudjam valósítani, megfogalmaztam az alábbi követelményeket:

- Készüljön el a robot mechanikai megvalósítása a kritikus alkatrészek letesztelésével együtt.
- A robot alkalmas legyen bármilyen irányú mozgás megvalósítására, függetlenül annak orientációjától, ezzel új járási algoritmust alkotva.
- A robot mozgás közben képes legyen orientációjának megváltoztatására.
- A robot új járási algoritmusának legyen ROS interfésze.
- A robot vezérelhető legyen ROS platformon keresztül.

A fentebb megfogalmazott céljaim eléréséhez először tesztelnem kell a robot tervének szerkezeti paramétereit és összeszerelhetőségét. Be kell kalibrálnom a robot

motorjain lévő szervokarokat, hogy azok megfelelő tartományban mozogjanak. Az általam tervezett járási algoritmus működését meg kell vizsgálnom, egy a Babits Mátyás által készített szimulációban, mielőtt azt a tényleges roboton tesztelném. Továbbá ki kell egészítenem a robot beágyazott kódját úgy, hogy az alkalmas legyen az új motorok használatára és a megváltozott mechanikai paraméterek feldolgozására. Végül újra kell integrálnom a robot rendszerét a ROS környezetbe.



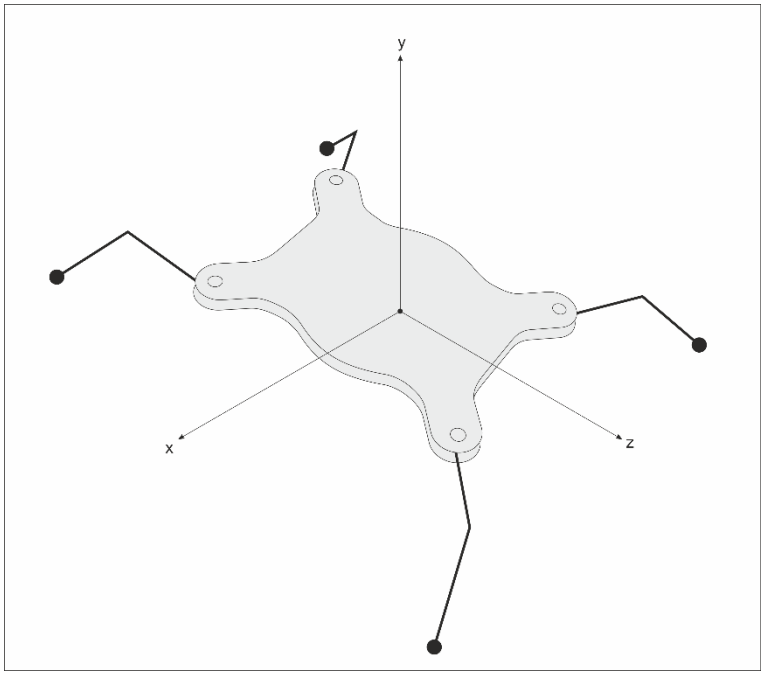
4. ábra: A kész pókszerű robot.

2 Mechanikai megvalósítás

Szakdolgozatom megkezdésekor először a robot legyártásával, illetve összeszerelésével foglalkoztam. A végleges gyártás előtt tesztelemeken próbáltam ki az összeszerelhetőségét, kijavítottam a felmerülő hibákat a tervben. A 2.1 pontban részletezem a robot mechanikai paramétereit, melyeknek nagy része megváltozott Mátyás tervei óta. Az alkatrészek gyártása során felmerült problémákat, változtatásokat a 2.2 fejezetben fejtem ki bővebben.

2.1 Mechanikai paraméterek

Az Önálló laboratóriumi beszámolómban [1] már részleteztem a robot új mechanikai paramétereit, ám ebben a szakaszban két okból is érdemesnek tartottam ezeket felfrissíteni. Egyrészt a robot korrekt működése érdekében szükségét láttam a finomhangolásnak, amit a szimuláció során szabad szemmel is észlelt pontatlanságok orvoslása miatt kellett elvégeznem (a paraméterek kerekítéséből adódó problémákat, és azok megoldását a 4.1.2 pontban fejtem ki részleteiben). Másrészt az alábbiakban definiált paramétereket, s azok származtatását a dolgozatom során későbbiekben vonatkozási pontként is használom.



5. ábra: A robot koordinátarendszere, *z* tengely előre, *x* tengely oldalra és *y* tengely felfelé.

Láb	dx (cm)	dy (cm)	dz (cm)	d_1 (cm)	d_2 (cm)	d_3 (cm)	d_3x (cm)
Jobb első	5.2462	1.285	7.4962	3.22	7.5	10.7593	-0.45
Jobb hátsó	-5.2462	1.285	7.4962	3.22	7.5	10.7593	0.45
Bal első	5.2462	1.285	-7.4962	3.22	7.5	10.7593	0.45
Bal hátsó	-5.2462	1.285	-7.4962	3.22	7.5	10.7593	-0.45

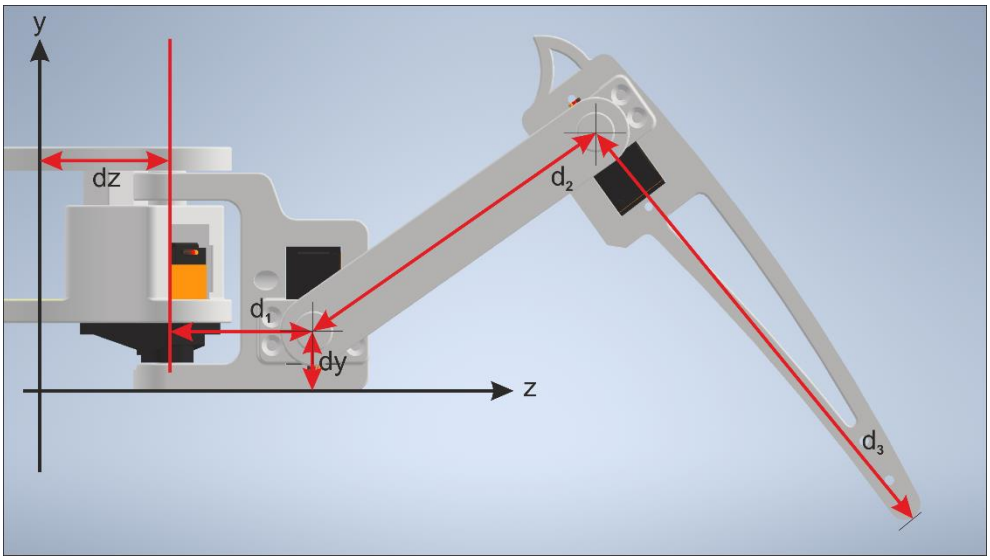
1. táblázat: A robot lábaihoz tartozó mechanikai paraméterek, centiméterben.

A geometriai modell az 1. táblázatban szereplő paraméterek alapján eltolásokból és forgási pontokból épül fel, ezt a továbbiakban a 6. ábra valamint az 7. ábra alapján részletezem. A dx, dy, dz paraméterek a teljes lábat transzformálják annak első forgási pontjába. Innen a láb további részeit az első motor szöge forgatja, majd a d_1 tovább tolja. A második csukó forgatása után a d_2 paraméter tolja tovább a harmadik csuklóba a láb végpontját, majd a d_3 és d_3x által eltolt pont adja meg a robot láb végpontjának tényleges pozícióját. Ezzel meghatározható a láb előrefele számolt geometriai egyenlete, melyet Babits Mátyás jegyzetéből [10] emeltem át a (2.1) egyenlettel együtt, ami pedig mátrix szorzásokkal hűen leírja a fent említett eltolásokat és forgatásokat. Az egyenletben lévő C_i és S_i paraméterek koszinusz és szinusz műveleteknek felelnek meg.

$$\begin{pmatrix}1&0&0&dx\\0&1&0&dy\\0&0&1&dz\\0&0&0&1\end{pmatrix}\begin{pmatrix}C_1&0&S_1&0\\0&1&0&0\\-S_1&0&C_1&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&d_1\\0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\0&C_2&-S_2&0\\0&S_2&C_2&0\\0&0&0&1\end{pmatrix}$$
$$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&d_2\\0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\0&C_3&-S_3&0\\0&S_3&C_3&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&d_3x\\0&1&0&0\\0&0&1&d_3\\0&0&0&1\end{pmatrix}$$

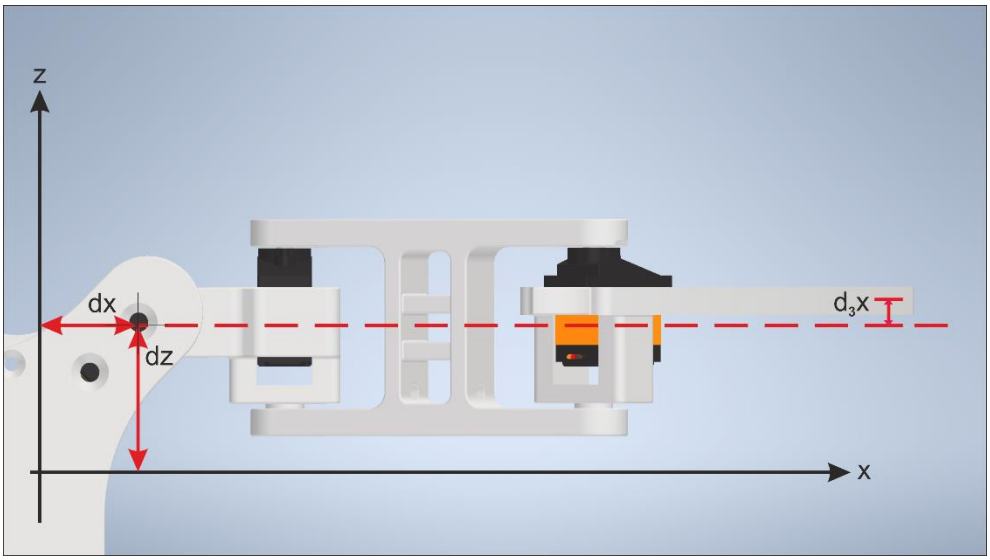
(2.1)

Az 1. táblázat lábanként tartalmazza azokat a paramétereket, melyeket az inverz kinematikai modell felhasznál a motorok új pozíciójának állításához [10]. A robot koordinátarendszere a szokványostól eltérő módon épül fel, mivel az a robot szemszögéből határozza meg az irányokat, ezeket az 5. ábra mutatom be. A z koordinátatengely mutat előre, az x tengely pedig a z koordinátatengely irányából nézve mutat jobbra. Az előbb említett két tengely metszéspontja határozza meg a robot középpontját felülnézetből nézve.



6. ábra: A robot lábának paraméterei oldalnézetből.

Az y tengely ebből az origóból mutat fölfelé, nullapontja pedig a robottest legalsó pontján van. Ezen analógiát követve a dz és dx paraméterek minden lábra meghatározzák az első motor tengelyét a robot testének nullapontjához képest, ezzel eltolva a teljes lábat abba a forgáspontba. A dy paraméter eltolja a láb kezdőpontját a második motor tengelyének magasságába, ám ez természetesen az első motorra nincs visszahatással, hiszen annak tengelye mentén történik az eltolás, viszont a második motor e tengely menti eltolását így már be lehet állítani. A d_1 eltolja a második motor tengelyét sugárirányba, ezzel meghatározza az első és a második csukló tengelyei közötti távolságot. A d_2 hasonló módon a harmadik motor tengelyét helyezi odébb, ezzel meghatározva a második és harmadik motor tengelyei közötti távolságot.



7. ábra: A robot lábának paraméterei felülnézetből.

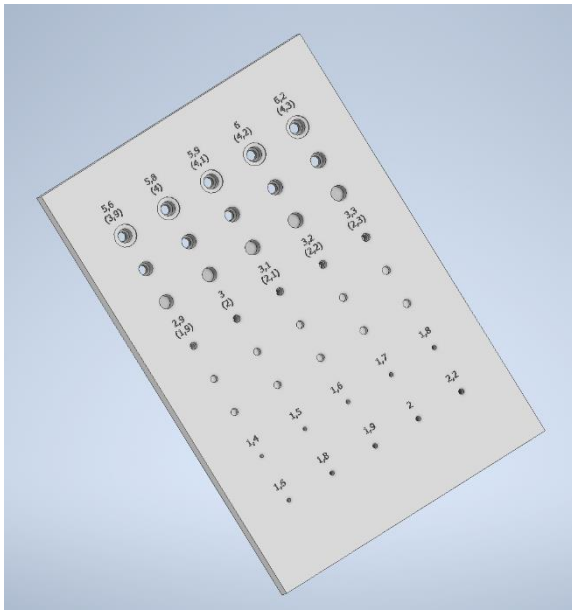
A d_3 paraméter a láb végpontja és a harmadik csukló közötti távolságot határozza meg. Végül a d_3x a második és harmadik csukló tengelye mentén tolja el a láb végpontját az első motor tengelyéhez képest. Erre azért van szükség, mert a láb utolsó alkatrésze kissé eltolva érinti a földet az első forgástengelyhez képest.

2.2 Nyomtatás

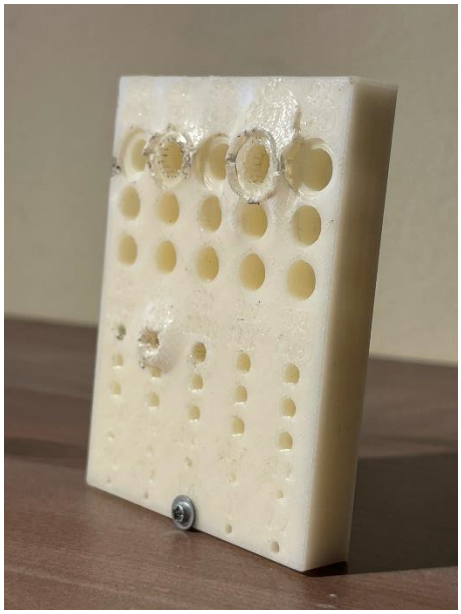
A robot gyártásának megkezdése előtt fontosnak tartottam pár tesztnyomtatás elvégzését, ugyanis ezen prototípust megelőzően még nem használtam műanyagba ágyazható réz menetes betétet, így az külön tesztelést igényelt, melyről a 2.2.1 pontban számolok be részletesebben. Továbbá leteszteltem egy teljes láb összeszerelését is, melyet a 2.2.2 pontban ismertetek.

2.2.1 Csavarok, betétek tesztelése

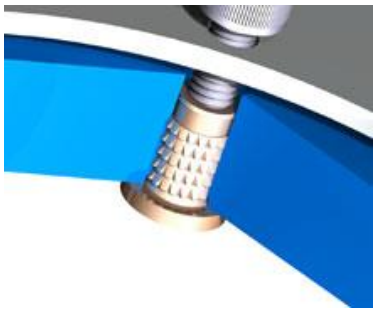
A menetes betétek, valamint minden egyéb csavar tesztelése érdekében létrehoztam egy egyszerű tesztelési panelt, melyen tizedmilliméter átmérő különbségű furatok vannak. Így a végleges tervre olyan átmérőjű furatok kerülnek, melyekbe a legjobban illeszkednek a betétek és egyéb csavarok. A 8. ábra segítségével mutatom be a tervezett panelt, melyen jól láthatóak a különböző méretű furatok. Minden átmérőjű furatból többet terveztem, hogy eltérő rögzítési lehetőségeket is ki tudjak próbálni.



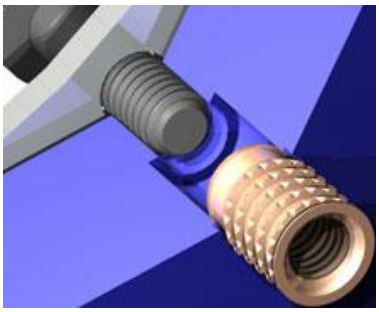
8. ábra: Teszt panel modellje [1]



9. ábra: Panel a tesztelés után.



10. ábra: Peremmel rendelkező insert [11]



11. ábra: Perem nélküli insert [11]

A betétek, melyeket a robotnál használok, fogakkal rendelkeznek, és ezeknél is többféle behelyezési módszerre találtam példát. Az egyik esetben egy satu segítségével lehet belepréselni a műanyagba a rézbetétet, és a csavart ebbe fentről rögzíteni, ekkor terhelés hatására kizárólag az előbb említett fogak tartják a betétet. Ennél biztosabb megoldás a peremes betét használata, melybe a peremmel ellentétes oldalról érdemes belecsavarni a csavart. Ezen felhasználáskor a tengelyirányú terhelést a perem, a csavaró erőhatást pedig jellemzően a fogak veszik fel. Erre a technológiára a 10. ábra mutat példát. Azon a helyeken, ahol megoldható volt, én ezt alkalmaztam a rögzítésre. Egyéb esetekben a 11. ábra látható insertet használtam fel, ám én gyártástechnológiai okokból a csavart a behelyezés irányából csavartam bele. További lehetőség a betét behelyezése közben melegítés használata, melyre egy forrasztópáka alkalmas, ekkor a réz jó hővezetőképességéből adódóan a betét körüli műanyag megolvad, és a nyomtatáskor létrejött rétegrendek elmosódnak, így jobban körülölelik a betét fogait. Ezzel egy sokkal erősebb és biztosabb rögzítés készíthető, mely a tesztek során is beigazolódott. A tesztek eredményét a 9. ábra segítségével mutatom be. Azon esetekben, amikor a panel belülről tört ki és felpúposodott, ott melegítést is alkalmaztam. Itt igen látványos volt az, hogy az hordozó anyag egyben szakadt ki a betéttel együtt, és a belső réteghatárokon tört szét.

Ezt az erősebb rögzítési lehetőséget végül nem alkalmaztam a végleges megoldásban, mert a melegítés hatására túlzottan megolvadt műanyag, s így a fellépő oldalirányú erőkkel szemben nem tartotta a betétet, ezért a folyamat közben eldőlt az alkatrész. Emiatt nem tudtam minden esetben biztosítani a pontos behelyezést, ráadásul a melegítés nélküli változat is megfelelő tartóerővel rendelkezett.

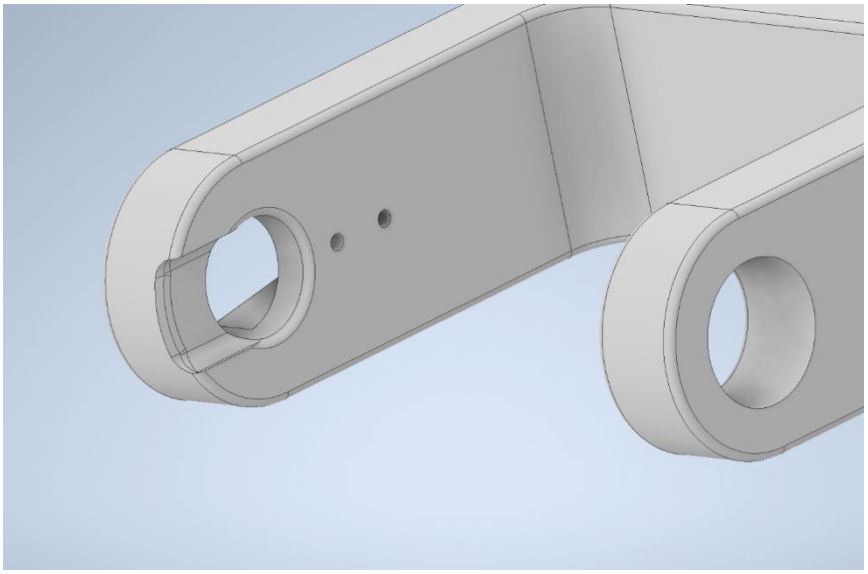
2.2.2 Alkatrész újra tervezés

A robot alkatrészeinek végleges elkészítése előtt kinyomtatásra került egy lábnyi alkatrész tesztelés céljából. A teszt alkatrészek összeszerelése során egy fő probléma

lépett fel, melyre előzetesen számítottam. A lábak csuklóinak összeszerelésekor a 12. ábra által bemutatott elem két szárát túlzottan szét kellet feszíteni. A végleges tervnél a képen is látható bevágásokat alkalmaztam az összeszerelés megkönnyítése érdekében. A bevágás mellett megnagyobbítottam a távolságot a két szár között, ezzel is elősegítve az összeszerelést, ez így nem visz tengelyirányú mozgást a csuklóba, mert a szervokarok felhelyezésekor csupán az egyik oldalra húzza magát a csuklót.

2.3 A fizikai robot megvalósításának értékelése

A robot összeszerelése kifejezetten problémamentes volt. Az előzetes tesztelések meghozták eredményüket, minden alkatrész pontosan összeillett egymással, együtt egy erős vázat alkotva. A csuklókban minimális holtjátékot tapasztaltam, ezen talán már csak a csapágy használata segített volna. A réz betétek egytől-egyig tökéletesen működtek, a robot könnyedén szétszerelhetővé vált tőlük, melyet alkalmaznom is kellett, mikor az egyik motor tönkrement. Azonban, ha újra tervezném a robot vázát, valószínűleg kissé máshogy kezdenék neki. A robot jelenleg fölöslegesen tartalmazza az ötödik, hatodik lábhoz tartozó rögzítési pontokat, ez növelte a robot tömegét és nagyította a fő elem méretét. Jelenleg elég távlati célkitűzésnek tűnik a további két láb használata, mert teljesen új mikrokontrollert is kellene használni hozzá. Az akkumulátor elhelyezésén is változtatnék, jelen helyzetében túlzottan hátulra helyeződik a tömegközéppontja a robotnak miatta.



12. ábra: A könnyebb összeszerelhetőség kedvéért bevágás a robot csuklóin.

3 Omnidirekcionális járási algoritmus

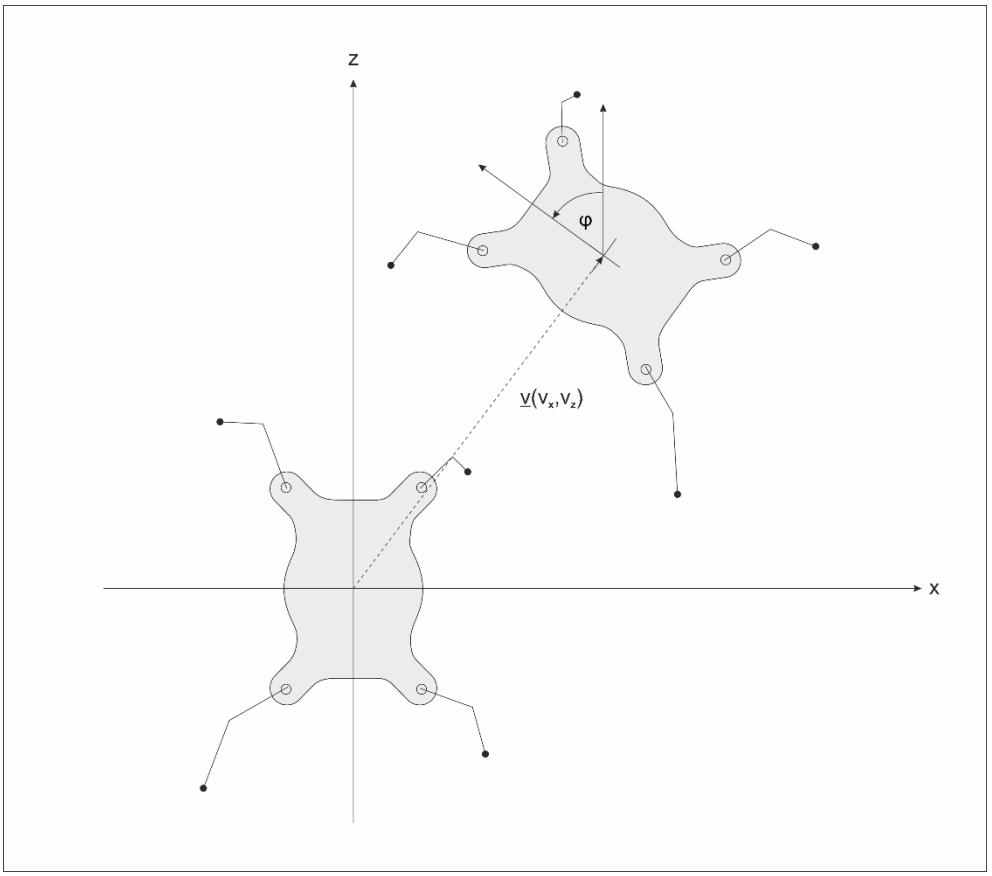
Egy négylábú robot mozgása elsőre egyszerűbbnek tűnhet, mint egy hat, vagy nyolclábúé, hiszen kevesebb láb irányítását és lépési pozícióját kell megvalósítani. Gondoljunk csak a hangyák és pókok mozgására, melyek igen kaotikusnak tűnnek, nehéz megállapítani azt, hogy hogyan mozognak. Azonban ezt kicsit jobban megvizsgálva, elsődleges becslésként kimondhatjuk azt, hogy egyensúlyozás nélküli, statikus járás esetén azon robotok, melyeknek több mint négy lába van, stabilabb és gyorsabb mozgást tudnak megvalósítani. Ez abból adódik, hogy a robot stabilitásának megtartása érdekében legalább három lábának mindenféleképpen a földet kell érintenie, és a robot tömegközéppontjának is a lábak által alkotott háromszögön belül kell maradnia. Amíg egy négylábú robot a fentiek alapján egyszerre csak egy lábát tudja felemelni, addig egy hat vagy nyolc lábbal rendelkező már három, illetve négy lábat - a szimmetria megtartása érdekében - emel egyszerre a levegőbe, ezzel egy sokkal gyorsabb és stabilabb mozgást érve el. Ezen robotok, a négy lábbal rendelkezőkkel szemben, egyszerre lábaik számának felét fel tudják emelni, ezért két lépés után ismétlődik a mozgási algoritmusuk. A fentiek alapján sejthetjük, hogy egy négylábú statikus mozgást megvalósító algoritmus sokkal lassabb, szaggatottabb, talán egy fokkal bonyolultabb eredményhez vezet, mint egy több lábbal rendelkező esetben.

A járási algoritmus újratervezésekor elsődleges szempontom az volt, hogy a régi programmal ellentétben a robot ne csak egyenesen tudjon menni és a tengelye körül tudjon forogni [12], hanem bármelyik irányba el tudjon indulni, függetlenül attól, hogy milyen a robot orientációja. A robot modelljének újratervezésekor, e célt figyelembe véve, úgy alakítottam ki a tervet, hogy az a tengelyekre szimmetrikus legyen, így a robot előre és oldalra a lépéssorrenden kívül ugyanúgy tudjon lépni. Azt változtatva, hogy mennyit lépjen előre és mennyit oldalra, a robot tetszőleges irányba mozgatható úgy, hogy végig egy irányba néz, tehát ily módon mechanikailag minden irányba mozgathatóvá válik.

A fent említett általános mozgási algoritmus tervezésekor figyelembe vettem azt, hogy a robot ROS környezetbe való integrálása is a céljaim között szerepel, melyre a 6 fejezetben térek ki részletesebben. A mozgási parancs két paramétert vár, a cél pozíció koordinátait a robot koordináta tengelyében, azaz a v elmozdulás vektort, illetve a robot

φ relatív elfordulásának szögét. Az algoritmus által várt paramétereket az 13. ábra segítségével mutatom be. Algoritmusom több különálló részre bontható, először kiszámolja a legnagyobb láblépés távolságot, melyet a lábak elérési tartománya határol. Majd kiszámolja a lelépendő távolság és az elfordulandó szög alapján azt a lépéstávolságot, illetve elfordulási szöget lépésenként, melyet a robot ténylegesen meg fog tenni. A tényleges lépéstávolság nem minden esetben egyenlő a maximummal, hiszen a legtöbb esetben a lelépendő távolság sem egész számú többszöröse a maximum lépéstávolságnak. A továbbiakban az algoritmus a mozgás irányának függvényében összeállítja a lábak lépésének sorrendjét, majd végül kiszámolja minden egyes lábhoz a hozzá tartozó új pozíciót.

A mozgás tervezése során elsődlegesen megvizsgáltam a különböző omnidirekcionális mozgást leíró publikációkat, melyek ilyen fajta robotot használnak [13] [14]. Megvizsgáltam különböző dokumentumokat, melyek a robot tengely körüli forgását tanulmányozták [15]. A robot forgását végül egy leegyszerűsített, Babits Mátyás algoritmusához hasonló [12], megoldással készítettem el, a rendelkezésemre álló időkeret lejárta miatt. Munkám során megvizsgáltam néhány kevésbé általános publikációt is, például amely az elromlott lábbal [16], csuklóval való közlekedésről szól, illetve amely kifejezetten egyenetlen talajon való mozgást elemez [17].



13. ábra: A mozgási algoritmus bemeneti paraméterei, $\underline{v}$ vektor és φ szög

3.1 Láb elérési tartománya

A mozgási algoritmus elkészítése előtt meg kellett vizsgálnom a robot lábainak elérési tartományát, melyet a mechanikai kialakításuk határol. A robot egy lábának elérési tartományának nevezzük azt a területet, melyen belül a láb mechanikailag képes kinyúlni. A legtöbb ezzel a témakörrel foglalkozó cikkben egy téglalappal egyszerűsítik le a lábak elérési tartományát az egyszerűbb számolás érdekében [14]. Az egyszerűsítést én nem alkalmaztam, mert a téglalap alakú elérési tartomány esetében a robot előre nagyobb távolságot tud lépni, mint oldalra. Számomra elsődleges cél volt a robot szimmetriájának megtartása, így a felhasználtam a teljes mechanikailag megengedett tartományt, melyet a 3.1 fejezetben részletezek.

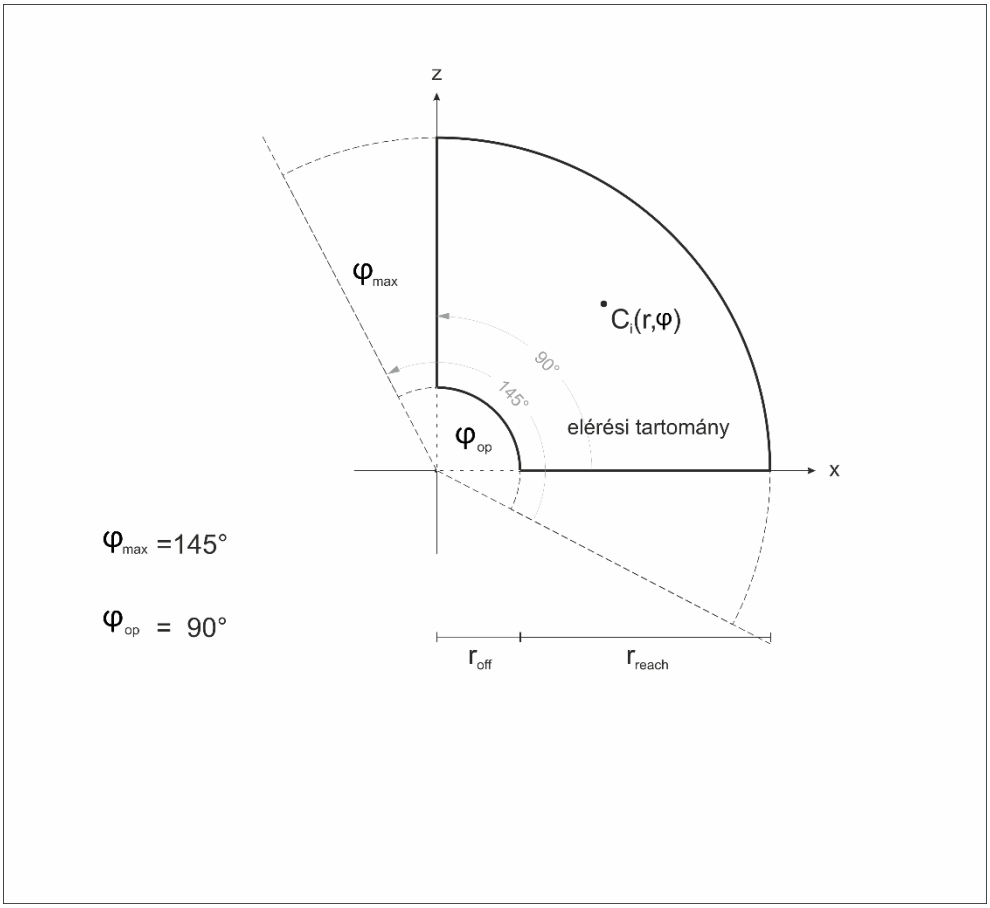
3.1.1 Az elérési tartomány

A továbbiakban bemutatom egy láb elérési tartományának felépítését és paramétereit, elegendő egy tartományt részleteznem, mert a többi ennek tükrözésivel kiszámolható. A robot koordináta rendszere úgy épül fel, hogy a robot testének középpontja az origó és a négy láb a négy síknegyeden helyezkedik el. A robot eleje a *z*

tengely felé, míg a jobb oldala az *x* tengely felé néz, így a jobb mellső lábának koordinátái pozitívok. Az *y* tengely mentén a robot hasmagassága állítható.

Egy láb mechanikai paramétereit és összeállítását a 2.1 fejezet részletezi, az elérési tartomány szempontjából az a fontos, hogy a három forgó pont közül az első az *y* tengely körül forog, míg a második és a harmadik az *x* tengely körül. A robot lábának inverzkinematikai modellje *x, y, z* koordinátákat használ a pozíció megadására. Viszont a robot lábának kialakítása miatt a láb egy pozíciója sokkal jobban leírható egy φ szög és egy *r* sugár segítségével, ahol előbbi az első forgó pont szögének felel meg, utóbbi a második és harmadik forgó ponttal egy egyenes mentén növelhető a távolság.

A fent leírtak alapján következik, hogy az elérhető tartomány egy kör valamilyen szelete, melynél a láb mechanikai paraméterei korlátozzák azt, hogy mennyire messzire tud nyúlni, ezzel egy $r_{off} + r_{reach}$ maximum nyúlási távolság számolható. A körszelet nagyságát tovább korlátozza az első forgó pontba épített szervo forgási korlátja, melynek teljes tartománya ugyan $\varphi_{max} = 145°$, viszont a tipikus használati tartománya csak $\varphi_{op} = 90°$ fok így én az utóbbi értéket használtam fel [18].

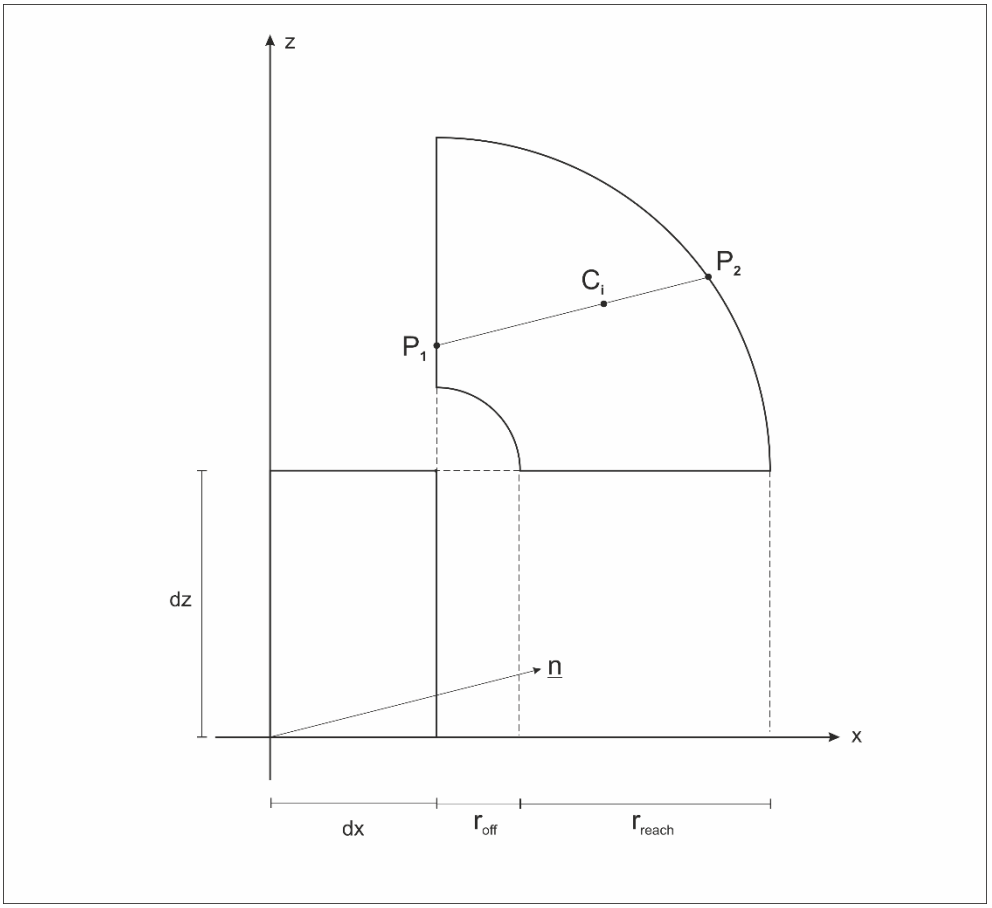


14. ábra: Egy láb elérési tartománya, annak paraméterei, valamint C_i középpontja.

Egy láb elérési tartományát tovább szűkíti az, hogy a végpont nem képes teljesen a *y* tengely körüli forgó pont alá hajolni, hanem csak egy adott paraméter értékig képes behúzni azt. Ebből következik, hogy az elérési tartományt határolja egy kisebb r_{off} sugarú körív is, melyen belülre nem tudja húzni a lábát a robot. Összefoglalva a robot lábának elérési tartománya egy fánkszeletre hasonlít, melyet az 14. ábra mutatok be. Korlátait elsősorban mechanikai korlátok adják.

3.1.2 Láb pályája

A robot lábának mozgási pályáját egy egyenes határolja az z-x síkon. A pálya járás közben minden esetben érinti a 3.1.3 pontban bemutatott C_i középpontot. A pálya irányultságát elsősorban a robot mozgási iránya szabja meg, melyet az 15. ábra $\underline{n}$ normálvektorral mutatok be. A robot mozgási pályája minden esetben két ponton metszi az eléréi tartományt, ezen két pont segítségével könnyedén kiszámolható a láb lépésének hossza, melyet a 3.2 fejezet tárgyal részletesebben. A láb pályáját az 15. ábra mutatom be, melyen a P_1 és a P_2 pontok által határolt egyenes a láb pályája.



15. ábra: Egy láb pályája a P_1 és P_2 pontok között, érintve a C_i középpontot

A robot lábának y tengely menti mozgását, Babits Mátyás munkája alapján [12] az (3.1) képlet adja meg. A képletben szereplő *dy* paraméter a robot hasmagasságát, a *h* változó pedig a láb végpontjának elemelkedését határozzák meg. A *t* paraméter az időt reprezentálja [0, 1] intervallumon, a mozgás kezdetétől annak végéig.

$$y = -dy + (1 - (2t - 1)^2) * h$$

(3.1)

3.1.3 Elérési tartomány középpontja

A mozgási algoritmus tervezését nagyban megkönnyíti az, ha a láb pályája, függetlenül annak szögétől, minden esetben átmegy egy fix ponton, ezzel meghatározva egy láb elérési tartományának C_i középpontját [13]. A 14. ábra látható C_i pontot a 3.1.3 bekezdésben részletezett elérési tartományba illesztettem be, így annak koordinátáit sugár-szög párral adtam meg. A szimmetria megtartásának érdekében a C_i szöge a teljes lefedettség felével egyezik meg (3.2), míg sugara az elérési tartomány két körívének átlaga (3.3).

$$r_{Ci} = \frac{r_{off} + (r_{off} + r_{reach})}{2}$$

(3.2)

$$\varphi_{Ci} = \frac{\varphi_{op}}{2}$$

(3.3)

A mozgás irányok közötti váltást tovább egyszerűsíti a C_i pont, ugyanis így mozgásiránytól függetlenül létezik olyan pont, mely mindig részese a mozgás pályájának. A járási algoritmus befejezésekor az egyik láb mindig ezen pontot fogja érinteni.

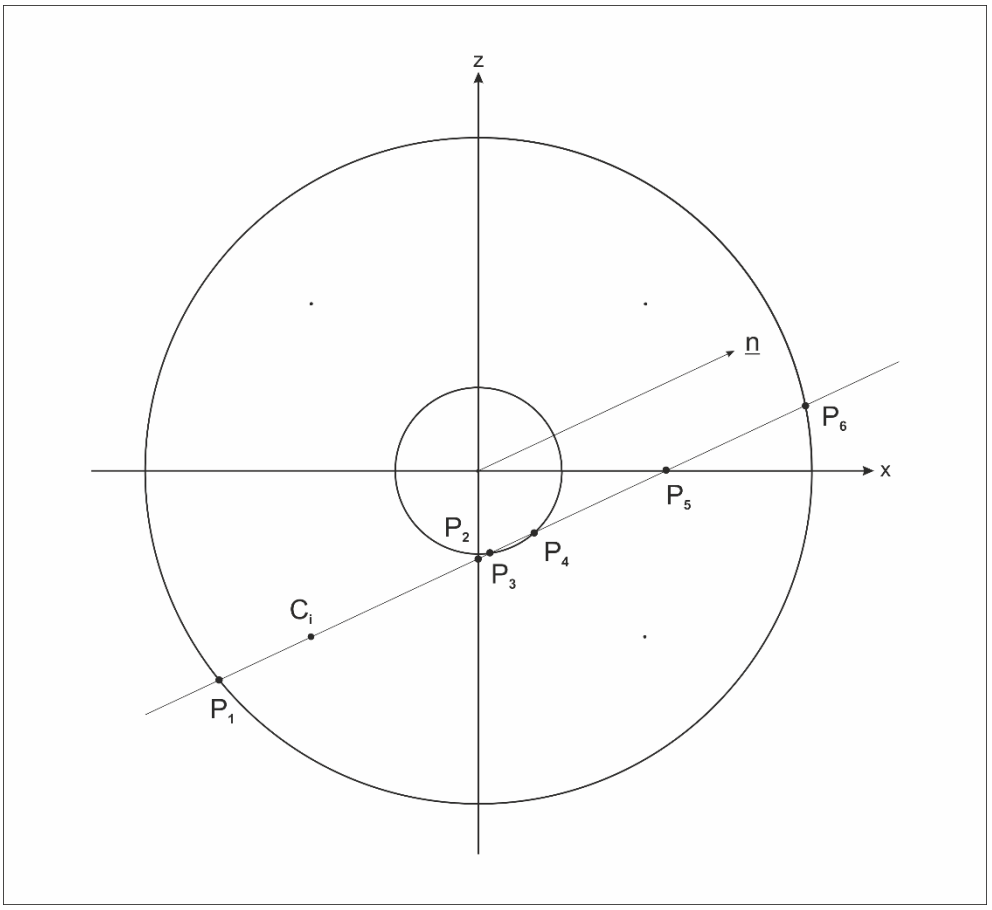
3.2 Maximális lépéstávolság

A maximális lépéstávolság az a távolság, amely a robot mozgás iránya függvényében meghatározza azt, hogy mi az a legnagyobb lépés, amely megtétele esetén még éppen egyik láb sem lép ki az elérési tartományából. Ezen szám kiszámításához először ki kell számolni mind a négy lábhoz tartozó lépéstávolságot. A járási algoritmus egyszerűsítése érdekében a négy lépéstávolság minimumát vettem, mert így biztosan egyik láb sem fog kilépni a saját területéről. Ezzel az algoritmust lábmozgatásért felelős

részét nagyban leegyszerűsítettem, mert így minden láb ugyanakkorát fog lépni, így nem szükséges külön lábanként figyelni az aktuális pozíciójukra, hiszen a minimum választásának köszönhetően biztosan tudom azt, hogy minden láb benne marad a mozgási tartományában. Ez a robot mozgására kevés hatással van, viszont az algoritmus implementálását nagyban megkönnyíti.

3.2.1 Láb pályájának metszete a lépési tartománnyal

A fent említett lábankénti lépéstávolság meghatározásához először az elérési tartomány és a láb pályájának metszéspontjait számoltam ki, mivel a láb pályája minden esetben érinti az elérési tartomány középpontját, így minden esetben két metszéspont keletkezik. A számítás során kihasználtam azt, hogy az elérési tartomány egy 90°-os, körívhez hasonlít, mert így a számítás során felhasználhattam azt, hogy a négy láb elérési tartománya leír egy teljes kört, melyet a síknegyedek határolnak. A számítás során elegendő volt a lábak forgáspontját az origóba tolnom, így az elérési tartományt határoló két körív, a négy lábra vonatkozóan két koncentrikus kört alkotott. A tartományt határoló két egyenes pedig a tengelyekre esett. A továbbiakban ezt használom fel a lépéstávolság kiszámítására, amit a 16. ábra mutat be.



16. ábra: A maximális lépéstávolság számításához felhasznált mozgási tartományok egy körbe tolva.

A metszéspontokat egy láb C_i középpontjából számolom ki, ezért minden lábhoz tartozik egy síknegyed, és az ehhez tartozó környegyedek leírják az adott lábhoz tartozó elérési tartományt. A síknegyedeket egy az arra vonatkozó vektorral írtam le, melynek mindkét koordinátájának abszolútértéke egy, tehát csak az előjelükben van különbség. Ezen vektorokkal így könnyedén tudtam számításokat, eltolásokat végezni. A különböző síknegyedekhez tartozó értékeket az 2. táblázat mutatom be.

Láb	x	z
Jobb első	1	1
Jobb hátsó	1	-1
Bal első	-1	1
Bal hátsó	-1	-1

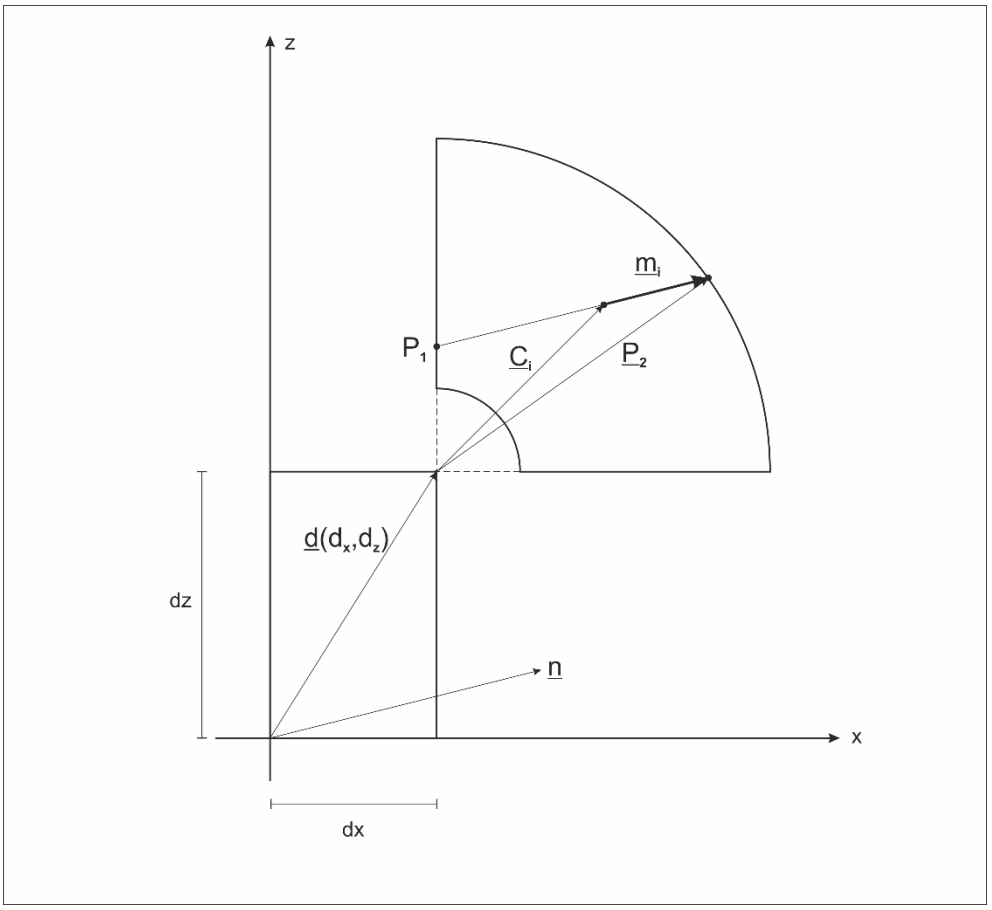
2. táblázat: Az eltolásvektorhoz tartozó koordináták síknegyedenként.

A lábhoz tartozó pálya egyenesét megszorozva az eltolásvektorral kiválasztom azt, hogy melyik síknegyedhez tartozó egyenes metszéspontjait számolom ki. Függetlenül a kiválasztott síknegyedtől kiszámolom az összes metszéspontot. Egy egyenesnek és egy körnek legfeljebb két metszéspontja lehet, míg két egyenesnek egy. Ezért a két körből és két egyenesből összesen legfeljebb hat metszéspont keletkezhet, melyeket a 16. ábra segítségével megjelöltem P_1-től P_6 pontokkal. Minden pontot megszorzok a fent említett eltolásvektorral, így csak azon pontok kerülnek a két kör esetében a pozitív tartományba, a két tengely esetében a két kör sugarának pozitív értékei közé, melyek a keresett elérési tartományhoz tartoznak. A megfelelő tartományba eső pontokat elmentem egy hatelemű tömbbe, a pontok megtalálásának sorrendjében. Így leellenőrizhető az, hogy az algoritmus tényleg csak két pontot talált.

3.2.2 Lépéstávolság

A maximális lépéstávolság kiszámítása előtt a négy láb lépéstávolságát számolom ki. A 3.2.1 részben leírtak alapján megkapott P_1 és P_2 pontokkal tovább számolom a robot lépéstávolságát. A kettő közül az lesz a megfelelő, melyet kivonva az elérési tartomány $\underline{C_i}$ középpontjának vektorjából a robot mozgásával közel megegyező irányú vektort kapok, ezt az 17. ábra közvetítésével mutatom be az $\underline{m_i}$ vektor segítségével. A számítás során a fentiekhez hasonlóan felhasználtam egy epszilon változót a pontatlanságból eredő hiba elkerülésére.

A négy lábhoz tartozó lépéstávolság kiszámítása után a maximális lépéstávolság kiszámítása már nem okoz problémát. Ugyan a változó neve ebből a környezetből nézve pontatlanul azon legnagyobb távolságra utal, melyet még minden láb leléphet, mielőtt elérné az elérési tartomány szélét. Ezek után egyértelművé válik az, hogy a négy lépéstávolság minimumát kell vennem annak érdekében, hogy soha ne léphessen ki egyik láb sem az elérési tartományból.



17. ábra: Maximális lépéstávolság kiszámolása lábanként.

3.3 Lépéstávolság kiszámítása a teljes távolság függvényében

A 3.2 előbb fejezetben részletezett maximális lépéstávolság számításakor a robot fizikai korlátait használtam fel, azonban a tényleges lépéstávolság kiszámításhoz további paramétereket is figyelembe kell venni. Ezek a lelépendő távolság, illetve az út alatti elfordulás, melyet a későbbiekben relatív elfordulásnak nevezek.

Jelen helyzetben a robot mozgása nem egyenletes, ha a lelépendő távolság nem egész többszöröse a maximális lépéstávolságnak, így a lépések legvégén egy maradék, kisebb lépés marad, mely megtöri a mozgást. Ezen maradék lépés eltűntetése érdekében a lelépendő távolságot elosztom a 3.2 fejezetben számolt maximális lépéstávolsággal, így megkapom az út megtételéhez szükséges lépésszámot. A lépésszám számolásakor a maximális lépéstávolságot használtam, ezért az így kapott eredmény a lépésszám minimumát képezi. Mivel a robot csak egész számú lépést tud megtenni, ezért az előbb számított lépésszámot egész számra felkerekítem, majd elosztom vele a lelépendő távolságot, így megkapva azt az új lépéstávolságot, mellyel immár egységes lépésekkel

lépi le a robot a kiadott útvonalat. A fentihez hasonló módon kiszámolom az elfordulási szöget lépésenként.

Abban az esetben, ha a robotnak forognia is kell mozgás közben, elképzelhető olyan variáció is – jellemzően kis lelépendő távolság esetén –, ahol a kiszámolt lépésszám megtétele alatt a robot nem tud elegendőt forogni. Ekkor újra számolom a lépésszámot a maximális elfordulással lépésenként. A relatív elfordulást elosztom a maximális elfordulással, majd ezt a fentihez hasonló módon felkerekítem, és kiszámolom a lépéstávolságot és az elfordulási szöget.

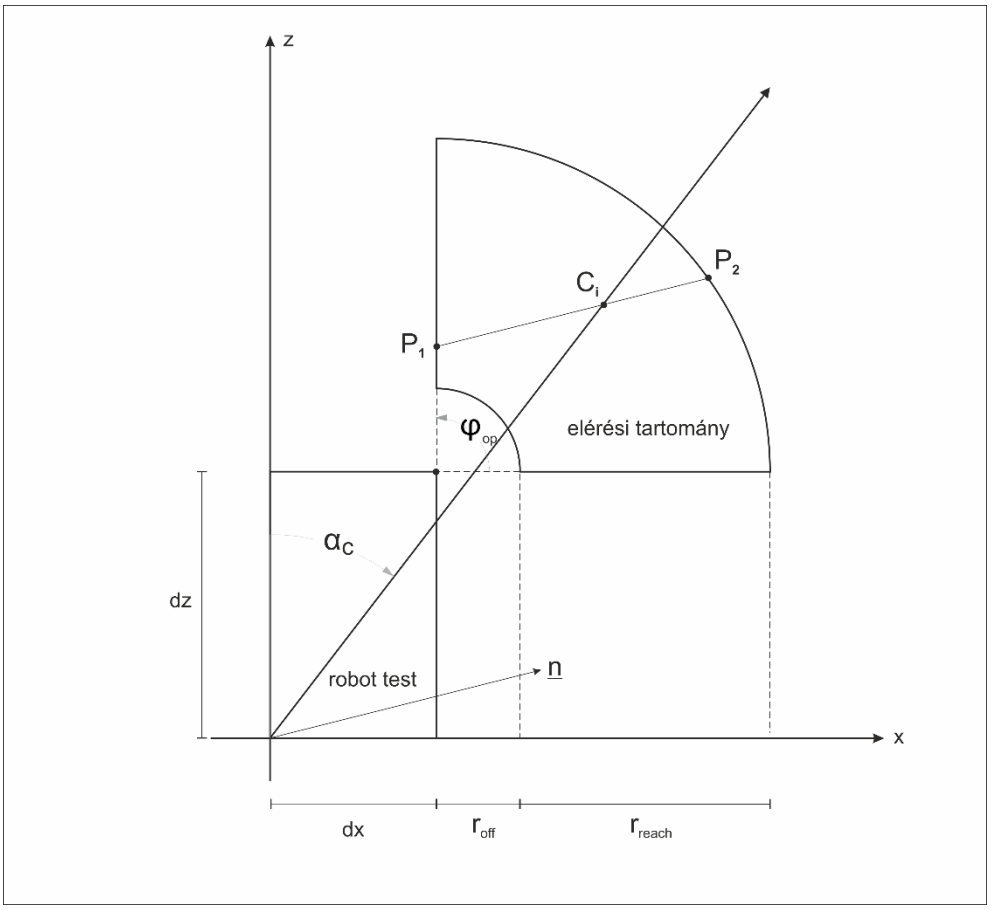
3.4 Lépéssorrend kiválasztása mozgásirány függvényében

A robot lábainak lépési sorrendjét folyamatosan változtatom annak függvényében, hogy a robot milyen irányba mozog. A lábak lépésének sorrendjét a $\underline{n}$ mozgásirány vektoron kívül egy úgynevezett α_c kritikus szög függvényében is változtatom, melyet bővebben a 3.4.1 részben fejtek ki. A robot lábainak tényleges lépési sorrendjét az előbb említett két paraméter függvényében a 3.4.2 pontban részletezem.

3.4.1 Kritikus szög

Lépés során a robot egy lába a levegőben van, míg a másik három lába a földön. A földön lévő lábak egy háromszöget alkotnak. Annak érdekébe, hogy a robot ne dőljön fel a tömegközéppontját ezen háromszögön belül kell tartania. Az α_c kritikus szög ezen háromszög határát adja meg. Az α_c szöget a robot tömegközéppontjából és a láb elérési tartományának C_i középpontjából számolom az (3.4) egyenlet segítségével, amennyiben a robot tömegközéppontja az origóban van [13]. Az algoritmus tervezése során még nem volt kész a robot, így ezen egyszerűsítést alkalmaztam. A kritikus szöget, valamint a robot mozgását a 18. ábra mutatom be. A robot lépéssorendjét az α_c szög befolyásolja annak érdekében, hogy a robot mozgás közben ne dőljön el.

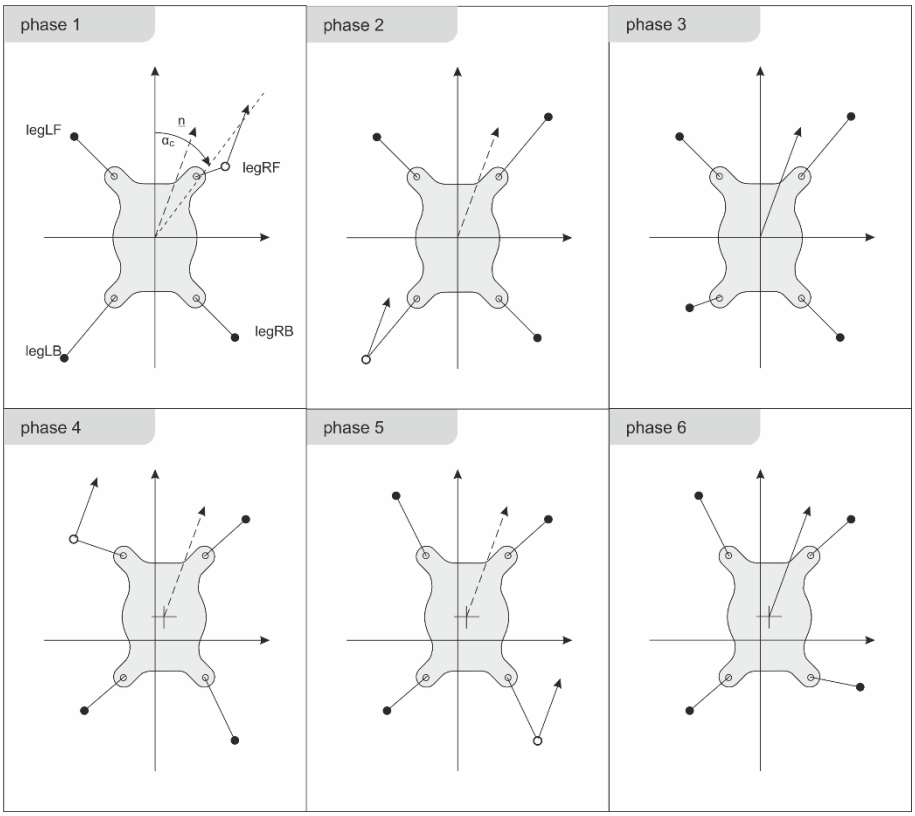
$$\alpha_c = \tan^{-1}\left(\frac{dx + C_{ir} * \cos C_{i\varphi}}{dz + C_{ir} * \sin C_{i\varphi}}\right) \tag{3.4}$$



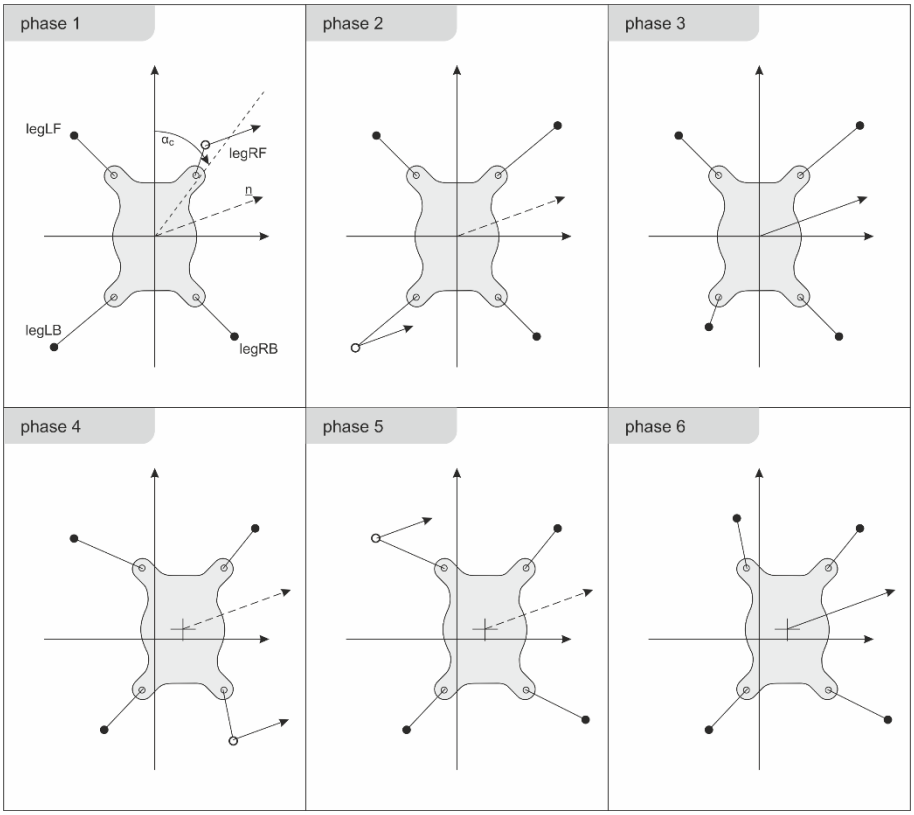
18. ábra: A robot testének és egy láb elérési tartományának kapcsolata, α_c szög elhelyezkedése.

3.4.2 Lépés sorrend

Az αc szög ismeretében meghatározható a robot lépésének sorrendje, mely kizárólag a robot mozgási irányától függ. Először az első két láb sorrendjét döntöm el, a robot $\underline{n}$ mozgásirány vektor alapján, úgy, hogy megnézem melyik síknegyedbe esik. Először azon láb lép, mely az $\underline{n}$ vektorral egy síknegyedben van, majd az ezzel szemben lévő láb. Két láb előremozgatása után a testét mozgatja előre a lépéstávolság mértékével az $\underline{n}$ vektor irányába. A fennmaradó két láb sorrendjét az n vektor és α_c szög viszonya dönti el. Mindig az a láb lép harmadjára, amelyikhez közelebb van az $\underline{n}$ vektor. Végül újra a testét mozgatja a robot. A mozgás szemléltetése érdekében két lépési sorrendet mutatok be, az egyiket a 19. ábra alapján, melyben a robot majdnem teljesen előremozog, a másikat a 20. ábra melyben majdnem teljesen oldalra mozog a robot. Jól megfigyelhető a különbség a két eset között, a negyedik állásban másik láb mozog.



19. ábra: Robot mozgása előre, kicsit jobbra. Lépés sorrend: [RF, LB, Test, LF, RB, Test]



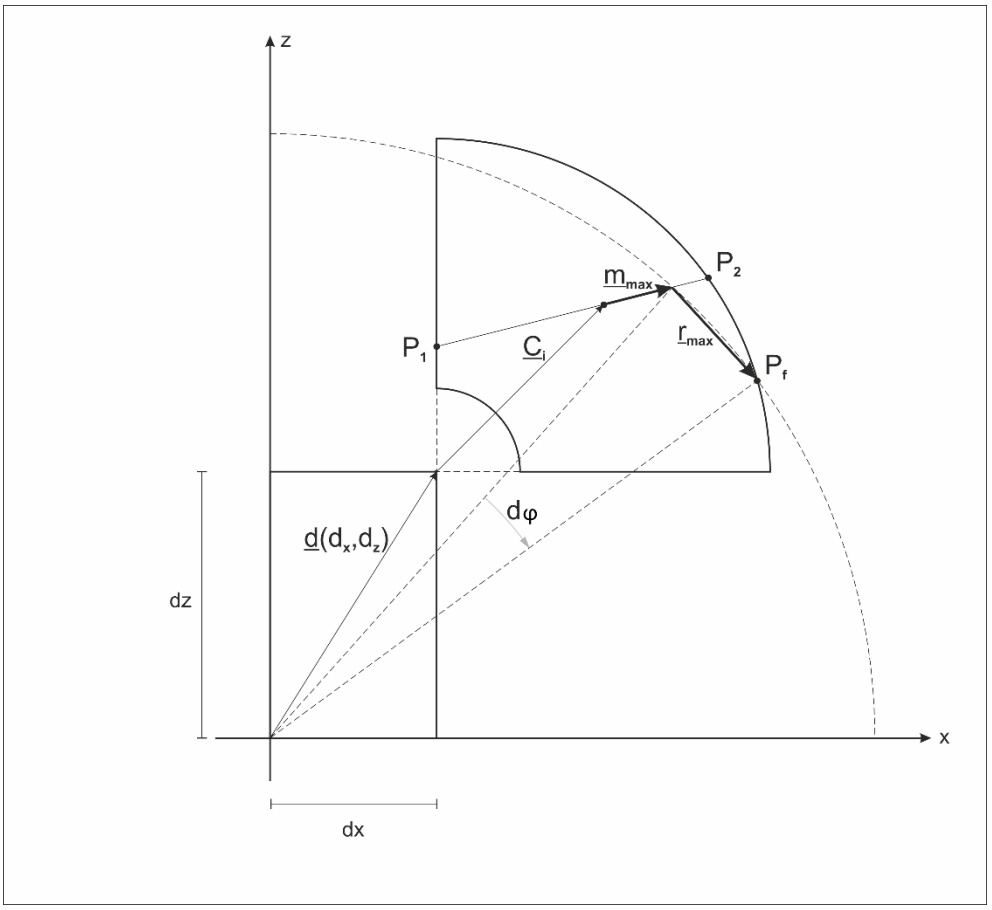
20. ábra: Robot mozgása oldalra, kicsit előre. Lépés sorrend: [RF, LB, Test, RB, LF, Test]

3.5 Új lépés pozíciójának meghatározása

Az algoritmus során a robot lépéseinek új célpozícióját határozom meg, melyek több eltolás segítségével alakulnak ki. Először az egyenesen mozgó – orientáció változtatás nélküli – mozgást valósítottam meg, majd utána egészítettem ki a forgó mozgással.

A robot lábának új célpozíciójának kiválasztását a 21. ábra mutatom be. A láb P_f végpontjának számítása a C_i pontból indul, melynek kiszámítását a 3.1.3 bekezdésben részleteztem. A pontot az előbb említett ábrán látható $\underline{d}$ és $\underline{C_i}$ vektorok összegével adtam meg. A továbbiakban ezt a pontot megszorzom a 2. táblázat szereplő értékekkel. Az új célpont meghatározása eddig a mozgás irányától független volt, viszont a továbbiakban a láb pályájával párhuzamos $\underline{m}_{max}$ vektorral tolom el a P_f végpontot, mely így annak a robot mozgás irányú összetevőjét adja meg. Az $\underline{m}_{max}$ vektor hossza a lépés távolsággal egyenlő, melynek számítását a 3.3 bekezdésben mutattam be. A P_f végpont forgás irányú összetevőjét az $\underline{r}_{max}$ vektor adja meg. A vektorral a P_f pontot egy az $\underline{m}_{max}$ vektor végpontjába állított körív mentén tolom el. Hosszát a körív mentén a $d\varphi$ elfordulási szög adja meg, melyet szintén a 3.3 bekezdésben mutattam be.

Forgással rendelkező mozgás esetében a robot testének mozgatása és forgatása után korrigálnom kellett a mozgás irányát annyival, amennyivel a robot teste elfordult, hogy így a robot továbbra is ugyanabba az irányba mozogjon. E nélkül a korrigálás nélkül a robot minden lépési szekvencia után másik irányba indult el, nem egy egyenes vonalon mozgott.



21. ábra: A láb új célpozíciójának meghatározása, $\underline{m}_{max}$ a mozgás irányú összetevő, $\underline{r}_{max}$ a forgás irányú összetevő.

4 Szimuláció

Mielőtt a járási algoritmust ténylegesen implementáltam volna a firmware-be, melyet a 2.3. fejezetben mutattam be, le kellett tesztelnem annak működését is. Erre legalkalmasabb egy szimuláció használata, melyben előre elkészíthettem a mozgástervezést anélkül, hogy a megjelenő hibák tönkretették volna a robotot. Szerencsémre Babits Mátyás is gondolt erre munkája során, így elkészített egy szimulátort, kifejeztetetten ezen robot mozgási algoritmusának megtervezéséhez. [19] Annak ellenére, hogy a szimuláció kifejezetten ehhez a robothoz készült, számomra nehéz volt lecserélni a régi mechanikai modellt az általam tervezettre, valamint az inverz geometriai modell új roboton való alkalmazása során is kihívásokba ütköztem. A szimuláció a kezdeti nehézségek ellenére kifejezetten hasznosnak bizonyult, mivel a szimulátor forráskódjának azon része, ami a robot lábainak irányításáért felel, teljesen megegyezik a firmware-en lévő függvényekkel.

A Babits Mátyás által készített szimulátort átalakítottam úgy, hogy az általam készített 3D modellel is problémamentesen működjön. A szimulátorba nem teljesen ugyanazon modelleket töltöttem be, mint amelyek ki lettek nyomtatva, mert abban az esetben a szimulátorban kellett volna összeszerelni a robotot is, és az nagyban megbonyolította volna azt. A szimulátorba betöltött elemeket külön a modelltervező programban készítettem el a kinyomtatott alkatrészek alapján. Így a robot szimulációs modellje csak a csuklók miatt áll több darabból. A modellek szimulációba való betöltésekor több hibába is ütköztem, melyeket a 4.1.1 pontban részletezem.

4.1.1 Modell betöltése

A modell betöltése során az elsődleges problémámat az általam használt Standard Tessellation Language (STL) és a Babits Mátyás által használt úgynevezett *.mdl* formátum közötti különbség okozta. Az általa használt formátummal kapcsolatban az interneten semmilyen használható információt nem találtam. Ezért tovább próbálkoztam az általa készített fájlok megnyitásával és a szimuláció tanulmányozásával, az abból szerzett információ felhasználhatóságának reményében, ám sajnos itt sem jártam sikerrel. Próbáltam továbbá az általam használt tervezői környezetből kiexportálni ebbe a formátumba a modellt, azonban így sem jutottam közelebb a célomhoz. Végül közvetlenül Babits Mátyáshoz fordultam segítségért, aki elküldte a főképpen általa

készített konvertert, melynek segítségével immár sikeresen el tudtam készíteni a szimulátorba beolvasható formátumú modell fájlokat. Azért nem jártam sikerrel a próbálkozásaim során, mert a Mátyás által használt fájlok kódolása és dekódolása is egyedi szintaxissal bírt.

A fájlformátum okozta problémák megoldása után ugyan sikerült az általam tervezett modelleket megjeleníteni a szimulátorban, azonban azok számomra rossz orientációban és méretarányban jelentek meg. Ezeket a modell tervező szoftverben az x, y, z tengelyek orientációja, valamint a modell STL formátumba való exportálásának méretaránya okozták. Én elsőre ezt a szimulátorba beépített pozíció, orientáció, méretarány paraméterekhez adott ofszettel terveztem kijavítani, ám itt sem értem el sikereket. A szimulátorban lévő modell úgy épül fel, hogy az egyes lábdarabok hierarchiailag egymás alá tartoznak, így a fő modell méretarányának változtatásával a többi alkatrész is változott. Ez azt eredményezve, hogy míg a szimulációban eredetileg minden paraméter deciméteres méretarányban volt megadva, a robottest modelljének méretarány változtatása után a test belső koordinátarendszere megváltozott centiméteres arányra. Így hiába értem el azt, hogy a robot teste újra jó méretű legyen a szimuláció többi részéhez képest, hiszen az inverz geometriai algoritmus egyszerre dolgozik a roboton kívüli és azon belüli paraméterekkel, így újfent méretarány hibát vittem a számításokba.

A méretarány különbségből adódó hiba mellett megjelent egy másik gond is, mely hatással volt az inverz geometriai számításokra és a lábak megfelelő összeállítására. Ezen probléma a modell szerkesztőben az x, y, z tengelyek orientációjából adódott. A modellek saját koordinátatengelye a legtöbb esetben nem esett egybe a geometriai számítások során felhasználtakkal, így a szimulátorban a várthoz képest másik irányba álltak az építőelemek.

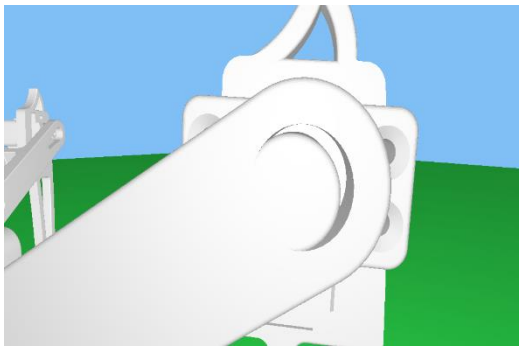
Ezen két probléma megoldásaként - a szimulátor jelentős megváltoztatása nélkül - az kínálkozott, hogy magának a modellnek változtassam meg az orientációját és a méretét. A modellt felépítő elemek orientációit úgy változtattam meg, hogy azok a kinematika során megadotthoz igazodjanak, ezzel kijavítva a hibák nagy részét. Ahogyan fentebb említettem, a szimulátor paraméterei deciméteres mértékegységben vannak megadva, viszont az általam használt tervezőprogram (Inventor) nem alkalmas modell exportálására ebben a méretarányban. Ezért úgy változattam meg a modellek tényleges méretét, hogy a nemlétező deciméteres export helyett centiméteres konverzió után újra

visszakapjam az eredeti méreteket. Az így megváltozatott fájlokkal már tökéletesen működött a szimulátor kinematikai algoritmusa.

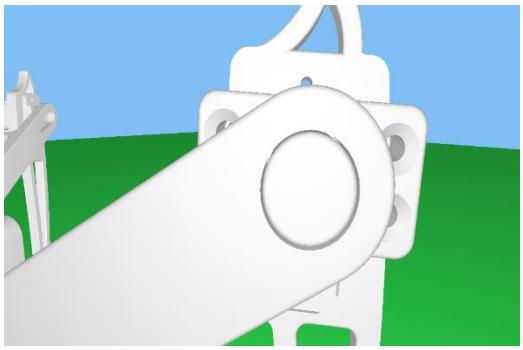
Sajnos a fenti problémák orvoslása igen sok időt vett igénybe a rendelkezésre álló gyakorlati időből, viszont ennek ellenére bizton kijelenthetem, hogy nagy szükség volt a szimulátor használatára a járási algoritmus tervezésekor. A szimulátor nagy biztonságot nyújtott a robot mozgásának vizsgálatakor és tesztelésekor is. Így, annak ellenére, hogy a program működésbe hozására kénytelen voltam a vártnál több időt szánni, mégis határozottan megtérült a befektetett energia.

4.1.2 Modell paramétereinek kerekítési hibái

A modell file-ok sikeres betöltése után egyértelművé vált az, hogy az Önálló laboratóriumom során a 3D-s terv paramétereinél használt kerekítések túl nagyok és az ebből adódó hibák egyértelműen szembetűnőek voltak. A kerekítési hibákra egy példát az 22. ábra mutatok, melyet összevetve az 23. ábra látszik igazán a különbség. A hiányosságok észrevétele után ezredmilliméteres pontossággal adtam meg minden paramétert, melyeket a 1. táblázat részletezek.



22. ábra: a régi, hibás paraméterekkel rendelkező szimuláció



23. ábra: az új paraméterekkel pontosított szimuláció

Nagy valószínűséggel az ilyen magasfokú pontosság alkalmazása értelmetlen, viszont az algoritmusok használatában ez nem okozott erőforrás problémát. Számomra fontos tanulság az, hogy robot modelljét azon paraméterek alapján érdemes megtervezni, amelyek később a számításokban is felhasználásra kerülnek.

5 Áttérés a fizikai robotra

A fizikai robotra való áttérés során először annak lábaihoz és a szervok mozgatásához tartozó új paramétereket kellett beállítanom, azaz kalibrálnom kellett az eszközt. A beállítást és annak paramétereit az 5.1 alfejezetben tárgyalom. továbbá a 5.2 pontban részletezem a járási algoritmus firmware-be való implementálását és annak hibáit.

5.1 Robot kalibrálása

Mielőtt a szimulátorban már jól működő algoritmust kipróbálhattam volna magán a roboton, először kalibrálnom kellett azt. A robot összeszerelése során az utolsó lépést kihagytam, mely a szervokarok feltétele volt. A végső összeszerelésből és a fizikai robot működéséhez elengedhetetlen paraméterek beállításából épül fel a robot kalibrálása. Ezen paraméterek kiegészítik a 2.1 bekezdésben leírtak. A lábhoz tartozó értékeket az 5.1.1 és az 5.1.2 pontokban részletezem őket. Annak érdekében, hogy a szervomotorok univerzálisabbak legyenek, külön darabban adják magát a motort és a hozzá tartozó szervokart. Ennek az oka az, hogy annak ellenére, hogy magának a motornak a működési tartománya limitált, mégis többféleképpen összeszerelhető. Az általam használt motor adatait az 5.1.3 részben fejtem ki bővebben.

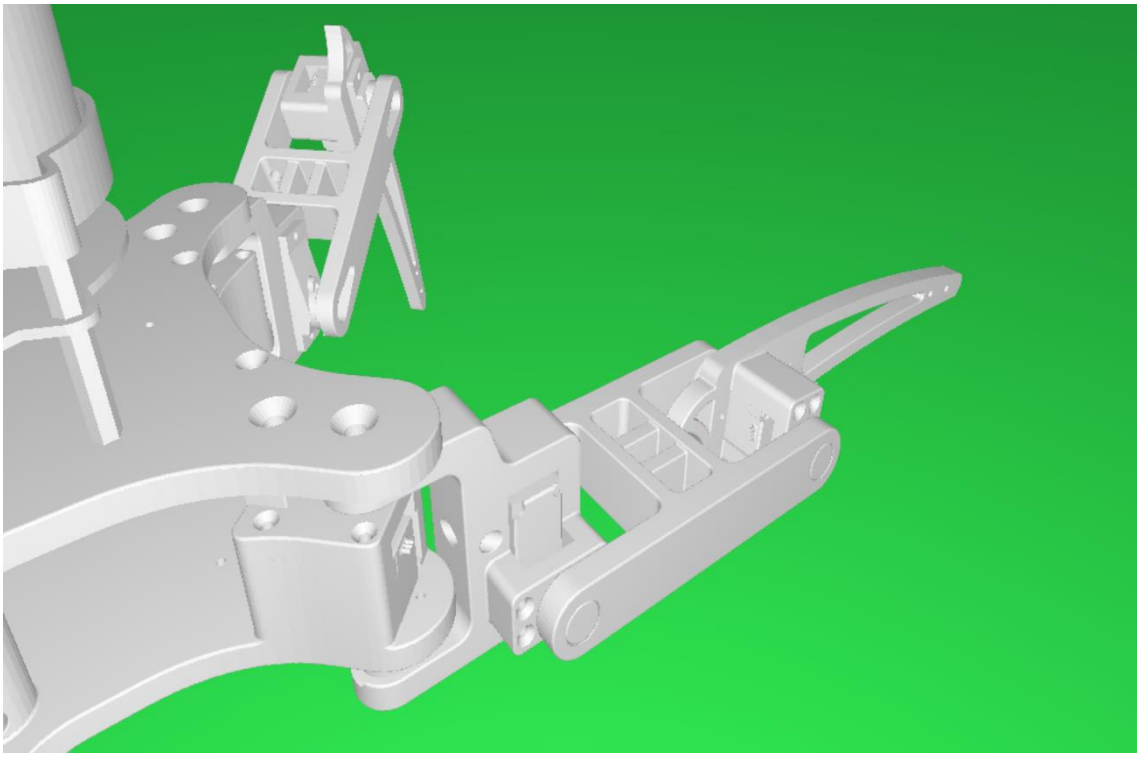
5.1.1 Ofszet szögek

A robot lábaihoz tartozik három paraméter, melyek a robot csuklóihoz tartozó ofszet szögek és a csuklók nulla helyzetének beállítására használhatók [10]. Ezeket a paramétereket meg kellett változtatnom, ugyanis az általam választott új szervok ugyan erősebbek, viszont forgási tartományuk kisebb, részletekbe menően a 5.1.3 bekezdésben mutatom be őket [18]. Az új szervok beállítása során azt tapasztaltam, hogy a motorok a középértéküktől $\varphi = +/- 0.34\pi$ radiánba állíthatóak, szemben a Babits Mátyás által használtakkal, melyek $\varphi = +/- \pi$ forgási tartománnyal rendelkeztek, amire a továbbiakban még visszatérek az 5.1.3 pontban. A kisebb mozgási tartomány miatt nekem ezen paramétereket pontosabban be kellett állítani. Az 24. ábra mutatom be azt az esetet, amikor az a_1, a_2, a_3 szögek mindegyike nulla helyzetben áll. Jól látható, hogy a motorok mozgatásához ezen nullértékek nem előnyösek, hiszen azoknak - a jó működés érdekében - a mechanikai mozgási tartományuk közepére kellene esniük. Az új paraméterek

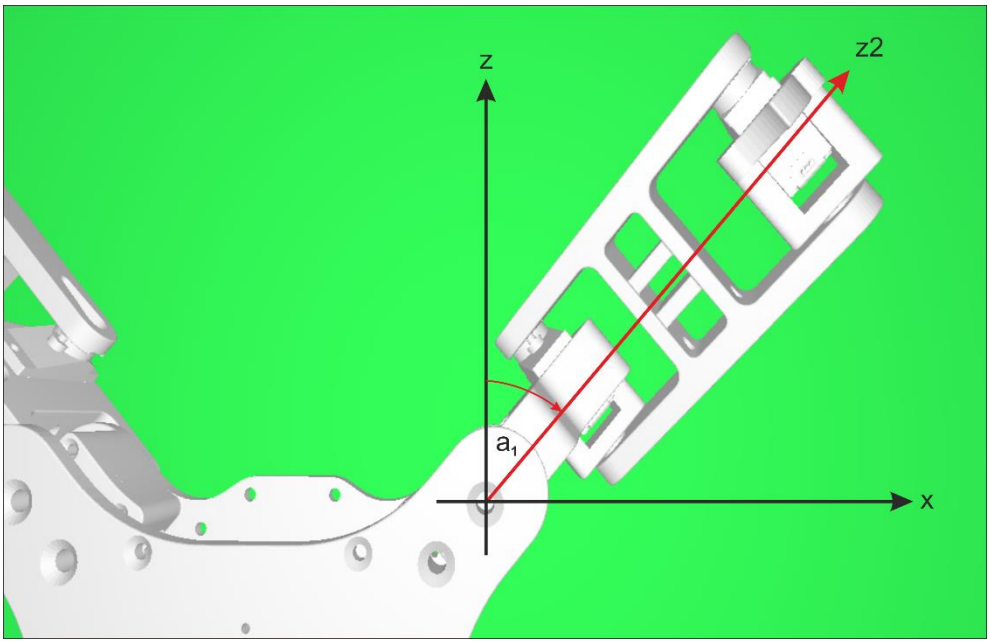
beállításakor - melyek megértését az 25. ábra és 26. ábra segítik - fontos volt számomra, hogy a motorok mozgási tartományát maximalizáljam. Ezért az a_1 szöget 45 fokra, azaz 0.25π radiánra állítottam, így a láb a null érték megadása esetén az elérési tartományának közepére mutat. Az a_2 és a_3 meghatározásakor hasonló elvet követtem azzal a különbséggel, hogy ezen csuklók esetében a fizikai határok szabják meg azt, hogy a robot mennyire képes behúzni a lábait, így a csuklók mozgási tartományának meghatározásakor a fizikai határokhoz közel állítottam be a mozgási tartományt is. A beállított szögeket a 3. táblázat mutatom be radiánban.

Láb	a_1	a_2	a_3
Jobb első	0.25π	-0.16π	0.5π
Jobb hátsó	0.75π	-0.16π	0.5π
Bal első	-0.25π	-0.16π	0.5π
Bal hátsó	-0.75π	-0.16π	0.5π

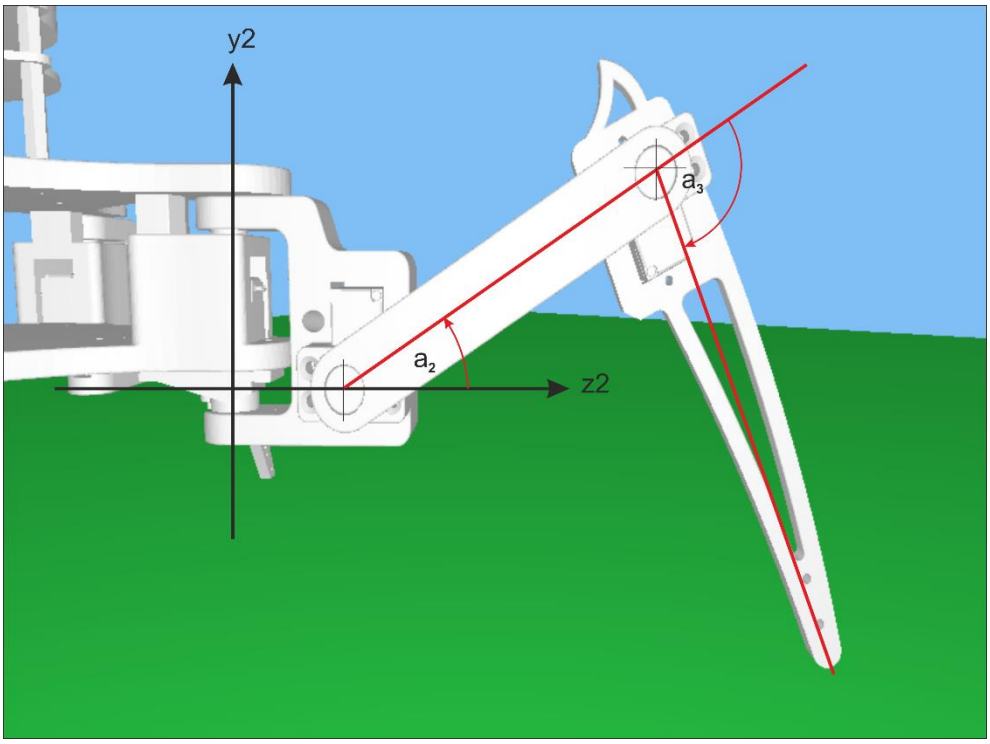
3. táblázat: a motorok beállításakor használt ofszet szögek lábanként, radiánban



24. ábra: a jobb oldali lábon a beállított ofszet szögek nulla értékűek, a bal oldali lábon a 3. táblázat megfelelő értékűek



25. ábra: Az ofszet szögek felülnézetből



26. ábra: az ofszet szögek oldalnézetből

5.1.2 Motorok mozgási iránya

A robot mechanikai kialakítása miatt egyes motorokat fordítva kellett beszerelnem a többiekhez képest. Ezzel a problémával már Babits Mátyás is szembesült munkája során, ezért lábanként létrehozott további három paramétert [20]. A dir_1, dir_2,

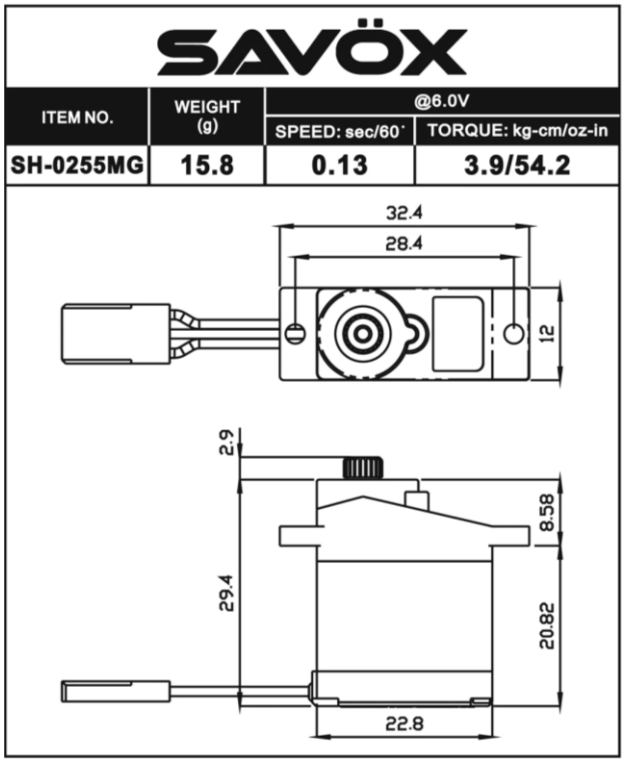
dir_3 változók +1 és -1 értékeket vehetnek fel, ezzel állítva a motorok mozgás irányát. A főként tapasztalati úton beállított változókat az 4. táblázat segítségével mutatom be.

Láb	dir_1	dir_2	dir_3
Jobb első	-1	-1	1
Jobb hátsó	-1	1	-1
Bal első	-1	1	-1
Bal hátsó	-1	-1	1

4. táblázat: a motorok mozgás irányáért felelős parméterek

5.1.3 Szervok beállítása

A robot tervezése során egyik első lépés volt az új robot mozgatásáért felelős szervomotorok kiválasztása [1]. Válogatás során célom az volt, hogy megtartva a micro típusú szervókat [21], akár a sebesség rovására is, minél nagyobb nyomatékú motorra essen a választás. Hosszas keresés után végül a Savöx gyártmányú SH-0255MG típusú szervot választottam, mert a gyártó egyik legerősebb micro típusú szervoja ez volt, mely adatait a 27. ábra mutatom be [18]. A motor 4.8 volt tápfeszültség mellett 3.1 kg-cm-es nyomatékkal rendelkezik, ez több mint másfélszeres teljesítmény növekedést jelent a kiinduló állapothoz képest. A motor fém fogaskerekekkel rendelkezik, így igen strapabíró, emellett 4.8 volt tápfeszültség mellett 0.16 másodperc alatt forog hatvan fokot, mely azt figyelembe véve, hogy nyomaték optimalizált szervot választottam, mégis kimondottan gyorsnak minősül. A szervo gyártói honlapja alapján ugyan a motor teljes működési tartománya 145°, a tapasztalatok nem ezt mutatták, ezért használtam végül a már fentebb megemlített +/-0.34π forgásitartományt, ami 122.4°-nak felel meg.



27. ábra: a robotban használt Savöx szervomotor főbb adatai

A robot végső összeszerelése során, a szervokarok felhelyezésekor igyekeztem a legpontosabban feltenni azokat, azonban fogas kialakításuk miatt csak bizonyos pozíciókban szerelhetőek össze. Így a robot lábaiban már a kiinduló állapotban mechanikai ofszet hiba keletkezett. Ezzel a problémával a korábbiakban Babits Mátyás is szembesült, ezért motoronként létrehozott egy úgynevezett *assemblyOffset* paramétert, mely alkalmas ezen kis hibák szoftveres kiküszöbölésére. Ezeknek értékeit az 5. táblázat mutatom be, az értékeket radiánban jelenítem meg.

Láb	$assemblyOffset_1$	$assemblyOffset_2$	$assemblyOffset_3$
Jobb első	0.0	0.02	-0.1
Jobb hátsó	-0.21	0.1	0.005
Bal első	-0.12	0.02	0.05
Bal hátsó	0.0	0.02	-0.1

5. táblázat: a robot motorjainak összeszerelése során keletkezett ofszetet kompenzáló értékek, radiánban

A fentebb részletezett mechanikai ofszet hibák megjelenése miatt több olyan helyzet is kialakult, mikor a motorok szélső határértékükben a lábak alkotóelemei

összeértek és egymásnak feszültek. Ennek megakadályozásának érdekében létrehoztam motoronként két határértéket, egy *positiveLimit* és egy *negativeLimit* paramétert, ameddig a motorok elfordulhatnak. Ezen változóknak alapértékként beállítottam a már fentebb említett +/-0.34π értéket, így csak azon paramétereket mutatom be az 6. táblázat, melyek értékei változtak az összeütközési lehetőség miatt. A táblázatban azért jelennek meg az 5. táblázat szereplő összeszerelési ofszetek, mert sok esetben ezen kis változtatások is már elegendőek voltak a robot alkatrészeinek feszülésének megakadályozására.

Láb	$positive_1$	$negative_1$	$positive_2$	$negative_2$	$positive_3$	$negative_3$
Jobb első	0.25π	-0.25π	0.34π+0.02		0.34π-0.1	
Jobb hátsó	0.25π	-0.25π		-(0.34π-0.1)		-(0.34π-0.005)
Bal első	0.25π	-0.25π		-(0.34π-0.02)		-(0.34π-0.05)
Bal hátsó	0.25π	-0.25π	0.34π+0.02		0.34π-0.2	

6. táblázat: a robot motorjainak mozgási tartományát limitáló paraméterek, radiánban

5.2 Algoritmus átültetése a fizikai robotra

A robot kalibrálása nem volt nehéz, azonban mégis több időt vett igénybe, mint amivel előzőekben kalkuláltam. Megbonyolította a helyzetet az is, hogy technológiai okokból fordítva szereltem fel a Massár Lóránt Mátyás által készített NYÁK-ot, így jelenleg a robot hátrafelé megy előre, de miután ez csak a motorok bekötése miatt van így, a járási algoritmusban ez nem okozott problémát. Tovább bonyolította a végrehajtást az is, hogy az egyik motor elromlott a limitek beállítása során, így azt ki kellett cserélnem.

A kalibrálás okozta nehézségeket viszont enyhítette az, hogy amikor ez elkészült, akkor a szimulátor hasonlósága miatt az abban elkészített járási algoritmust csak át kellett másolnom a firmware kódjába. Futtatás után a robot szinte tökéletesen végrehajtotta a mozgást. Az 5.1.2 részben kifejtett mozgási irányok beállítása után már jól lépegetett a robot. A tesztelés elején még némi problémát okozott az is, hogy a robot nehezen bírta el a saját testtömegét, ám később kiderült az, hogy a kalibrálás során egy másik motor is

megsérült, ezért nagyobb áramot vett fel, majd pedig le is állt. A motor kicserélése után már kiválóan, s egyben problémamentesen működött a berendezés.

6 Vezérlés ROS platformon

A robot ROS platformon keresztül történő mozgásvezérléséért egy a roboton futó ROS node osztály felel. Minden ROS platform alá tartozó egységet ROS nodenak nevezünk, melyek egyenként különálló egységeknek felelnek meg. Majd ezen nodeok kommunikálnak egymással. Ennek feladata az üzenetek küldése és fogadása a számítógép és a robot között. A mozgatásért felelős üzenet fogadása a ROS node osztály egy subscriber típusú tagváltozóján keresztül történik, mely a ROS core-t futtató számítógépről kerül kiküldésre. Ugyan a mozgást vezérlő tagváltozó már létezett korábban is, a járási algoritmus megváltozásával megváltoztak azon paraméterek, melyeket a robot vár a ROS rendszeren keresztül. Az üzenet pontos tartalmát a 6.1alfejezetben, míg annak implementálását a 6.1.1 pontban mutatom be. A robot ROS platformon keresztüli vezérlésének tesztelését, a 6.3 bekezdésben részletezem.

6.1 Robot mozgatásért felelős ROS message

A robot ROS alapú mozgásvezérlés tervezése során a két lehetőséget vizsgáltam meg; a sebesség-szögsebesség- és a pozíció-orientációalapú vezérlést. Előbbit a 6.1.1pontban, míg utóbbit a 6.1.2 pontban fejtem ki bővebben, valamint részletezem előnyeiket és hátrányaikat.

6.1.1 Sebesség-szögsebesség alapú vezérlés

Ahogyan azt már a 2.3 fejezet elején is ismertettem, a robot járási algoritmusa egy relatív pozíciót és egy orientációt vár. Ez felveti a kérdést, hogy miért fontoltam meg a sebesség-szögsebesség alapú vezérlést, hogyha az algoritmus nem így várja? A legtöbb ROS alapú globális és lokális pályatervező sebesség-szögsebesség párokkal vezérli a robotokat, mert így a robot mozgása közben is beleszól annak irányításába, ezzel jobb vezérlést elérve.

A vezérlés implementálása a jelenlegi mozgás algoritmusba megoldható, ám némi bonyodalmakkal jár. A tényleges mozgást végrehajtó rész first in first out (fifo) alapú, így az általam írt algoritmus csak elmenti a kiszámolt pozíciókat, melyeket később végrehajt. Ezért sebességre való vezérlés esetében elengedhetetlen egy minimumút választása, ami alapján vezérelhető lesz a robot. Emiatt ez a módszer – a jelenlegi mozgás végrehajtásért

felelős alrendszert alkalmazva – a hatékony implementálást megnehezítő procedúra bevezetését igényelte volna.

6.1.2 Pozíció-orientáció alapú vezérlés

A sebesség-szögsebesség alapú vezérléssel szemben a pozíció-orientáció alapú vezérlést nagyságrendekkel egyszerűbb implementálni, ugyanis az algoritmust meghívó *AddPathElementMove()* függvény eleve a robot új pozícióját és új orientációját várja. A vezérlés egy nagy hátránya a 6.1.1 pontban részletezettekkel szemben az, hogy az általános ROS által használt pályatervező algoritmusok nem képesek ezzel az eljárással irányítani a robotot. Viszont a fenti függvény megvalósításával kvázi egy ROS lokális pályatervezőt készítettem, mert ezen tervezők is célpozíciót kapnak paraméterül. [22]

6.1.3 Konklúzió

A sebesség-szögsebesség alapú vezérlés tehát pontosabb működést lehetővé tevő, több beavatkozási lehetőséget felkínáló, kreatívabb vezérlést megvalósító robotizációt biztosít. A szakdolgozatom készítése során azonban a rendelkezésre álló időkerettel is gazdálkodnom kellet, ami viszont az időhatékony megoldás figyelembevételét helyezte előtérbe. Emellett már a járási algoritmus megtervezésének kezdetén eldöntöttem azt is, hogy nem kívánok változtatni a járást végrehajtó modulon. Így végül a pozíció-orientációt alkalmazó vezérlés mellett döntöttem, melynek implementálását a 6.2 bekezdésben ismertetem.

6.2 ROS interfész firmware oldalon

A robot által várt ROS üzenetet egy *geometry_msgs::Pose* típusú üzenetként [23] képezi le a programom, melyben egy három koordinátából álló pozíciót, valamint egy kvaterniókban megadott elfordulási szöget adok meg. Míg előbbivel a robot új pozícióját definiálom, addig az utóbbival a robot elfordulását határozom meg. A paraméterekből nem használom fel az összes értéket, csak a pozíciónak az *x*, *y* koordinátáját, valamint az *x* kvaterniót. Az üzenet lekezelését az alábbi kódrészlet tartalmazza:

```
void ROSNode::walk_inCommandCallback(const geometry_msgs::Pose& pose_msg)
{
      //Get position from pose
      walk_Command.x = pose_msg.position.x;
      walk_Command.y = pose_msg.position.y;

      //Get heading from quat
      walk_Command.phi = acos(pose_msg.orientation.x) * 2.0f;
```

```
        walk_command_status = true;
}
```

Jól látható, hogy a pozíció és szög paramétereinek átvétele igen egyszerű. Az üzenet callback függvénye a gyors lefutás érdekében csak átmásolja a pozícióváltozókat, valamint a kvaterniókból kiszámolja a robot elfordulásának paraméterét. Továbbá a program „igaz” állapotba állít egy flaget, mely jelezi, hogy kiértékelhető a *walk_Command* tagváltozó tartalma.

Az adatok kiértékelése előtt a ROS node futtatásáért felelős taszk a *walk_command_status* „igaz” értéke esetén elküldi az adatokat a járásért felelős taszk számára. Majd a járás működtetésért felelős taszk fogadja az üzenetet, végül a *walk.StoreCommand()* függvény segítségével eltárolja azt:

```
while(pdFALSE == xMessageBufferIsEmpty(RosToWalkMessage))
{
      xMessageBufferReceive(RosToWalkMessage,(void*)&(walk_Command),
sizeof(WalkCommand), 0u);
      walk.StoreCommand(walk_Command);
}
```

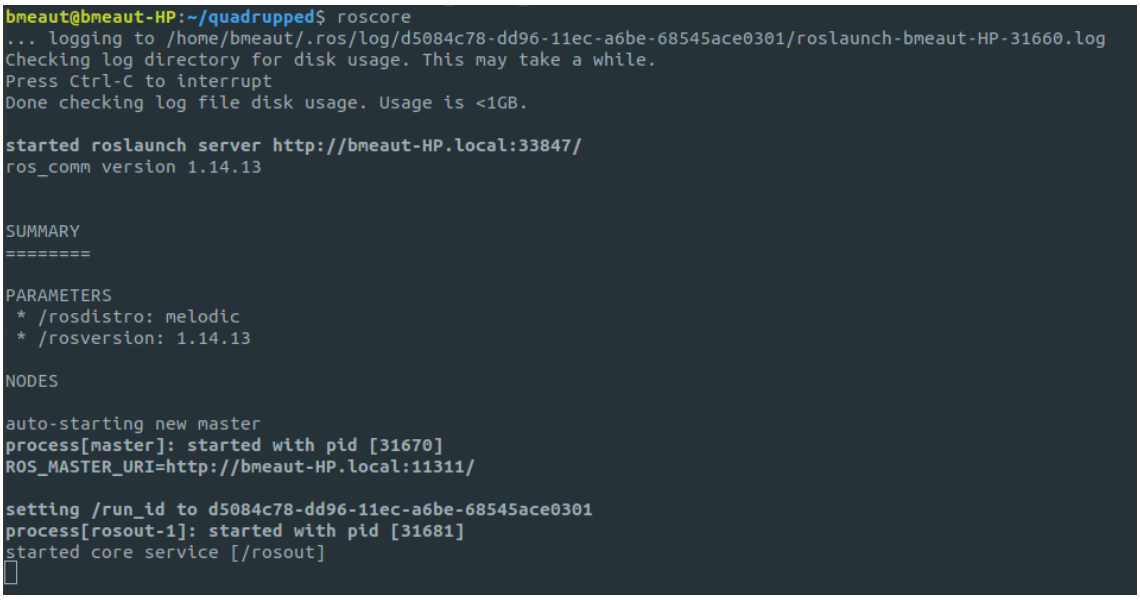
A *walk.StoreCommand()* függvény már csak az általam készített *AddPathElementMove()* függvényt hívja meg, mely végrehajtja a szükséges számításokat és elmenti a lábak és a test pozícióit a mozgás végrehajtásához.

```
void Walk::StoreCommand(WalkCommand command)
{
      m_walkScipt.AddPathElementMove(mth::float2(command.x,command.y),
command.phi);
}
```

A paraméterek és az algoritmus értelmezése viszonylag egyszerű feladat volt, így ez példa is jól reprezentálja azt, hogy miért ennyire elterjedt a ROS használata, és miképpen garantálja a hatékony alkalmazás elsajátítását. A robot ROS-on keresztüli tesztelése is hasonlóan egyszerű, s egyben egyértelmű volt, ezt a 6.3 fejezetben fejtem ki bővebben.

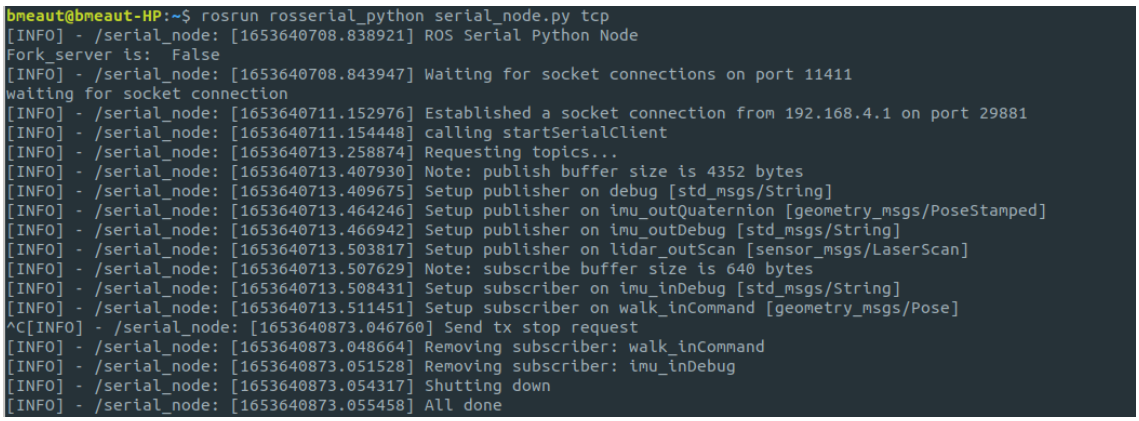
6.3 ROS alapú vezérlés tesztelése

A robot ROS interfészének tesztelése érdekében valódi ROS üzenetekkel teszteltem le. A robothoz a rajta lévő ESP-01 wifi modulon keresztül kapcsolódtam a ROS core-t futtató számítógéppel. A kapcsolat megteremtése után a számítógépen elindítottam a ROS core-t 28. ábra szereplő parancsok segítségével.



28. ábra: ROS core elindítása a számítógépen.

A ROS core futása után a ROS Serial node-ot kellett elindítanom, mely kapcsolatot létesíti a kapcsolatot a mikrokontrolleren lévő ROS taszkkal és elrejti a wifis kapcsolatot a többi ROS node-tól. Ilyenkor a többi node számára a roboton futó ROS node úgy látszódik, mintha az a számítógépen lenne. A ROS Serial elindítását a 29. ábra látható parancsokkal végeztem el.



29. ábra: A ROS Serial elindítása.

A roboton futó program teszteléséhez manuálisan küldtem üzeneteket a roboton futó ROS node számára. A robotra küldött üzenet típusát és paramétereinek jelentését a 6.2 bekezdésben mutatom be. Az üzenetek, melyekkel teszteltem a robot mozgását a 30. ábra láthatóak.

```
^Cbmeaut@bmeaut-HP:~$ rostopic pub /walk_inCommand geometry_msgs/Pose -0.5, -2,]0]' '[1, 0, 0, 0]'
publishing and latching message. Press ctrl-C to terminate
^Cbmeaut@bmeaut-HP:~$ rostopic pub /walk_inCommand geometry_msgs/Pose '[-0.5, -20]' '[1, 0, 0, 0]'
publishing and latching message. Press ctrl-C to terminate
bmeaut@bmeaut-HP:~$ HP:~$ rostopic pub /walk_inCommand geometry_msgs/P2, 0, 0]'5'[0.92388, 0, 0, 0]'
publishing and latching message. Press ctrl-C to terminate
^Cbmeaut@bmeaut-HP:~$ rostopic pub /walk_inCommand geometry_msgs/Pose '[2, 0, 0]'[0.92388, 0, 0, 0]'
publishing and latching message. Press ctrl-C to terminate
^Cbmeaut@bmeaut-HP:~$ rostopic pub /walk_inCommand geometry_msgs/Pose -0.5, -2,]0]' '[1, 0, 0, 0]'
publishing and latching message. Press ctrl-C to terminate
^Cbmeaut@bmeaut-HP:~$
```

30. ábra: A robot számára küldött ROS üzenetek.

A robot mozgása szerencsére egyből jól működött, egyedül a forgás nem volt tökéletes, de ez a kvaterniók átszámolásából következett. A jövőben érdemes lesz a ROS beépített műveleteit használni, viszont a tesztelésre ez a megoldás is tökéletesnek bizonyult. A 31. ábra bemutatom a robotot működés közben, igaz a kép nem képes igazán megmutatni a járást.

31. ábra: Robot működése ROS platformon keresztül

7 Értékelés, továbbfejlesztési lehetőségek

Összességében úgy vélem, hogy a robotot sikerült több új funkcióval is sikeresen kiegészítenem. A robot új modellje egy lényegesen strapabíróbb, erősebb vázzal és az új szervomotorokkal kiegészítve megbízható alapot biztosít a robot számára. A robothoz tartozó paraméterek beállításával újra működőképessé vált a robot és annak kinematikai algoritmusa is.

A mechanikai újítások mellett a robot mozgásáért felelős járási algoritmus is több újításon ment keresztül. Alkalmassá vált – orientációjától függetlenül – a tetszőleges irányú mozgásra, valamint orientációjának változtatására mozgás közben. Ezen járási algoritmus alkalmazásával a robot mozgatása flexibilisebbé vált. A robot mozgatásáért egyetlenegy függvény felel, mely egy új relatív pozíciót, valamint egy relatív elfordulást vár bemenő paraméterként.

A járási algoritmus szükséges egyszerűsítései ellenére is teljesértékű mozgás megvalósítására alkalmas a robot. Sajnos a robot tömegközéppontjának akkumulátor általi eltolása miatt a robot kissé dülöngélve járt. A robot egy lábának felemelésekor az algoritmus nem kalkulálja ki a maradék három láb által határolt háromszög helyzetét, valamint azt sem állapítja meg, hogy robot tömegközéppontja vajon ezen háromszögön belül helyezkedik-e el. Ez megnöveli a robot instabilitását. Minden előzőekben említett hiányosság ellenére a robot mégis teljes mértékben képes bármilyen irányú mozgás megvalósítására. A robot az orientációjának megváltoztatására is képes, ugyanakkor ez a funkció nem működik olyan jól, mint az általános mozgásért felelős rész. A lábak lépési sorrendjének kiválasztásakor - a folyamatos forgás miatt - néha kétszer ugyanazzal a lábpárral lép, emiatt a másik két láb lemarad, ezért egy mozgási parancs kiadásával csak kisebb szöget tud fordulni a robot.

Végül a robot képes volt ROS interfészen keresztül mozgási parancsokat fogadni és végrehajtani. Így a robot új járási stratégiája elérhető ROS-on keresztül is.

7.1 Fejlesztési lehetőségek

Úgy vélem, hogy a munkám a pókszerű robottal egy kerek egészet alkot, ám emellett számos továbbfejlesztési lehetőség adódik. A robot járási modellje korántsem tökéletes. Célszerű lenne megvizsgálni a stabilabb járást biztosító mozgási

algoritmusokat és azok implementálhatóságát, valamint a forgómozgást lehetővé tevő eljárás zökkenőmentesebbé tétele is perspektivikus területnek tűnik.

A jobb ROS alapú vezérlés érdekében a teljes mozgást végrehajtó egységet érdemes úgy továbbfejleszteni, hogy az a sebesség és a szögsebesség paraméterek fogadására legyen optimalizálva. Így a robot alkalmas lenne a ROS-ba épített pályatervező programok integrálására. Jelenleg az új roboton csak annak mozgásának vezérlését teszteltem le a ROS platformon keresztül. Massár Lóránt Mátyás munkája ennél sokkal szerteágazóbb volt, így az Ő munkájának letesztelésére, valamint továbbfejlesztésére is van lehetőség.

A robot további szenzorral való felszerelése további fejlesztési lehetőségeket vet fel, mert a szenzorok bekötéséhez a robot elektronikáján is változtatni kell. Az általam használt új motorok képesek hat voltról működni, emiatt lényegesen erősebbek az elődeiknél. Így az elektronika újratervezése esetén érdemes lenne erre is figyelni. Továbbá a robot mechanikailag is képes további két láb felszerelésére, azonban ezek felszereléséhez a teljes elektronikai újratervezés szükséges, a mikrokontroller lecserélésével együtt, így ez igen távlati cél lehetne. Hasonló bonyolultsággal rendelkező fejlesztési lehetőség a dinamikus mozgási algoritmus implementálása, mellyel akár egyszerre két lábát is felemelve tudna közlekedni a robot.

Ahogyan korábban említettem, még rengeteg továbbfejlesztési lehetőséggel rendelkezik a pókszerű robot. Pont ez az, amiért én is ennyire szerettem rajta dolgozni, hiszen a szakmákon átívelő lehetőségek kifogyhatatlan tárházában mindenki megtalálhatja a számára kihívást jelentő feladatot.

8 Irodalomjegyzék

[1] S. Komáromi, „Pókszerű, járó robot fejlesztése,” 2021.

[2] ROS, „Robot Operating System,” [Online]. Available: https://www.ros.org/.

[3] M. Babits, „Járó robot készítése,” 2018.

[4] L. M. Massár, „Pókszerű, járó robot fejlesztése,” 2020.

[5] Wikipedia, „Lidar,” [Online]. Available: https://en.wikipedia.org/wiki/Lidar. [Hozzáférés dátuma: 25 05 2022].

[6] Wikipedia, „Inertial measurement unit,” [Online]. Available: https://en.wikipedia.org/wiki/Inertial_measurement_unit. [Hozzáférés dátuma: 25 05 2022].

[7] ROS, „ROS Serial,” 01 10 2018. [Online]. Available: http://wiki.ros.org/rosserial. [Hozzáférés dátuma: 26 05 2022].

[8] ROS, „RVIZ,” 16 05 2018. [Online]. Available: http://wiki.ros.org/rviz. [Hozzáférés dátuma: 26 05 2022].

[9] ROS, „Hector SLAM,” 14 04 2014. [Online]. Available: http://wiki.ros.org/hector_slam. [Hozzáférés dátuma: 26 05 2022].

[10] M. Babits, „Direkt és inverz geometria,” in *Járó robot készítése*, 2018, pp. 6-9..

[11] Tappex, „Multisert brass threaded insert,” [Online]. Available: https://www.tappex.co.uk/products/brass-threaded-inserts/multisert. [Hozzáférés dátuma: 20 05 2022].

[12] M. Babits, „Robot mozgása,” in *Járó robot készítése*, 2018, pp. 9-10..

[13] H. Shiego, F. Yasushi és K. Hidekazu, „The gait control system of a quadruped walking vehicle,” in *Avanced Robotics, 1:4,* Online, Taylor & Francis, 1986, pp. 289-323..

[14] M. Shugen, T. Takashi és W. Hideyuki, „Omnidirectional Static Walking of a Quadruped Robot,” in *IEEE Transactions on Robotics, 21:2*, IEEE, 2005, pp. 152-1612.

[15] H. Shigeo, K. Hidekazu és U. Yoji, „The standard circular gait of a quadruped walking vehicle,” in *Advanced Robotics 1:2*, 1968, pp. 143-164..

[16] Y. Jung-Min, „Crab walking of quadruped robots with a locked joint failure,” in *Advanced Robotics, 17:9*, Online, Taylor and Francis, 2003, pp. 863-878..

[17] L. Vo-Gia, M. K. Ig, T. T. Duc, P. Sangdoek, M. Hyungpil és R. C. Hyouk, „Improving traversability of quadruped walking robots using body movement in 3D rough terrains,” in *Robotics and Autonomous Systems 59:12*, 2011, p. 1036.1048..

[18] Savöx, „SH0255MG - Micro Digital Servo,” [Online]. Available: https://www.savoxusa.com/collections/micro-servos/products/savsh0255mg-micro-digital-mg-servo-13-54#technical-details. [Hozzáférés dátuma: 20 05 2022].

[19] M. Babits, „Szimulátor,” in *Járó robot készítése*, 2018, pp. 14-15..

[20] M. Babits, „Áttérés fizikai robotra,” in *Járó robot készítése*, 2018, p. 15..

[21] Sparkfun, „Servos Explaned,” [Online]. Available: https://www.sparkfun.com/servos#types-of-servos. [Hozzáférés dátuma: 28 05 2022].

[22] ROS, „Base local planner,” 04 04 2019. [Online]. Available: http://wiki.ros.org/base_local_planner. [Hozzáférés dátuma: 28 05 2022].

[23] ROS, „geometry_msgs/Pose Message,” 02 03 2022. [Online]. Available: http://docs.ros.org/en/api/geometry_msgs/html/msg/Pose.html. [Hozzáférés dátuma: 25 05 2022].