Széchenyi István Egyetem

Gépészmérnöki, Informatikai és Villamosmérnöki Kar

Informatika Tanszék

SZAKDOLGOZAT

Varga Zoltán

Mérnökinformatikus BSc szak

2021

SZAKDOLGOZAT

Kis méretű robotjármű mechatronikai kialakítása és szoftveres fejlesztése

Varga Zoltán

Mérnökinformatikus BSc szak

2021

Nyilatkozat

Alulírott, Varga Zoltán (GIRJ8O), Mérnökinformatikus, BSc szakos hallgató kijelentem, hogy a Kis méretű robotjármű mechatronikai kialakítása és szoftveres fejlesztése című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Tatabánya, 2021. december 20.

hallgató

Kivonat

Kis méretű robotjármű mechatronikai kialakítása és szoftveres fejlesztése

A projekt célja egy kis méretű robotautó mechanikai és szoftveres kifejlesztése, amely a Széchenyi István Egyetem későbbi önvezető járműves hallgatói fejlesztések alapjaként szolgálhat. A könnyebb javíthatóság és fejleszthetőség érdekében a fizikai kialakítás tervezése során törekedtem arra, hogy a jármű túlnyomórészt Robotis Turtlebot elemekből épüljön fel, ahol ez nem lehetséges ott kereskedelmi forgalomban kapható alkatrészek kerültek beépítésre. A robot mechanikai szem pontból hét fő blokkra került felosztásra. A járműben a fő vezérlő modul, a LIDAR és a kormányzás egyes elemei nem a Robotis fejlesztő készlet részei. A robot a közúti autók egy elterjedt felépítését követi, azaz négy kerék amelyből kettő a hajtott kerék (hátsó tengely), kettő pedig a kormányzott (első tengely). Ez utóbbi az Ackermann kormányzás módszere szerint került kialakításra.

A robot működése a későbbi könnyebb szoftveres továbbfejlesztés érdekében ROS (Robot Operating System) alapú. A fő feladat egy olyan szabályzó szoftver létrehozása volt, amely a bemeneti adatként kapott sebesség és irány adatoknak megfelelően működteti a hajtott kerekeket, valamint a kormányzott kerekek motorját. Ehhez szűkség volt az Ackermann bicikli model szoftveres megvalósítására.

A jármű mozgatásán kívül szükséges ismerni annak pozícióját, valamint orientációját két dimenzióban. Így a robot szoftvere ezen információkat folyamatosan közli egy az ROS-en belüli un. topic-ba meghatározott struktúrájú, standard üzenetben. Ezen információk elengedhetetlenek egy trajektória követési szabályzó algoritmus megfelelő működéséhez.

Az önvezető járműveknél fontos a környezet ismerete, például az objektumok pozíciója, mérete. A jármű tartalmaz egy LIDAR szenzort amely adatokat szolgáltat a robot közelében található lehetséges akadályokról. A LIDAR egy folyamatosan változó ponthalmazt generál amely az ROS-en belül szintén egy topic-ba kerül üzenet formájában.

Abstract

Mechatronic design and software development of a small robotic vehicle

The aim of the project is to develop a small-sized robot vehicle. The focus would be on both the software and mechanical side, in order to be suitable as a platform for future student developments at the Széchenyi István University. The vehicle consists of standard Robotis Turtlebot parts mostly because of its simpler repairability and development. In circumstances where this not possible, commercially available components have been installed would have been installed. The robot has been divided into seven blocks from a mechanical point of view. The control block, the LIDAR and some elements of the steering mechanism are not parts of the Robotis development kit. The robot follows a widespread structure of road cars, i.e. four wheels, where two of them are the driven wheel (rear axle) and two are the steered (front axle). The latter is designed according to the method of Ackermann governance.

The operation of the robot is based on ROS (Robot Operating System) for the purposes of later as well as easier software development. The main task is to create a control software that drives the driven wheels as well as the steering wheel motor. This is in accordance with the speed and direction data that is received as input data. This required for the software implementation of the Ackermann bicycle model.

In addition to moving the vehicle, it is necessary to know its position as well as its orientation in two dimensions. Thus, the robot's software continuously communicates this information ~~to~~ in a so-called standard message with a defined structure. This information~~s~~ is essential for the proper operation of a trajectory tracking control algorithm.

In the case of self-driving vehicles, it is important to know the environment, such as the position and size of the objects. The vehicle is equipped with a LIDAR sensor that provides information about possible obstacles in the robot’s surrounding. LIDAR generates an ever-changing set of points that are also placed in a topic within the ROS as a message.

Tartalomjegyzék

Bevezetés

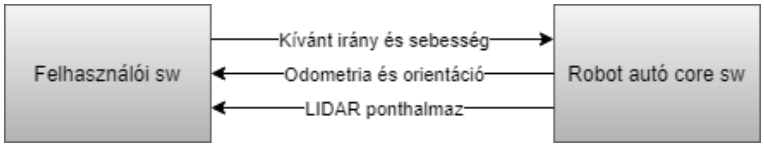
A projekt célja egy kisméretű robot építése, amely alapvetően egyezik egy széles körben elterjedt, négykerekű közúti jármű felépítésével. Azaz négy kerék két tengelyen, melyből legalább az egyik tengely hajtott, valamint egy tengely kormányzott.

A könnyebb reprodukálhatóság és egyszerűbb, gyorsabb hibajavítás érdekében a minimálisra kell csökkenteni az egyedi tervezésű alkotóelemek számát. Ezért a robotjármű elsősorban a Robotis vállalat TurtleBot3 Burger fejlesztő készletében megtalálható elemekből épül fel. Azok a részegységek, amelyek nem építhetők meg ezen készlet elemeiből, elsősorban szabványos, a kiskereskedelemben is kapható alkatrészekből állnak. Egyes esetekben, az alkotóelemeken utólagos megmunkálást kellet végezni, hogy a robotba illeszthető legyen.

Első körben szerettem volna megtervezni számítógéppel a jármű modelljét. A CAD tervezés egy projektben sok későbbi kellemetlenségtől (pl. alkatrészek illeszkedési hibája) mentheti meg a szakembereket. Amennyiben a kellő infrastruktúra és egyéb feltételek – úgy, mint megfelelő számítógép, megfelelő CAD fejlesztő program, nagy mértékű jártasság annak használatában – rendelkezésre állnak, az előzetes számítógépes tervezéssel időt is lehet megtakarítani. Ez főleg nagyobb méretű projektek esetében jelentős, ahol több fejlesztő dolgozik egy-egy bonyolultabb eszköz, berendezés kifejlesztésén. Eddigi műszaki, azon belül is R&D területen szerzett tapasztalataimra hagyatkozva, valamint a rendelkezésemre álló idő ismeretében úgy döntöttem, hogy a CAD tervezés helyett elsőként a specifikációk szerint egy prototípus építésébe kezdek. Korábbi projektmunkáim során kifizetődőbbnek bizonyult, ha kisebb méretű, kevésbé összetett eszközök esetében először egy mintadarabot készítettünk. Azon végezhettünk finomításokat, gyorsan és hatékonyan el tudtuk végezni a kivitelezés során felmerülő változtatásokat. Amennyiben ez az első verzió maradéktalanul megfelelt az elvárásoknak, úgy az alapján elkészülhet egy CAD terv, amely szerint már könnyen reprodukálhatóvá válik az adott eszköz. A későbbi kisebb fejlesztések és átalakítások már ezen számítógépes modellen is elvégezhetővé váltak függetlenül a gyártásban már futó verziótól. A rendelkezésemre álló idő alatt a meglévő CAD-es ismereteim, valamint alkalmas eszköz és szoftver rendelkezésre állás alapján úgy

gondoltam, gyorsabban haladhatnék a fent említett módszer használatával. Így ezen hiányosságok pótlása helyett több időt szánhatok a projekt lényeges részleteinek kidolgozására, úgy mint megállapítani azokat a pontokat, amelyek a szűk keresztmetszetet jelenthetik a projekt sikerességében, valamint melyek azok a szoftveres és szerkezeti megoldások amelyek a leginkább kielégítik a robottal szemben támasztott követelményeket. Azonban a CAD modell alkalmas a munkadarab vagy eszköz – ez esetben a robot jármű – azon fizikai paramétereinek mérésére, amelyek az elkészült munkán nem vagy csak nagyon nehezen, estleg pontatlanul állapíthatóak meg mérőeszközök segítségével. Egy ilyen paraméter a kormányzott kerekek kitérési szöge a jármű hossztengelyéhez képest, amely ismerete fontos az Ackermann kormányzás számításánál. Továbbá fontos ismerni a LIDAR pozícióját a talajhoz és a jármű dimenzióihoz képest egy olyan trajektória tervezési feladat kivitelezésénél, ahol a robot egymáshoz közeli objektumok közt mozoghat.

A robot, várhatóan az önvezető járművek témakörében végzett hallgatói projektek alapjául fog szolgálni, így ennek a célnak legmegfelelőbb szoftver struktúrával kell rendelkeznie. Az autonóm járművek fő feladata, egy adott útvonalon történő haladás a kívánt paraméterek szerint, figyelembe véve a környezeti sajátosságokat. Az ezt megvalósító algoritmusoknak szűksége van egy összekötő törzs szoftverre, amely az autó meghajtó és kormányzó motorjait a megfelelő módon képes mozgatni. Ugyanennek a szoftvernek kell kezelnie és megfelelően továbbítania a robot érzékelői által mért adatokat, amelyek elengedhetetlenek a trajektória követéshez és objektum detektáláshoz. Ez tulajdonképpen egy interface-ként szolgál a felhasználói vezérlő szoftver és a jármű között.



1. ábra A jármű törzs szoftverének adatfolyamai felhasználói oldalról

A jármű mozgása alapvetően az Ackermann bicikli modell szoftveres megvalósításán alapul. Így a későbbiekben implementált önvezető algoritmusnak elegendő a kívánt sebesség és szögsebesség (elfordulás) értékeket meghatároznia a robot kívánt útvonalon történő mozgatásához. Az autó vezérlése a robotikában széles körben használt ROS rendszer használatával került implementálásra. Ezáltal egy olyan platform áll rendelkezésre a későbbi fejlesztésekhez amely jól definiált, megfelelően strukturált és átlátható.

1 A jármű felépítése

A járművet a mechanikai fejlesztés szempontjából hét fő részegységre bontottam fel. Ezek az alábbiak:

- a jármű váza
- elektromos tápellátás
- hátsó tengely
- első tengely
- vezérlő modul
- SBC (Single Board Computer) modul
- LIDAR (Light Detection and Ranging)

A robot felépítése egy – a közúti személyautók esetében elterjedt – négy kerekű jármű struktúráját követi. Az első két kerék a kormányzott, a hátsó tengely a hajtott. A kormányzást egy, míg a meghajtást kerekenként egy-egy, egymástól független szervo motor látja el.

2. ábra A prototípus végső kialakítása

A robot az eredeti TurtleBot burger verziójához hasonlóan szintekre bontható. Az alsó szinten kerültek elhelyezésre a meghajtó és irányító motorok, a kormányszerkezet és az akkumulátorok. A középső szintre helyeztem az SBC modult és a vezérlő panelt, míg a jármű legfelső pontjára szereltem fel a LIDAR szenzort.

1.1 A jármű váza

A vázszerkezetnek több szempontnak kell megfelelnie. A legfőbb követelmények az alábbiak:

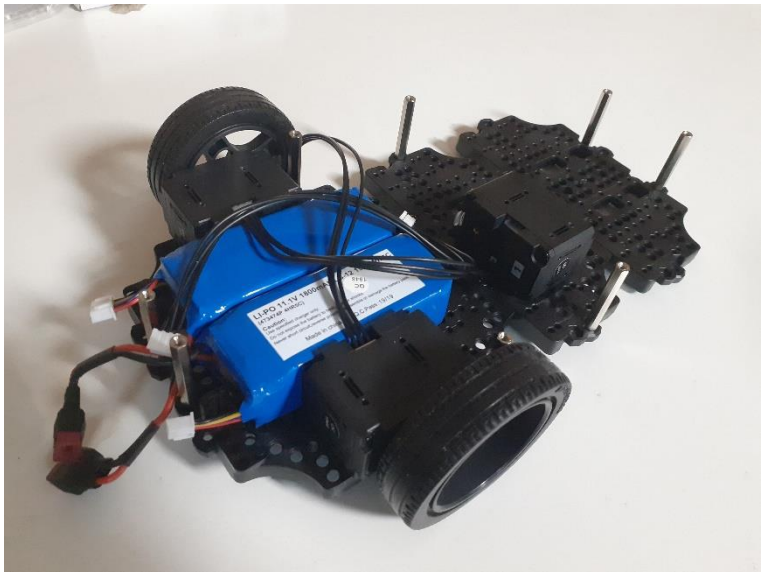
- szerkezeti szempontból elegendő merevséget kell biztosítania a megfelelő, stabil és pontos működéshez, mivel egy nem kellően alaktartó váz hatással lehet a jármű iránytartására.
- minden egyéb részegységnek szilárdan rögzíthetőnek kell lennie.
- könnyen szerelhetőnek kell lennie egy esetleges hibaelhárítás és javítás során.

Természetesen itt is figyelembe kell venni azt a szempontot, hogy amennyiben lehet, elsősorban a TurtleBot elemeit, vagy kereskedelmi forgalomban kapható alkatrészeket előnyös felhasználni. A fejlesztő készlet elemei, azon belül is a műanyag lemezek, amelyek a TutleBot „szintjeit” alkotják, valamint a hatszögletű fém távtartók amelyek a szinteket választják el egymástól, elég jól variálhatónak bizonyultak. Az első prototípus építésekor sikerült elérni, hogy a váz csak a készlet elemeiből álljon (műanyag lapok, hatszögletű távtartók 35 és 45mm hosszúak, M3x8-as kereszthornyú csavarok, valamint M3-as csavaranyák. A váz három szintes. Az alsó és középső szintet a 35mm-es távtartók kötik össze, így a motorok – méreteikből adódóan - pontosan bele illenek a két műanyag lap közé. A középső és felső szintet a 45mm-es távtartók kötik össze, így elegendő hely marad a vezérlő modul bekötéseinek. A lapok a készletben található M3x8-as csavarokkal és csavaranyákkal vannak egymáshoz rögzítve.

Az OpenCR vezérlő modul, az SBC modul valamint a LIDAR felszerelése a Turtlebot készletben található, elektronikai panelek rögzítésére szolgáló elemekkel lettek megvalósítva. Ezek kis mértékű szabadságot adnak a panelek elhelyezésénél, amely előnyös az oldalsó csatlakozók hozzáférhetősége szempontjából. A motorok rögzítéséhez a műanyag építőelemek megfelelő méretű furatait használtam fel. A kormányszerkezet tartó és beállító elemei szintén ezen műanyag lapok furataihoz lettek csavarozva. Utóbbi esetben két 2,5 mm-es furatot fel kellett tágítani 3 mm-esre. A LIDAR illesztő moduljának sajnos nem találtam megfelelő rögzítési pontot, így azt a LIDAR fejet tartó lap aljához, műanyag kábelkötegelőkkel erősítettem fel.

1.2 Elektromos tápellátás

A TurtleBot csomagban egy 11,1 V-os, 1800 mAh-s kapacitású akkumulátor található. Mivel a fejlesztéshez számomra két készlet is rendelkezésre állt, a prototípusba mindkét akkumulátort beépítettem. Ez egy kisebb súlygyarapodást eredményezett azonban így a jármű rendelkezik egy tartalék akkumulátorral is. A redundáns tápellátás abban az esetben nyújthat előnyt, ha egy hosszabb teszt közben lemerül az egyik akkumulátor. Ebben az esetben nem kell a tesztet félbe szakítani míg az akkumulátor feltöltésre kerül, valamint nem kell szétszerelni egy akkumulátor cseréhez, csak a tápellátást biztosító vezetéket kell a másik, tartalék akkumulátorra csatlakoztatni. A két akkumulátor a jármű alsó szintjén foglal helyet egymás mellett középen, amelynek köszönhetően a súlypont is elég alacsonyra került. Ez jótékony hatással van a jármű stabilitására.



3. ábra Akkumulátorok elhelyezése a járműben

A TurtleBot alapvetően egy RaspberryPi 3B+ modult tartalmaz, amely az OpenCR1.0 modulon keresztül az USB csatlakozón kap tápellátást. A Raspberry ezen változata 2A-es betáplálást igényel. Ez az USB2.0 maximális áramerősségével egyezik meg. Azonban én a Raspberry-t lecseréltem egy Nvidia Jetson NANO-ra amely a specifikáció szerint 3A-es betáplálást igényel 5 V feszültség mellett. Nagyobb számítási kapacitás esetében így az USB porton keresztül történő elektromos tápellátás már kevésnek bizonyulna. A JetsonNANO elektromos ellátása két féle módon történhet: micro usb csatlakozón vagy koaxiális jack

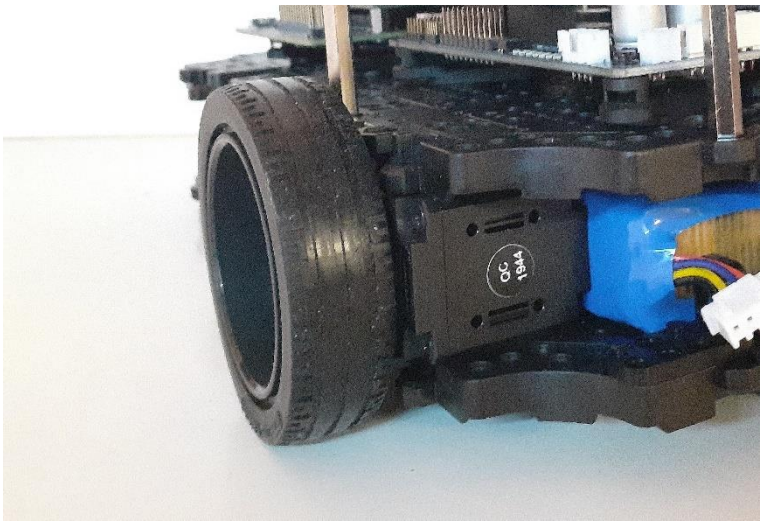
aljzaton keresztül. Mivel az usb porton keresztül nem tud 2 A-nél nagyobb áramot felvenni a modul, elengedhetetlen a jack aljzaton keresztüli megtáplálás. Ehhez azonban készítenem kellett egy egyedi kábelt, amely az OpenCR panel 5 V-os 4 A-es Molex Mini-spox csatlakozóval rendelkező kimentét összeköti a JetsonNANO jack aljzatával. Ahhoz, hogy a

4. ábra A tesztelt akkumulátorok

SBC ezen tápcsatlakozóját használja, szükséges a J48-as tüske párt jumper-elni. Ekkor már nem device üzemmódban működik a modul és az üresjárati áramfelvételei is magasabb (körülbelül 2 A). A turtlebot akkumulátorának kapacitása elegendőnek bizonyult a projekt fejlesztés során, ugyanis a lemerülési idő meghaladta a töltési idejét, így amikor az éppen használatban lévő akku már merült, a másik már feltöltött állapotban volt. Rendelkezésemre állt egy nagyobb kapacitású, 5000 mAh-s akkumulátorhoz. Azonban ez csak két cellás ami azt jelenti, hogy a kapocs feszültsége alacsonyabb, mégpedig 7,4 V. Az OpenCR modul specifikációja szerint annak elektromos betáplálása az akkumulátoros csatlakozóján 5-24 V. Ennek megfelelően ez az akkumulátor is megfelelő a jármű működtetéséhez amelyet teszteltem is. A nagyobb kapacitás ellenére, az alacsonyabb feszültség miatt nincs szignifikáns különbség az utóbbi és a két Turtlebot akkumulátor együttes rendelkezésre állási ideje közt. Mégis egy hosszabb teszt esetén a nagyobb kapacitásút ajánlom a későbbi használatra, mivel nem kell akkupack-ot váltani idő közben. Ennek használatához azonban módosítani kell az akkumulátor feszültség figyelés értékét az OprnCR modul szoftverében, ugyanis az a 11,1V.os feszültség szinthez van igazítva.

1.3 Hátsó tengely

Az egyik legegyszerűbb felépítésű egység a hátsó tengely a robot fő részei közül. Ez a jármű hajtott tengelye. Mechanikai szempontból egyszerű felépítésű, mivel nincs fizikai összeköttetés a két kerék között. Sem fix tengely, sem differenciálmű. Az egység négy elemből áll: két Dynamixel meghajtó motor, valamint két kerék és az azokat a motor tengelyére rögzítő kerekekként négy-négy csavar. Ez az úgynevezett agymotoros megoldás megtalálható a modernebb elektromos közúti autókban is (pl. Ford Fiesta eWheelDrive). A motorok, a kerekek és a csavarok egyaránt a TurtleBot fejlesztő készlet elemei. A motorok a vázat alkotó műanyag lapok ugyanazon rögzítő pontjain kerültek elhelyezésre, amelyeken a TurtleBot Burger verzióján is. Ennek megfelelően a nyomtáv szélesség is 160 mm, ami megegyezik a Turtlebot-éval.



5. ábra A hátsó kerék és a hajtó motor

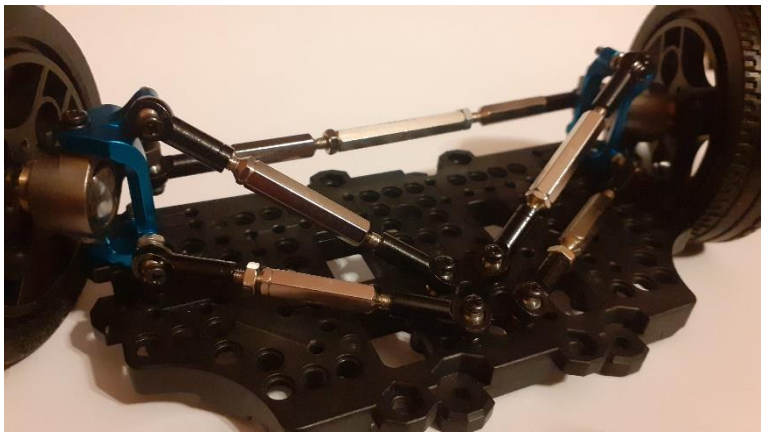
Előnye, hogy nem kell bonyolult, kis méretű differenciálművet tervezni és beépíteni. Kevés alkatrészből áll így kisebb a meghibásodás esélye is, valamint nem tartalmaz egyedi gyártású elemet. Hátránya hogy nem egyenes vonalon történő haladás során, a két keréken fellépő fordulatszám különbséget szoftveresen kell megoldani.

A motorok Dynamixel XL430-as típusú szervo motorok. A motorba épített lassító hajtómű áttétele rendkívül nagy, 258,5:1. Ennek köszönhetően méretéhez képest elég nagy - 1,4 Nm-es - indulási nyomatékot képes leadni. Azonban ez nagyon kis (57 r/min terheletlenül) fordulatszámot is eredményez, ami a TurtleBot 66mm-es átmérőjű kerekeivel alig több mint

0,5 km/h-s sebességet fog eredményezni. Ez a fordulatszám a három cellás, 11,1 V-os akkumulátor esetén értendő. Kisebb feszültségű akkumulátor használata esetén ez a sebesség csökken. Amennyiben ez kevésnek bizonyul, úgy szükséges lehet a motorok cseréje, vagy egy mechanikus áttétel tervezése és beépítése.

1.4 Első tengely

Az első a jármű kormányozható tengelye. Itt a két kerék szabadonfutó, így nincsenek hajtó motorok. Ahhoz, hogy a jármű pontosan a kívánt útvonalon haladhasson fontos, hogy a kormányzott kerekeket össze tartó szerkezet geometriája minél pontosabb és alaktartó legyen. Amennyiben a merevsége nem megfelelő, ennek következtében pedig a kerekek járműhöz és egymáshoz viszonyított helyzete változik, a jármű útvonal követése pontatlanná válhat. Ezért szűkség volt egy olyan szerkezet beépítésére amely a kívánt követelményeknek megfelel. Sajnos a Turtlebot készlet nem tartalmaz olyan elemeket amelyek felhasználásával kivitelezhető egy erre a célra megfelelő futómű építése. Mivel az egyedi alkatrészek felhasználását kívántam elkerülni, olyan megoldást kerestem amely nem tartalmaz ilyen elemeket.



6. ábra Az első futómű kialakítása

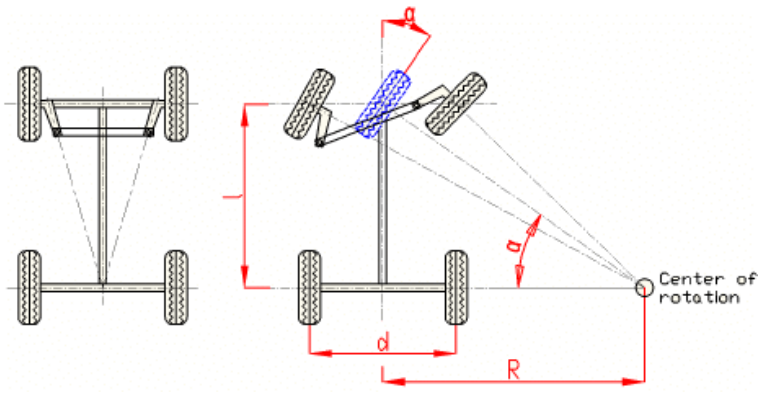
A távirányítású autók világszerte nagy népszerűségnek örvendenek nem csak a gyerekek, hanem a felnőttek körében is, így nem csoda hogy kereskedelmi forgalomban kapható már különböző méretarányú, formájú és meghajtású autó. Ezen modell autók többsége a méretükhöz képest elég nagy sebesség elérésére képesek. A használatból eredő sérülések, a

kopás és a fejlesztés iránti igény azt eredményezte, hogy kereskedelmi forgalomban kapható számos alkatrész ezen hobbi járművekhez.

Az autók mérete kissé eltérő, ennek ellenére egységes és kategorizált amely a valós méretű személyautókhoz viszonyított. Ennek megfelelően a megvásárolható pótalkatrészek is részben egységesített paraméterekkel rendelkeznek méret kategóriákként.

Így a projekt robot járműve esetében is egy megfelelő választás volt ilyen elemekkel kialakítani az első futóművet és kormány szerkezetet. A piacon kapható alkatrészek közül az 1:18-as méretarányú modell autókhoz gyártott alkatrészek bizonyultak megfelelő méretűnek. A kormányzó szervo motor, a kerekek tengelye és a kerekek kivételével, az első futómű teljes egészében ilyen alkatrészekből épül fel. A kiválasztott elemek stabilan illeszkednek egymáshoz valamint a jármű vázához.

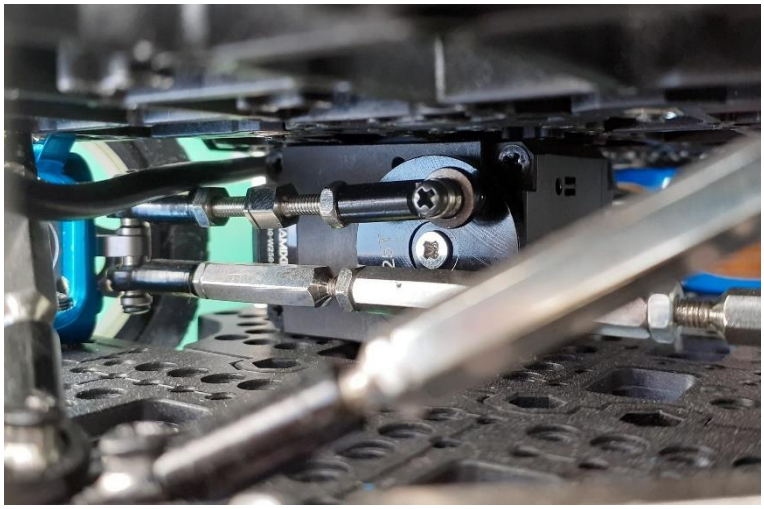
A kerekek két dimenzióban konfigurálhatók amely elegendő ahhoz, hogy a jármű iránytartása stabil legyen. Kerekenként egy tartó rúddal állítható be annak dőlése, egy másik rúddal pedig a kerék össze tartás.



7. ábra Ackermann kormányzás

A jármű fordulásakor a fordulókör külső és belső oldalán lévő kerekek eltérő körív mentén haladnak. Mivel a fordulókör középpontját a kerekek tengelyével összekötő szakaszok nem esnek egy egyenesre, a kerekek ideális szögelfordulása – amellyel azok a megfelelő köríven képesek haladni – a jármű hossztengelyéhez képest egymástól eltérők lesznek. A két kerék szögelfordulás különbsége a fordulókör sugarának csökkenésével nő. Ha a kerekek azonos szögben térnek ki fordulóban, úgy a két kerék közül az egyik (amelyiknél a tapadás kisebb)

megcsúszhat oldal irányban, amely eredményeként a jármű forduló íve meg fog egyezni a még tapadó kerék szöge által meghatározott ívvel.



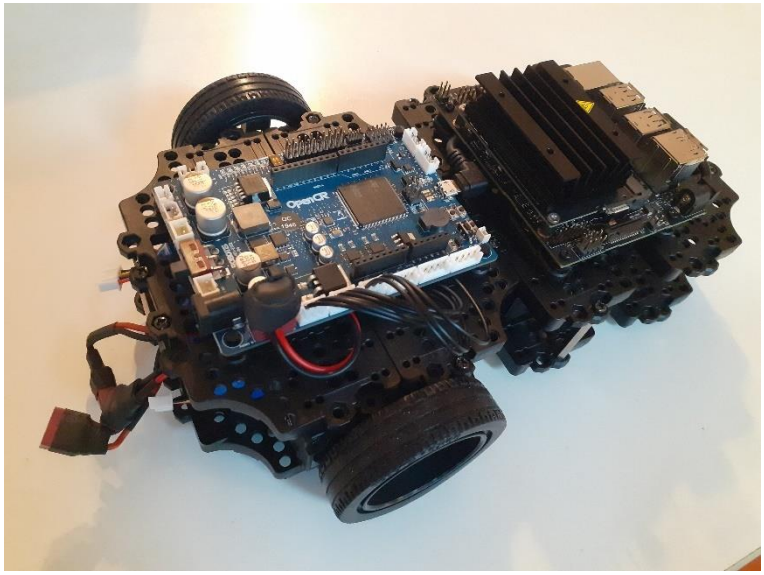
9. ábra A kormány összekötők kialakítása

Ennek kiküszöbölésére jó megoldás az Ackermann kormányzás, amely közel az ideális szögeltérést eredményezi mind a külső, mind pedig a belső kerekek esetében. Könnyen kivitelezhető, széles körben használt módszer valós méretű járműveknél. Ideális Ackermann geometria kialakítás esetében, a kormányzott kerekek forgás pontját és a kormány összekötő rúd csuklópontját összekötő egyenesek a másik tengely felezőpontjában metszik egymást. Ezen projekt esetében a kerékagyak nem a jármű méreteihez tervezett alkatrészek, így ez a metszéspont kissé a hátsó tengely előtt van. Azonban a kerekek méretei és azok váztól mért távolsága miatt a kerekek nem téríthetőek ki nagy mértékben, ennek megfelelően a jármű nem képes olyan kis sugarú íven fordulni amely esetén az ideálistól eltérő geometria számottevően hatással lenne a trajektória követés pontosságára.[2]

A két kerék közti kapcsolatot egy hosszabb, míg a kormányzó motor és a kerekek között egy rövidebb kormányrúd valósítja meg. Utóbbit a motorhoz nem lehetett volna holtjáték mentesen rögzíteni, ami az iránytartás pontosságát rontotta volna. A motoron lévő eredeti műanyag tárcsát eltávolítva a motor tengelyére egy másik fém tárcsa került felszerelésre, amelyhez már stabilan lehetett rögzíteni a kormányrudat.

1.5 Vezérlő modul

A jármű vezérlő modulja a TurtleBot készletben megtalálható OpenCR1.0 panel. Ezt a modult, kimondottan ROS támogatott beágyazott rendszerek építéséhez fejlesztette ki a Robotis vállalat. A hardver és a szoftver egyaránt nyílt forráskódú. A panel több port-tal rendelkezik. Az elektromos tápellátása történhet usb csatlakozón keresztül 5 V-os feszültséggel, vagy egy másik szélesebb feszültségtartományt megengedő csatlakozón keresztül kapcsoló üzemű tápegységgel vagy akkumulátorral. Ez esetben a Turtlebot saját 11,1 V-os lítium polimer akkumulátorával történik az energiaellátás. A panelen található egy 3,3 V-os, egy 5 V-os és egy 12 V-os kimeneti csatlakozó is amelyről megoldható más egységek ellátása. Az autón ezek közül az 5 V-os kimenet van használatban, amelyről az SBC kapja a működéshez szükséges elektromos ellátást.[3]



10. ábra Az OpenCR vezérlő modul beépítése a robot járműbe

A modul rendelkezik öt kommunikációs port-tal: egy micro usb, három TTL és három RS485 csatlakozóval Dynamixel motorokhoz, két darab négy tüskés UART soros port, valamint egy CAN bus port. Ezek közül a micro usb csatlakozó került felhasználásra amelyen keresztül az SBC-vel történő kommunikáció valósul meg, valamint a három TTL motor csatlakozóra a két hajtott kerék motorja, valamint a kormányzást végző szervo motor vezetéke lett illesztve.[3]

Ebből egy külön csatlakozó van az elektromos betáplálás számára közvetlenül a saját akkumulátorról. A robotjárművön ez a csatlakozó kerül használatra. Ezen kívül egyéb csatlakozók is nyújtanak külső áramforrás csatlakoztatási lehetőséget. Ezen felül több digitális ki-és bemeneti port-tal, analóg bemenettel, valamint arduino csatlakoztatásához fenntartott érintkezőkkel van felszerelve az OpenCR. Ugyan ezek nem kerültek felhasználásra, későbbi fejlesztések során (plusz szenzor beépítése) hasznosíthatóak.[3]

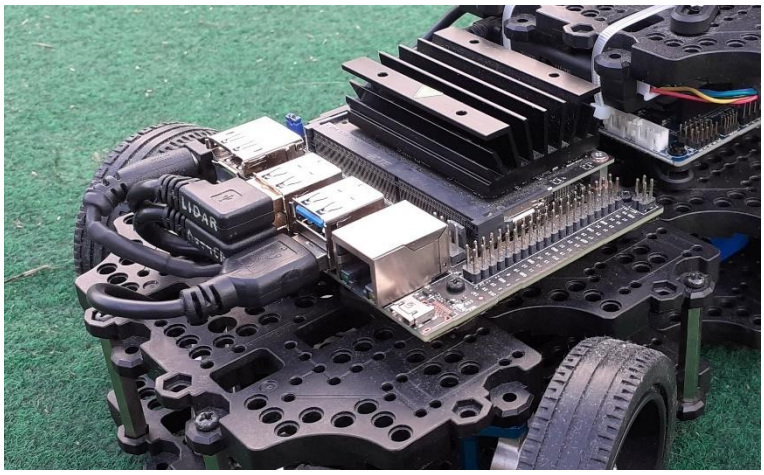
A modul több beépített szenzort is tartalmaz: 3 tengelyű giroszkóp, 3 tengelyű gyorsulásmérő és 3 tengelyű iránytű. Ezek közül a giroszkóp és a magnetométer jól használható a robot orientációjának meghatározására a függőleges tengely körüli elfordulásának figyelésével. Továbbá található még a modulon teszt és reset gomb, vissza jelző ledek és egy buzzer is.[3]

1.6 SBC modul

Az OpenCR1.0 vezérlő modul nem rendelkezik operációs rendszerrel, így főként interface szerepet lát el a felhasználói szoftver és motorok valamint a szenzorok közt. A fejlesztés és a valós idejű ellenőrzés nehézkesebb lenne kizárólag az OpenCR használatával, ezért előnyös beépíteni egy egykártyás számítógépet – elterjedtebb angol nevén: **S**ingle **B**oard **C**omputer – az OpenCR kártya mellé, amely ellátja a jármű fő szabályzó modul szerepét. Az ilyen paneleket főként fejlesztési célokra tervezik, architektúrájuk részben hasonló a személyi számítógépekéhez: központi processzor (többnyire ARM), beépített operatív memória, I/O portok, illeszthető háttértár. A legtöbb típusnál lehetséges valamilyen kijelző, billentyűzet és egér csatlakoztatása, amely nagy mértékben megkönnyíti a fejlesztői munkát. Ezen felül több (általában IEEE) szabványos kommunikációs port-tal is rendelkeznek, amikhez köszönhetően számos különböző periféria is illeszthető. Alapvetően a TurtleBot3 burger kit egy RaspberryPI 3B+ modult tartalmaz, amely jelenleg az egyik legnépszerűbb SBC a hobby fejlesztők körében. Az 1,4 GHz-es, 64 bites, 4 magos (ARM Cortex-A53) processzorral, 1 GB DDR2 RAM memóriával úgy vélem, hogy a számítási kapacitása a projekthez elegendő lenne. Azonban lehetőségem nyílt az NVIDIA cég Jetson Nano számítógépét használni a Raspberry helyett. Processzora szintén ARM architektúrájú és közel hasonló paraméterekkel bír (ARM Cortex-A57), azonban a beépített operatív memóriája lényegesen nagyobb és gyorsabb, 4 GB DDR4-es típus. A Jetson Nano

kimondottan mesterséges intelligencia megoldásokhoz lett optimalizálva: többszintű mesterséges neurális hálók futtatása, gépi látás algoritmusok használata, úgymint tárgyak azonosítása, objektum detektálás. A jármű későbbi felhasználását ezen tulajdonságok nagy mértékben megkönnyíthetik. Sem a Raspberry, sem pedig a Jetson Nano nem rendelkezik héttértárként működő beépített memóriával. Ezért egy microSD kártyát kell az erre a célra kialakított slot-ba helyezni.[4][5]

Az SBC nem közvetlenül kap tápellátást az akkumulátorról, hanem egy kábelen keresztül az OpenCR panel 5 V-os kimenetéről. Ezen felül a négy usb csatlakozó közül három került

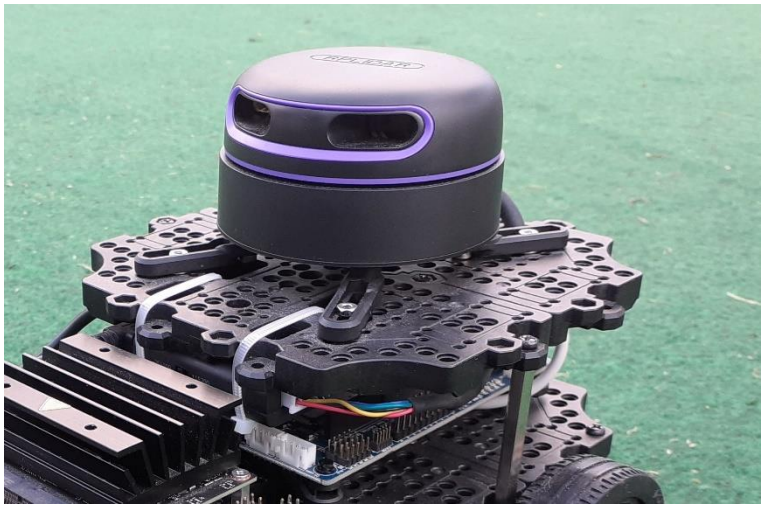


11. ábra A Jetson elhelyezése és bekötése a járművön

felhasználásra. Egy, a Turtlebot készletből felhasznált usb kábel köti össze a Jetson-t az OpenCR panellel, amely a két modul közti kommunikációt biztosítja. Egy további – az előzővel azonos típusú – kábel köti össze az SBC-t a LIDAR interface moduljával, amely szintén a két egység közti adatforgalom biztosításáért felel. A harmadik csatlakozót, egy a LIDAR saját kábele foglalja, amelyen keresztül az interface modul és a LIDAR fej kap tápellátást. A negyedik usb csatlakozó a fejlesztés során volt használatban. Mivel a Jetson-nak csak vezetékes ethernet hálózati csatlakozója van, így egy wlan modul segítségével elérhető volt headless módban is és nem kellett leválasztani az OpenCR-t vagy a LIDART, hogy a billentyűzet és az egér is csatlakoztatható legyen. Az SBC elhelyezésénél elsődleges szempont volt az, hogy a csatlakozók könnyen hozzáférhetőek legyenek, hogy tesztelés vagy hibakeresés során ne kelljen megbontani a jármű szerkezetét. A Jetson Nano hűtőbordájára szerelhető hűtőventilátor, amennyiben annak túlzott melegedése azt indokolja.[4][5]

1.7 LIDAR

Egy önvezető járműnek elengedhetetlen, hogy képes legyen információt gyűjteni a fizikai környezetéről. Ehhez számos szenzor használható különböző detektálási technológiával, eltérő méretekben és különféle kialakítással. A LIDAR szenzor lézer technológiával képes a környezetét feltérképezni. A szenzor fő eleme egy folyamatosan forgó fej amely tartalmaz egy lézer fényt kibocsájtót, valamint egy szenzort amely a visszaverődött lézerfényt detektálja. A LIDAR a lézer fény kibocsájtása és annak visszaverődés utáni detektálása közt eltelt időt méri. Mivel a fényenergia terjedési ideje ismert, a visszaverődési idő alapján kiszámítható annak a felületnek a szenzortól mért távolsága, amelyről a lézerfény visszaverődött. A LIDAR ezen távolság adat mellé szolgáltatja azt a szögelfordulás értéket, amelyben a fej a meghatározott 0°-hoz képest állt a detektálás pillanatában. Így a távolság és szögelfordulás ismeretében meghatározható a visszaverődési pont helyzete a szenzor fejhez képest.[6]



12. ábra Az RPLidar A3 elhelyezése a járművön

LIDAR szenzorokból elérhetők egy és több csatornás típusok. Az egy csatornás szenzorok esetében egy időpillanatban csak egy lézerimpulzus kerül kibocsájtásra, míg a többcsatornás eszközöknél egyszerre több. A legjellemzőbb típusok az 1, 16, 32, 64 és 128 csatornás szenzorok. Ezen kívül választhatunk több féle felbontású érzékelők közül, ahol a felbontás a szögelfordulást jelenti két mintavételezés között (többnyire fokban megadva). Tapasztalataim szerint egyes felületek nagy mértékben rontják a LIDAR hatékonyságát.

Ilyen például a tükör, ahol a mért távolság nem a szenzor és a tükör felület közti szakasz hossza lesz, hanem annál nagyobb, mivel a tükörről még egyéb felületre is tovább halad a lézer nyaláb. Ezen kívül a fényt nagy mértékben elnyelő sötét, matt felületek okozhatnak hibás méréseket, amelyekről nem érkezik vissza elegendő intenzitású fény, így azt a LIDAR nem képes detektálni.

A TurtleBot készlet tartalmaz egy LDS-01-es típusú szenzort. Ennek a maximális érzékelési távolsága ~3,5m, felbontása 1°. Azonban lehetőségem nyílt a robotjárművet egy Slamtec Rplidar A3-as szenzorral felszerelni, amely beltéri használat esetén, ideális körülmények közt 25m-es hatótávolsággal bír és 0,3375°-os felbontással. Úgy vélem a 3,5m-es érzékelési távolság elegendő lenne, azonban az 1°-os felbontás mellett a robot számára észrevehetetlen maradhat olyan akadály, amely csak 1-2mm vastag. Emellett a nagyobb felbontásnak köszönhetően az egyes tárgyak meghatározása hatékonyabbá válhat a pontfelhő szegmentálása során, mivel a sűrűbb pontok könnyebb éldetektálást tesznek lehetővé. Útvonaltervezés során a nagyobb érzékelési távolság is hatékonyabb tervezést eredményezhet.[7][8]

A LIDAR a vezérlő modul felett a második szintre került a prototípuson azért, hogy a jármű különböző részei ne generáljanak holttereket az érzékelés során. Azonban így nagyobb az esélye, hogy a jármű közelében lévő alacsonyabb akadályokat nem lesz képes detektálni. Egy másik előnye a LIDAR ezen elhelyezésének, hogy így annak függőleges forgástengelye merőleges a robot Ackermann bicikli modelljének hátsó kerék tengelyére. Ez a későbbiekben előnyös lehet mivel így a LIDAR által generált ponthalmaz középpontja egybe esik a jármű azon pontjával amely a kívánt pályavonalon halad.

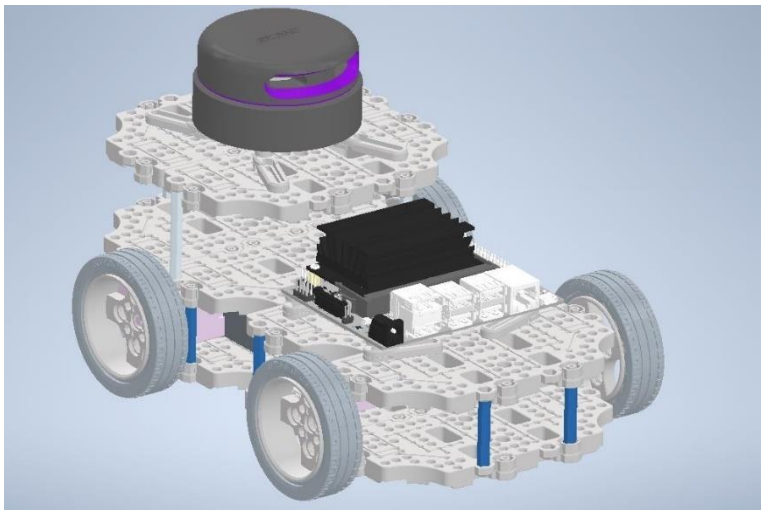
1.8 CAD tervezés

Egy nagyobb, összetett eszköz létrehozása során a folyamat első lépése a tervezés, amely a személyi számítógépek megjelenése előtt többnyire a tervező asztalon, papíron történt. A számítógépek megjelenésével és fejlődésével ez azonban lényegesen egyszerűbbé és gyorsabbá vált a különböző tervező szoftverek segítségével. A CAD (Computer Aided Design) technológia használatával az ipari tervezési folyamat sokkal hatékonyabbá vált. Segítségével még a projekt elején meghatározhatók mik lesznek azok a tényezők, amelyek a gyártás során technológiai nehézséget okozhatnak, milyen funkcionális hibák léphetnek fel a működés során. Ezen felül a CAD terv alkalmas az eszköz vagy berendezés azon paramétereinek, különböző dimenzióinak megadására amelyeket az elkészült mintán nehezen vagy egyáltalán nem lehet mérésekkel meghatározni. Ezen projekt esetében egy ilyen adat a kerekek tengelytávjának vagy nyomtávjának megadása.[9]

Sajnos technológiai nehézségek és a projektre fordítható idő nem tette lehetővé, hogy a fejlesztést a CAD tervezéssel kezdjem, továbbá úgy ítéltem meg, hogy a kész alkatrészek használata és a jármű várható komplexitása nem feltétlenül indokolja az előzetes digitális tervezést. Azonban az autó CAD terve a későbbi fejlesztések során már nagyon sok segítséget nyújthat, ezért a rendelkezésemre álló erőforrásoknak megfelelően létre hoztam a robot jármű egy olyan modelljét, amely egy jó alapot jelent annak tovább bővítésére és alkalmas egyes méretek meghatározására.

A szoftver piacon számos tervező alkalmazás áll rendelkezésre különböző tudományágakban. Egyaránt találhatunk építész, gépész, elektronikai vagy település tervező programokat. Ez a széles skála egyes szoftver gyártók esetében is tapasztalható. Az Autodesk vállalat termékei segítséget nyújtanak például az építészet, média és gyártmány tervezés területén. Alapvetően az általános felhasználású AutoCAD segítségével szerettem volna elkészíteni az autó modelljét, azonban konzulensem tanácsára a főleg gépészeti tervezésre optimalizált Inventor alkalmazást választottam. Ugyan az Autodesk tervező szoftvereinek ára tükrözi azok kiválóságát és az azok kifejlesztésére fordított fejlesztést, egy magánember számára mégis eléggé magas. Azonban a vállalat lehetőséget nyújt egyes szoftverek esetében, azok ingyenes használatára egyetemi hallgatók esetében.[9]

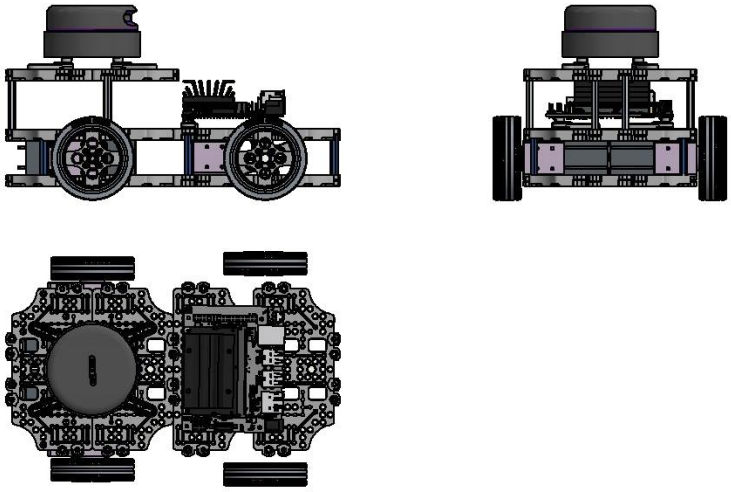
A modell egyes alkatrészei nagy mértékben összetett elemekből állnak. Ilyenek például a Turtlebot „waffle” lapjai amelyek sok furatot és lekerekítést tartalmaznak. Mivel a jármű váza túlnyomó részt ezen elemekből áll, minden egyéb elem ezekre van rögzítve. Ezen felül a további fejlesztések során felszerelésre kerülő elemek is hasonló módon kerülnek rögzítésre, fontos, hogy a modell ezen alkatrészei méretpontosak és részletek legyenek. Mivel a Turtlebot széles körben használt fejlesztő eszköz, az egyes elemeinek step modellje megtalálható egyes CAD modelleket tartalmazó webes adatbázisokban.[10]



13. ábra A robot jármű CAD terve

Ennek köszönhetően nem kellett feleslegesen sok időt töltenem az összetett alkatrészek részletes megtervezésével. Ezen felül azok méretpontossága is megbízhatóbb. A modellben az elektromos és jelvezetékek nem kerültek megjelenítésre, mivel mechanikai szempontból nem lényeges azok elvezetése. Ezen felül a valós járművön azok megfelelő elvezetése nem okozott problémát, mivel a Turtlebot elemeken megfelelő mennyiségű rögzítési pont található. Az ilyen elemek gyakran .stp vagy .step kiterjesztésűek, amelyeket a legtöbb CAD fejlesztő alkalmazás képes kezelni. Azonban ezen felül még számos kiterjesztés van használatban, egyes esetekben csak adott szerkesztőhöz dedikáltan amelyet más környezet nem kezel. Az inventor a létrehozott három dimenziós modell fájlt .iam, míg az abból generált képeket – például három képsíkos vetületi rajzot, vagy méretezett ábrát – pedig .idw kiterjesztéssel látja el.

A CAD modell készítésekor a valós jármű paramétereit mérésekkel határoztam meg. Az egyes elemek ezen adatoknak megfelelően lettek egymáshoz illesztve. A kormányzás elemei különféle gyártók termékeiből kerültek kialakításra, amelyek modellje nem állt rendelkezésemre. Azok többsége összetett geometriai tulajdonságokkal és méretekkel rendelkezik, amelyek meghatározása és az alkatrészek azok alapján történő létrehozása túl sok időt vett volna igénybe, így a kormányszerkezetet nem tartalmazza a jármű modellje. Ennek ellenére az első kerekek pozíciója a valós elhelyezkedésüknek megfelelő. Amennyiben a jövőben a kormányszerkezet átalakításra vagy fejlesztésre kerül, úgy szűkségessé válhat annak elhelyezése a modellbe.



14. ábra A jármű modelljéből generált három képsíkos vetületi rajz

A szerkesztés során főként a *constrain* parancsot használtam az elemek egymáshoz illesztéséhez. Ezen funkció használatával megadható, hogy a kiválasztott alkatrészek mely sík felületeit helyezzük egy síkba. Ezen felül ezzel a paranccsal egy pontba mozgathatunk különböző elemeket az azokon meghatározott pontok alapján, valamint mozgathatók alkatrészek oly módon, hogy azok meghatározott furatai, tengelyei egytengelyűek legyenek. Például a kerekek motor tengelyre igazításához síkba és tengelybe egyaránt egyeztetni kellett a motort és a kereket.

2 Szoftveres háttér

A projekt fő célja egy önvezető robot jármű alapjainak létrehozása, amely megfelelő az egyetem hallgatói számára az autonóm járműves algoritmusok fejlesztéséhez és teszteléséhez. Ehhez arra van szükség, hogy a jármű egy olyan alap szoftverrel rendelkezzen amely vezérli az autó motorjait és szenzorjait a felhasználói szoftver által küldött utasítások szerint, valamint megfelelő információkat szolgáltat a felhasználó felé.

2.1 Jetson Nano

Az Nvidia Jetson Nano fejlesztő board computere nem rendelkezik beépített háttértárral. Erre a célra egy MicroSD kártya szolgál, amelyet a modul alsó részén lévő foglaltba kell helyezni. Az operációs rendszer telepítéséhez a gyártó egy image fájlt bocsájt rendelkezésre amely letölthető a *developer.nvidia.com* webhelyről. Először a Turtlebot készletben található Rapberry Pi-hez mellékelt 16 GB-os microSD kártyát használtam. Azonban ez túl kicsinek bizonyult. Az operációs rendszer és az ROS telepítése után 1 GB-nál kevesebb tárhely maradt szabadon. Így az Nvidia ajánlásának megfelelően kicseréltem egy 32 GB-os UHS-1 microSD kártyára amely már megfelelő méretűnek bizonyult. Ugyan ez a minimális méret amelyet a gyártó javasol, nem szerettem volna nagyobb kapacitású kártyát használni, mert a nagyobb méretű kártya kezelése kis mértékben lassíthatja annak kezelését. A kártya formázását az *sdcard.org* webhelyről letölthető alkalmazással végeztem. Azt követően a *balena.io* domain alatt található Etcher alkalmazást telepítettem. Ennek segítségével az operációs rendszer image fájlját az SD kártyára töltöttem, amelyet az után behelyeztem a Jetson SD kártya foglalatba.[5][11]

Mivel rendelkezésemre állt billentyűzet, monitor, ethernet csatlakozási lehetőség valamint egér, nem volt szűkség úgynevezett headless módban telepítenem az operációs rendszer. Ezen felül így elegendő volt az usb porton keresztül egy 2 A-es tápegységet csatlakoztatni (ekkor még nem készült el a jack aljzathoz szűkséges tápkábel). Az első bekapcsolás után elindul az operációs rendszer telepítése amely pár alapbeállítással kezdődik, úgymint végfelhasználó licensz elfogadás, hálózati beállítás, stb. A felhasználó létrehozása és bejelentkezés után megjelenik az Ubuntu 18.04 LTS felülete. Ez a verzió 2023-ig támogatott.

Mivel a Jetson összes usb portja foglalt (egy szabad, ha nem használunk WLAN modult), szerettem volna távoli elérést beállítani, mivel így a jármű teljesen összeszerelt állapotában sem egeret sem pedig billentyűzetet nem lehet csatlakoztatni. Első körben a TeamViewer programot próbáltam telepíteni, azonban több verzió (Linux, Raspberryy) próbája után sem jártam sikerrel. Végül az OpenVPN alkalmazás installálása működött megfelelően. Távoli eléréshez, valamint más eszközökkel történő hálózati kommunikációhoz két fix IPv4-es címet állítottam be: vezeték nélküli hálózati összeköttetéssel 192.168.0.10, míg ethernet kábellel történő csatlakozás esetén a 192.168.0.11-es címet.[11]

2.1.1 Robot Operating System (ROS)

A robotika tudománya olyan szerkezetek kutatásán alapul, melyek előre megírt algoritmusok szerint hajtanak végre feladatokat félautomata vagy automata módon. Ezért a robotika és a számítástechnika egymással szorosan egybe fonódó tudomány területek. Utóbbi egy gyorsuló tempóban fejlődő ága a mesterséges intelligencia, amely megjelenése nagy hatással volt a robotika világára. Ma már a tudományos munka mellett sokan foglalkoznak robotok fejlesztésével hobbi szinten is, sőt a LEGO játékgyártó vállalatnak köszönhetően a gyermekek is egészen fiatalon megismerkedhetnek a robotok építésével és programozásával.

A robotika elég összetett tudomány, mivel hardveres és szoftveres ismereteket is igényel. Egy jó szoftver fejlesztő nem biztos, hogy rendelkezik megfelelő hardver ismeretekkel, esetleg ha valaki nagyon jó a gépi látás terén nem biztos, hogy ismeri a szervo motorok vezérélését. Ezt felismerve 2007-ben az egyesült államokbeli Stanford Egyetemen egy olyan keretrendszer megalkotását kezdték meg, amely segítséget nyújthat a szakemberek számára a robotok fejlesztése során. Ennek eredményeképp született meg a Robot Operating Sysytem, röviden ROS.[12]

Az ROS egy összekötő kapocs a felhasználó és a hardver között, egy úgynevezett middleware. Nem operációs rendszer, azonban nem is alkalmazás. Nyílt forráskódú keretrendszer, amely az egyik leg népszerűbb ilyen típusú middleware. Ennek köszönhetően nagy közösség használja és fejleszti. Saját, jól felépített könyvtárrendszerrel rendelkezik. A ROS első verziója óta már számos változat került kiadásra. 2014-ben elindult a ROS2 is, amely létrehozását az indokolta, hogy egyes megoldások implementálásához nagy mértékben és alapjaiban véve kellene módosítani az ROS első verzióját. A projekthez én az

ROS bevált és kiforrott kiadását, a Melodic Morena-t választottam, mivel ez még támogatott (2023-ig akárcsak a Jetson operációs rendszere) és stabil kiadás.[12]

2.1.1.1 Az ROS működése

Az ROS működése úgynevezett csomópontok egymás közti információ cseréjén alapul. Ezen csomópontok mindegyike egy jól meghatározott feladatot ellátó folyamat. Valójában ezek mindegyike egy forrásfájl amelyet csomópontként, úgynevezett *node*-ként kell az ROS rendszer munka könyvtárában, a *workspace*-ben definiálni. A könyvtár szerkezete kötött.

```
catkin_ws/                -- WORKSPACE
  src/                    -- SOURCE SPACE
    ...
  build/                  -- BUILD SPACE
  devel/                  -- DEVEL SPACE
    setup.sh
    setup.bash
    setup.zsh
    ...
  install/                -- INSTALL SPACE
    setup.sh
    setup.bash
    setup.zsh
    ...
```

15. ábra A workspace felépítése

A workspace négy almappát tartalmaz: *src, build, devel, install.* Az source (src) mappában kell definiálni a különböző csomagokat (*package*). Csomagból létrehozható több is, attól függően hogyan szeretnénk strukturálni a workspace-t. A csomag létrehozásakor generálunk egy a csomagon belüli source fájlt, amelybe a node forráskódja kerül. Minden csomag létrehozásakor keletkezik egy leíró file, a CMakeLists.txt. Ebben a szöveg fájlban a csomaghoz tartozó konfigurációs beállítások tárolódnak. Ha egy forrásfájlból csomópontot szeretnék létrehozni, ebben a txt állományban kell regisztrálni. A node-ok működése attól függ, hogy milyen funkciót tölt be. Egyes csomópontok információt közölnek, míg mások információt gyűjtenek. Az információ csere node-ok között két eltérő módon történhet attól függően, hogy milyen jellegű a csomópontok közti kapcsolat. Nagy több-több egy irányú üzenet közlés esetén célszerű a ROS *service* szolgáltatást használni. Egy-egy vagy egy-több

kapcsolatú node-ok között rugalmasabb és célravezetőbb a *publish/subscrice* eljárást alkalmazni. Ebben a projektben is ez a módszer került implementálásra.[13][14]

Azt a node-ot amely meghatározott metódus szerint (meghatározott frekvenciával, vagy csak ha az adat változik, stb) adatot közöl, *publish* típusúnak nevezzük. Az adatot fogadó (valójában kiolvasó) pedig a *subscriber* típusú. Az elnevezés abból adódik, hogy a csomópontok között nem közvetlen adatcsere valósul meg mint a service használatánál. A publish node egy úgynevezett *topic*-ba küldi a közölni kívánt adatot, amelyből bármely más subscriber típusú node olvashatja azt, amelyik úgymond feliratkozik az adott topic-ra. Ez egy közlő-feliratkozó(k) kapcsolat. Az üzenetek részben kötött formátummal rendelkeznek. Az ROS egy egyszerűsített leíró nyelvet használ az üzenetek azaz a *message*-ek leírására. Ezen üzenet leíró fájlok .msg kiterjesztéssel rendelkeznek és az ROS csomag msg alkönyvtárában találhatóak. A message leíró fájlnak két része van: a mezők és az állandók. A mezők az üzenetben elküldött adatok, amely *mező típus – mező név* szerkezetűek, például *int32 x*. Az ROS telepítésével kapunk számos, előre definiált üzenet csomagot. Az alapvető (*empty, string, stb*) üzenet típusokat az *std_msgs* tartalmazza, míg például a gyakori általános robot specifikus üzeneteket a *common_msgs (geometry_msgs/Twist)*.[15]

2.1.1.2 A ROS főbb elemei

Az ROS funkciói közé tartozik például a folyamatok (csomópontok) kezelése, az azok közötti kommunikáció megvalósítása, alacsony szintű eszköz vezérlés és az adatok rögzítése, valamint megjelenítése. Az ROS ezeket a funkciókat különböző eszközök segítségével valósítja meg. A projekt során ezek közül főként az alábbiakat használtam:

- *catkin* – ROS infrastruktúra és build rendszer. A robot jármű projektben a szűkséges csomagok egy catkin workspace-ben lettek implementálva. Különböző szolgáltatásokat nyújt, például a *catkin_make*-kel ’build’-elhetjük a workspace-t egyszerűen.
- *roslaunch* – egy ROS eszköz amely segítségével egyszerűen indíthatók csomópontok. Ahhoz, hogy megfelelően működjön, a node.hoz legalább egy xml formátumú konfigurációs fájlnak kell tartoznia amely *.launch* kiterjesztésű. Ezen fájlok tartalma meghatározza az adott csomópont megfelelő futásához szűkséges beállítandó paramétereket.

- *roscore* – tulajdonképpen a roscore is egy roslaunch eszköz. A roscore parancsot futtatva előre definiált folyamatok indulnak el. Ezen alapvető folyamatok az ROS rendszer működéséhez szükségesek, mint például a node-ok futtatása és a köztük megvalósított kommunikáció.
- *rosbash* – egy olyan csomagokból álló gyűjtemény, amely megkönnyíti az ROS konzolban történő használatát. Egy ilyen eleme a *rosrun* amellyel úgy indíthatók futtathatók a node-ok, hogy elegendő csak az adott csomag és node nevét megadni az elérési útja helyett.[14]

2.1.2 Csomópontok

Ahhoz, hogy a robot jármű hibátlanul működjön és igény szerint fejleszthető legyen, létre kell hozni egy stabil alap szoftver környezetet. Ehhez az ROS-ben úgy kell kialakítani a node-okat és kommunikációjukat, hogy azok felépítése és egymás közti kapcsolata logikus és átlátható legyen. A projekt esetében több csomópontra van szükség.

2.1.2.1 Mozgás vezérlő node

A jármű fő feladata a helyváltoztatás a felhasználói szoftver utasításainak megfelelően. Ehhez szükség van egy csomópontra, amely a megfelelő sebesség és irány eléréséhez szükséges adatokat továbbítja a robot vezérlő szoftverének amely közvetlenül a motorokat mozgatja. Ennek megfelelően ez egy publisher, azaz adat közlő node. A csomópont elnevezése *motordriver_pub*, amely utal a funkciójára és típusára. Két fő információt továbbít amely a mozgáshoz szükséges: az egyik a jármű szükséges sebessége, míg a másik nem egyenes vonalú mozgás esetén fellépő szögsebesség. A vezérlő szoftver a sebességi adatot m/s-ban, míg a szögelfordulást rad/s-ban várja bemeneti információként. Mivel az ROS-ben topic-okba küldött üzenetekkel történik a node-ok közti kommunikáció, ebben az esetben is ilyen formában került megvalósításra. A sebességi és szögsebesség adatok a *geometry_msgs* üzenetcsomag *Twist msg* üzenetével kerülnek közlésre. Ez az üzenet csomag pontok, vektorok és pozíciók egységes formátumú közlését szolgálja. A Twist msg üzenet felépítése szerint két részre bontható. Egyrészt három dimenziós egyenes vonalú sebességi adatok közölhetőek az alábbi formában: linear.x, linear.y és linear.z. Az x,y és z betűjelek a mozgás irányát jelölik. Mindegyik adat float64 típusú. Másrészt ezen irányok körüli elfordulás Euler szögsebessége adható meg: angilar.x, angular.y és angular.z. Ezen adatok

szintén float64 típusúak. Mivel az autó csak két dimenzióban mozog, a vezérlő szoftver csak egy irányú – az autó hossztengelye mentén értendő – sebességi adatot vár. Ezt az üzenet linear.x változójában közli a node. A másik adat amelyet továbbítani kell, az autó függőleges tengelye körüli elfordulás szögsebessége, amely az angular.z változóba kerül. A node deklarál egy *vel* és egy *avel*, azaz velocity és angular velocity változókat:

```
extern float vel = 0;
extern float avel = 0;
```

Ezeket a változókat kell deklarálni a felhasználói szoftverben a sebesség és szögsebesség adatok meghatározásához. Az *advertise()* függvény segítségével megadható, hogy mely topic-ba szertne a node üzenetet küldeni. Ez a függvény meghívja a master node-ot is, amely nyílvántartást vezet arról, hogy mely csomópontok feliratkozók és melyek közlők. A függvény lefutása után a master node értesíti az összes csomópontot amely az adott témára feliratkozó, valamint létrehoz köztük egy peer-to-peer kapcsolatot. Az advertise() függvény egy *publish()* függvényt ad vissza, amely meghívásával lehetővé válik az adott topic-ba történő üzenet küldés. Az advertise() függvény másik paramétere az üzenet közzétételéhez használt üzenetsor mérete. Amennyiben az üzenet közzététele gyorsabb mint amennyit el lehet küldeni, ez az érték adja meg, hogy hány üzenetet kell pufferelni.[16]

```
ros::Publisher motordriver_pub = n.advertise<geometry_msgs::Twist>("cmd_vel",
1000);
```

A motor motordriver_pub csomópont esetében az üzenet a *cmd_vel* topicba kerül kiküldésre. A felhasználói szoftvertől kapott sebesség értékek átkerülnek egy-egy átmeneti változóba, majd az üzenet megfelelő elemei ezen változók értéki alapján kerülnek módosításra. Azt követően az üzenet elküldésre kerül:

```
velocity = vel;
avelocity = avel;

geometry_msgs::Twist msg;
msg.linear.x = velocity;
msg.angular.z = avelocity;

motordriver_pub.publish(msg);
```

A node fizikailag az Jetson Nano-n helyezkedik el, a motorokat vezérlő szoftver rész pedig az OpenCR modulon, két panelt pedig usb kapcsolat köti össze. Ennek megfelelően a csomópontnak kezelnie kell ezt a soros szabványos átviteli módot. Ehhez szűkség van a *serial.h* header fájl betöltéséhez. Ha ez megtörtént, a csomópont törzsében meg kell nyitni a port kapcsolatot, hogy a kommunikáció a két modul között létre jöhessen. Ehhez az alábbi kódsorokra van szűkség:

```
    try{
        ser.setPort("/dev/ttyACM0");
        ser.setBaudrate(115200);
        serial::Timeout to = serial::Timeout::simpleTimeout(2000);
        ser.setTimeout(to);
        ser.open();
    }

    catch (serial::IOException& e){
        ROS_ERROR_STREAM("Unable to open port ");
        return -1;
    }

    if(ser.isOpen()){
        ROS_INFO_STREAM("Serial Port initialized");
    }else{
        return -1;
    }
```

2.1.2.2 LIDAR kezelés

A robot autonóm működésének tovább fejlesztéséhez elengedhetetlen, hogy valamilyen módon feltérképezhető legyen annak közvetlen környezete. Erre a célra több féle érzékelési eljárás elérhető. A SLAM (Simultaneous Localization And Mapping) egy technológia amely egyidejű lokalizációt és leképzést jelent. Egy olyan folyamat, amely során a robot szenzorok segítségével felméri környezetét, arról egy térképet készít és képes abban elhelyezni magát. Ezért az alapja a folyamatos mérés, különösen a robot mozgása közben. Minél gyorsabban mozog, annál nagyobb gyakorisággal szűkséges a környezet felmérése. [17]

A jármű jelenleg egy egycsatornás LIDAR-ral van felszerelve, amely képes 25m-en belül feltérképezni a környező objektumokat. Ez a típusú szenzor másodpercenként 16000 pont felvételét teszi lehetővé. Mivel az autó mozgási sebessége kicsi (kisebb mint 1km/h), ez a mintavételezési érték és hatótáv előre láthatólag megfelelőnek bizonyul. A projekt egyik követelménye a jármű ROS kompatibilis működése. Fontos, hogy a különböző érzékelők által szolgáltatott adatok is kezelhetők legyenek az ROS-en belül. A beépített RPLIDAR A3

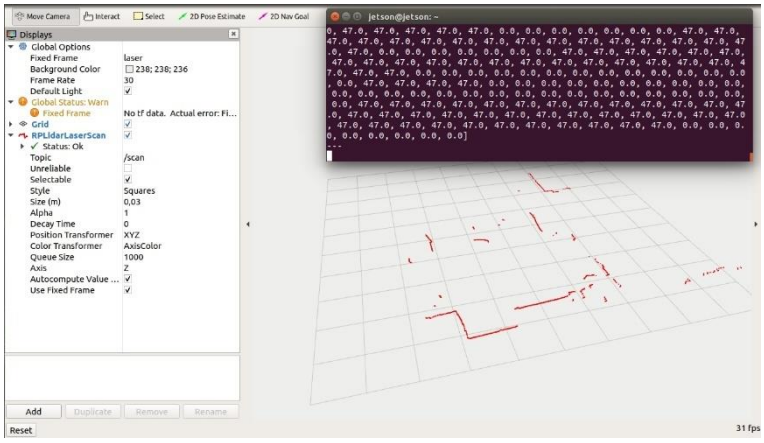
teljes mértékben megfelel ezen elvárásnak. Az ROS rendszerben a csomópontok szolgáltatásokon keresztül is kommunikálhatnak egymással közvetlenül. A robot autó projektben ezzel ellentétben topic-oka küldött üzenetek útján valósul meg a node-ok közti adatközlés.[8]

Ahhoz, hogy a LIDAR adatok elérhetővé váljanak a felhasználó számára, létre kell hozni egy csomópontot amely megadott topic-ba küldi üzenet formájában a szenzor által generált pontokat és az azokhoz tartozó adatokat. Így egy feliratkozó típusú csomópont létrehozásával könnyen olvasható a ponthalmaz. Ezen bemeneti információk elengedhetetlenek egy későbbi trajektória tervező algoritmus működéséhez.

A Slamtec terméktámogatása nagyon jól működik. A LIDAR publisher node-ját nem szükséges külön elkészíteni. A Github.com webhelyen elérhető a vállalat egy repository-ja amelyet klónozva a csomópont könnyen létrehozható. Ehhez *ctrl+T* billentyűzetkombinációval nyitnunk kell egy terminál ablakot. A *cd ~/catkin_ws/src* paranccsal az ROS workspace source könyvtára megnyitásra kerül. A *git clone* paranccsal egy git verziókövető által kezelt repository-t lehet klónozni, vagyis a repository-ban szereplő könyvtár és fájlrendszert lemásolni, ezzel egy klónt létrehozni. A megnyitott terminál ablakba a *git clone https://github.com/Slamtec/rplidar_ros.git* parancsot beírva, a catkin workspace source könyvtárába létre hozunk egy *rplidar_ros* nevű csomagot. A csomag felépítése lehetővé teszi az azonnali használatot. Az ROS-ben a node-ok különböző módon futtathatók. Létre hozhatunk úgynevezett *launch* fájlokat amelyek segítségével több csomópont és szolgáltatás indítható egyidőben. A launch fájlok xml formátumúak és *.launch* kiterjesztésűek. Különböző paraméter beállításokat és node indítási műveleteket definiálnak.[18]

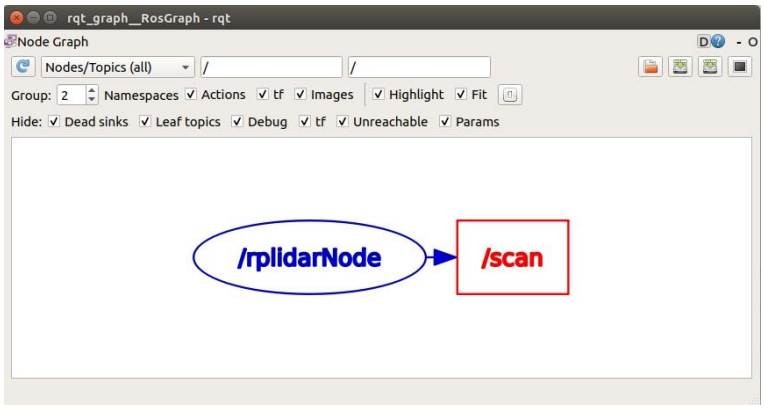
A LIDAR megfelelő működéséhez, első használat előtt be kell állítani pár paramétert. Mivel a LIDAR usb porton keresztül kap elektromos tápellátást, valamint usb porton keresztül valósítja meg az SBC-vel történő kommunikációt is, szükséges a port használatát engedélyezni az eszköz számára. A port engedélyezés és írás engedélyezés az alábbi bash parancsok segítségével adható meg: *sudo chmod 666 /dev/ttyUSB0*. Ellenörizzük, hogy a LIDAR csatlakoztatva van-e a parancs kiadásának időpontjában, ellenkező esetben a művelet hibát eredményezhet. Ha nem várható a szenzor más porthoz történő

csatlakoztatása, akkor az usb port kiosztást rögzíthetjük a *./scripts/create_udev_rules.sh* parancs futtatásával.[18]



16. ábra A launch fájl indításának és a "rostopic echo scan" parancs futtatásának eredménye

A klónozott repository alkalmas a Slamtec RPLIDAR A1, A2 és A3 típusok használatára egyaránt, mivel mindegyik típushoz tartalmaz launch fájlt. A projekt esetében a az A3-as típus került beépítésre, így az ahhoz tartozó *view_rplidar_a3.launch* launch fájlt kell használnunk. A launch fájlok a *roslaunch* paranccsal futtathatók. A parancs szintaxisa: *roslaunch csomag_neve launch_fájl.* Így a lidar node indításához az alábbi parancsot kell egy új terminál ablakban futatni: *roslaunch rplidar_ros view_rplidar_a3.launch.*[18]



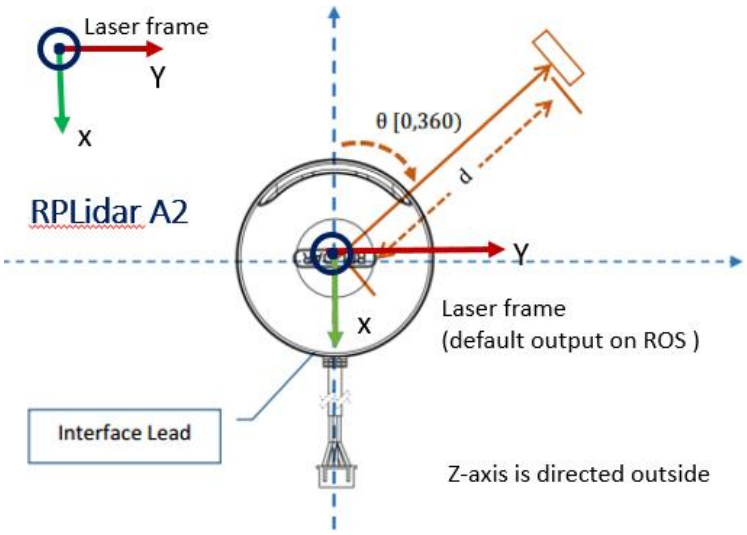
17. ábra A LIDAR kommunikációja az ROS-ben

Ez az indító fájl nem csak a LIDAR subscriber node-ját indítja. A launch fájl elindít egy *rviz* ablakot is. Az rviz az ROS eszköze, egy interface amely segítségével megjeleníthetők a nyers adatok grafikusan és strukturálva. A robot modellje is megjeleníthető ha az definiálva lett xml formátumban.[19]

Az rviz segítségével realtime módban látható a szenzor által létrehozott ponthalmaz, valamint kiolvashatók az azokhoz tartozó adatok. A launch fájl ezen felül elindítja a szenzort magát, azaz a forgó mozgást és bekapcsolja a lézer fejet, valamint elindítja az rplidarNode nevű csomópontot amely a mért értékeket továbbítja a megfelelő topic-ba.[19]

A ponthalmaz adtai a *scan* nevű topic-ba kerülnek továbbításra. A *sensor_msgs* egy üzenet csomag amely szabványos és általánosan használt szenzorok adatainak közlésére szolgál. Ilyenek pélgául a kamerák és a lézeres távolságmérők.[19][20]

Az üzenet *sensor_msgs/LaserScan msg* struktúrájú. Az üzenet tartalmaz egy fejlécet amely azonosítja az adott üzenetet. A fejléc tartalmaz például időbélyeget és id azonosítót amelyek segítségével egyértelműen azonosíthatóak. Ezen felül az üzenet többi adata 32 bites lebegőpontos, a detektált pontok azonos típusú tömbökben kerülnek tárolásra. Az üzenetben lévő adatok az alábbiak: a scan-nelés kezdő szöge (radiánban), a befejező szöge, a mérések közti szögeltérés (radiánban), mintavételek közt eltelt inkrementálódó idő (másodpercben) amely mozgó jármű esetében lehet hasznos információ, két mintavétel közt eltelt idő (másodpercben), minimális mért távolság (méterben) és a maximálisan mért távolság. A pontok távolsága és azok intenzitása külön tömbökben tárolódnak.[20]

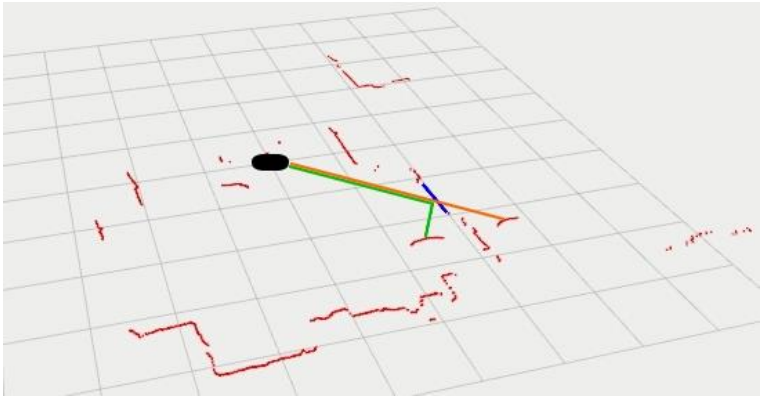


18. ábra RPLidar A3 koordináta rendszere (az ábrán az A2 LIDAR látható, azonban az A3 koordináta rendszere azonos az A2-ével)

A szenzor fej forgásakor a nulla radián a szenzor elektromos vezeték bekötésével szemközti oldalán van. A szenzor oly módon került felszerelésre, hogy a nulla érték az autó eleje, azaz

a lineáris haladási irányba mutat az X tengely mentén. Ez a későbbi útvonal tervezési feladatok során válik fontos információvá.

A robot tesztelése során a LIDAR scanner-t változatos körülmények között teszteltem. Próbáltam beltéren és kültéren egyaránt, ugyanis a gyártói ajánlás alapján mindkét környezetben megfelelően működik. Ősz lévén nem tudtam erős napsütésnek kitenni, azonban a kültéri használat során nem vettem észre szignifikáns különbséget a beltéri használathoz képest. Azonban egy hálószobai teszt során tapasztaltam egy a LIDAR működési elvéből adódó hiányosságot, amely egy későbbi trajektória tervezési algoritmus tesztelésénél hibát eredményezhet, ha a robot csakis a ponthalmaz szegmentálás útján jut információhoz a környezetével kapcsolatban.



19. ábra LIDAR mérési hiba tükröződő felületen. Kék: tükörfelület, zöld: lézerfény valós útja, narancs: lézerfény látszólagos útja.

A szobában található egy tükör felület, amely a LIDAR számára „nem látható”. A szenzor az általa kibocsájtott és az objektum felületéről visszavert lézer fényt detektálva méri a távolságot. Ha a szenzor lézer fényének beesési szöge nem merőleges a tükör felületére, akkor a fény a tükör felületéről nem a szenzor irányába verődik vissza. Ez azt eredményezi, hogy a mért érték nem a LIDAR és a tükör síkjának távolságával lesz egyenlő, hanem ehhez hozzá adódik a tükörfelület és az objektum távolsága amelyre a fény a tükörről érkezik. Ezen felül a tükröződő tárgy pozíciója is a tükör síkja mögé kerül. Ezen hibalehetőség elkerülése érdekében javasolt egy más technológiájú (ultrahang, sztereó kamera, stb.) szenzor felszerelése, amely működése nem eredményez ilyen jelenséget a tükröződő felületek esetében.

2.2 OpenCR1.0

A Robotis OpenCR panelje egy operációs rendszer nélküli, univerzális programozható vezérlő kártya. Működése attól függ, hogy milyen célszoftver kerül feltöltésre a memóriájába. Tehát a működéshez mindenképp egy más eszközön fejlesztett kódot kell létre hozni, majd azt a panel programmemóriájába tölteni. Ehhez szükség van egy IDE (Integrated Development Environment) szoftverre, amely képes kezelni az OpenCR-t.[3]

A Robotis Turtlebot3-hoz készült setup leírásában az eredetileg a turtlebot-hoz használt Raspberry PI-n keresztül történő leírás szerepel. Azonban úgy gondoltam, hogy célra vezetőbb ha más úton oldom meg a szoftver feltöltést, mivel a robot felépítése nem egyezik a Turtlebot-éval, sőt az SBC típusa sem azonos. Ebből adódóan a Turtlebot telepítéshez képest kibővített funkciókra is szükség lehet. A Robotis erre a célra az Arduino nyílt forráskódú, integrált fejlesztő környezetet ajánlja. Sajnos ez az IDE nem működik ARM architektúra alatt. Annak telepítése sem lehetséges, ennek megfelelően a Jetsonra nem sikerült installálnom. Windows7 és 10 alatt notebook-on megfelelően működik, így azonban meg kell bontani a jármű OpenCR és Jetson közti usb kapcsolatot, ha szoftvert szeretnék módosítani, mivel a kártya csak egy usb csatlakozóval rendelkezik. Ezen felül programozható JTAG/SWD csatlakozón vagy UART-on keresztül is, azonban az usb porton keresztül lényegesen egyszerűbb. A notebook-ot vagy asztali PC-t usb kábellel kell összekötni, melynek a panel oldali csatlakozója micro usb-b kivitelű.[3]

Az arduino alapesetben nem kezeli az OpenCR modult. Ehhez fel kell vinni a további panelek listájába az alábbi módon: a File > Preferences menüjébe lépve megadhatjuk mely egyéb board-ok at szeretnénk használni. Itt a megfelelő beviteli mezőbe megadhatjuk a Robotis ide vonatkozó github linkjét ami egy json fájlra mutat. Segítségével az OpenCR beállításai installálásra kerülnek. Azt követően már kiválaszthatóvá válik az OpenCR panel és a beállítandó COM port a kommunikációhoz a Tools > Board, valamint a Tools > Port menüben. A megfelelő port csak akkor lesz aktív, ha a panel a számítógéphez van csatlakoztatva, windows esetében ez COMx formátumú lesz.[3]

2.2.1 Motorok címzése

Az egyik nagy különbség az épített robot autó és az eredeti Turtlebot3 között, hogy a Turtlebot kettő, míg a projekt jármű három szervo motort tartalmaz. A Turtlebot készletbe a Robotis a motorokat már szoftveresen megcímezve adja. Mivel két készletet használtam, két motornak ugyan az volt a címe, ezért az egyik címét módosítanom kellett. A legegyszerűbb megoldásnak a Dynamixell Workbench vagy a Dynamixel Wizard szoftwerek valamelyikének használata tűnt, azonban mindkettő esetében többször is telepítési hibába futottam. Amikor sikerült feltelepítenem a Workbench szoftvert, az nem működött megfelelően, nem találta a csatlakoztatott motort. Szerencsére az arduino tartalmaz minta kódokat az OpenCR panelhez, azon belül is a Dynamixel motorok kezeléséhez. Többek között motor címzéshez is. A mintaprogram a Fájl > Examples > OpenCR > DynamixelWorkbench > ID_Change elérési úton betölthető. A kód feltöltésével a motor id címe módosítható, azonban a kód feltöltése előtt el kell végezni a szükséges módosításokat. Az alábbi konstansok definiálási értékét módosítani kell az alábbiak szerint:

```
#define BAUDRATE 1000000
#define DXL_ID 2        //a csatlakoztatott motor aktuális id-ja
#define NEW_DXL_ID 3    //az új id
```

A módosítás után a kódot fel kell tölteni az OpenCR panelre és annak lefutása után a motor új címet kap. Fontos, hogy ekkor csak az átcímezni kívánt motor legyen csatlakoztatva a modulhoz azzal az aktuális id-val, amely definiálva lett DXL_ID konstansként. Ellenkező esetben az a motor is új címet fog kapni. A robot autó esetében a bal hátsó az „1”-es, a jobb hátsó a „2”-es, míg a kormányzó motor a „3”-as címet kapta. Mivel a motorok egyedi id-val rendelkeznek, mindegy milyen sorrendben csatlakoznak a vezérlő panelhez. Ha egy motor a későbbiekben meghibásodik és lecserélésre kerül a címzési eljárást újra végre kell hajtani, ha annak id-ja nem egyezik a kiszerelt motoréval.[21]

2.2.2 Kormányzó motor alap-és végpozíció

A jármű kormányzása ugyan azon típusú - Dynamixel XL430-as – motorral került megépítésre mint a meghajtás. Ennek a típusnak a felbontása 4096 impulzus/fordulat, azaz ~11,38 impulzus/fok. Ezen típusú motor négy különböző üzemmódban üzemeltethető: sebesség szabályzó, pozíció szabályzó, kiterjesztett pozíció szabályzó és PWM (Pulse Width Modulation) szabályzó mód. A sebesség szabályzás esetében a motor folyamatos forgását egy adott sebesség érték megadásával vezéreljük, a pozíció szabályzás esetében egy 0-360° tartományban megadhatjuk a motor kívánt pozícióját, míg a kiterjesztett pozíciós szabályzásnál ez a tartomány nagyobb, azaz a motor több fordulatot is megtehet a kívánt pozíció eléréséhez (Pl. ha a célérték 520°), továbbá a PWM szabályzás esetében a motor sebességét egy négyszögjel kitöltési tényezőjének változtatásával állíthatjuk be. A kormányzás esetében a pozíció szabályzás a megfelelő üzemmód, ugyanis a kormányszerkezet kialakításából adódóan a kerekek két irányú maximális kitérése között a motor elfordulása nem éri el a teljes fordulatot, azaz a 360°-ot.[1]

A motor abszolút jeladóval rendelkezik. Ez azt jelenti, hogy egy kódtárcsát tartalmaz amely több sávos, amely lehetővé teszi a tengely abszolót pozíciójának meghatározását. Így a motor bármely pozíciójában megállapítható annak elfordulási mértéke a nullpontjához képest még egy tápfeszültség szünet után is. Ezzel szemben az inkrementális encoder-ek csak egy vagy több sávban rovátkolt tárcsát tartalmaznak, így az elfordulás mértékéhez számolni kell a sávokat. Ez problémát jelentene abban az esetben, ha az autó tápellátása nem a kormányszerkezet alappozíciójában kerülne elvételre. Úgy az alappozíció beállítást mechanikusan kellene megoldani.[1]

Mivel az XL430-as abszolút jeladóval rendelkezik, meg kell határozni azt a pontját amelybe állítva a kormányszerkezet középállásban van. A kerekeket mechanikusan egyenes futás állásba kell pozícionálni és ekkor le kell kérdezni a motor aktuális pozícióját. A kerekek dőlésének és párhuzamosságának beállítása után a járművel 1 m-es egyenes vonalú teszt futásokat végeztem. Így meghatározható volt az a beállítás, amely során a jármű nem tért el az egyenes útvonaltól. A kormányszerkezet középállásához tartozó motor pozíció így már meghatározható.[1]

A motor pozíció lekérdezésénél fontos beállítani az átviteli sebességet, amely ennél a motor típusnál *1000000*. Ezen túl oda kell figyelni, hogy a megfelelő motor pozícióját kérdezzük le. Ellenkező esetben valamelyik hajtó motor pozíciója lesz beállítva, ami sérülést eredményezhet a kormányszerkezetben az első teszteléskor. Amennyiben a motor pozíció kiíratást egy végtelen ciklusba helyezzük, úgy fizikai állítás közben is kaphatunk folyamatos visszajelzést a motor pozícióról.[21]

```
#define BAUDRATE  1000000
#define DXL_ID    3
```

A motor aktuális pozícióját meg kell határozni teljes kerék kitérés esetén is, hiszen ez lesz az az érték, ameddig a motort mozgatni szabad. Az eljárás megegyezik az alappozíció meghatározásával azzal a különbséggel, hogy ebben az esetben a kormányzott kerekeket mindkét irányba ki kell téríteni mechanikusan, majd az arduino IDE port monitorozásával megállapíthatók a szélső értékek.[21]

2.2.3 Vezérlő szoftver

Az OpenCR modul fő feladata a hardwer környezet kezelése a Jetson Nano-tól érkező utasításoknak megfelelően. Elsősorban a motorok mozgatása, valamint szenzor adatok továbbítása. A Robotis a Turtlebot különböző változataihoz biztosítja azt a szoftver törzset, amely segítségével a fejlesztők kialakíthatják saját robotjukat. Ennek megfelelően a Turtlebot3 Burger és Waffle verziójához is biztosít az adott hardver felépítésnek megfelelő core szoftvert. Ezen két verzión felül a Robotis indított több projektet számos hardver felépítéssel. Ezek közül a Turtlebot bike felépítése áll legközelebb a robot autóhoz. Két hajtott kerék és egy kormányzott. Azonban a bike három kerekű és a core szoftvere csak a Robotis RC100-as konrollerrel történő vezérlésre alkalmas. Ahhoz, hogy a szenzor adatokat továbbítani tudjuk a Jetson felé, nagy mértékben módosítani kellene a törzs szoftvert.[21][22]

A Turtlebot burger OpenCR szoftvere több ponton alkalmasabb a robotautó projekthez, azonban több ponton szűkséges módosítani, mivel a Turtlebot burger csak két motort kezel. Előnye, hogy az ROS kompatibilis vezérlés feltételei részben implementálásra kerültek. A szoftver három részből áll, egy main és két header fájl. A ’turtlebot3_burger.h’-ban kerültek

definiálásra a robot fizikai kialakításából adódó konstansok. A ’turtlebot3_core_config.h’ fájl főként konfigurációs beállításokat valósít meg, úgymint a szükséges header fájlok betöltése, változók deklarálása, konstansok és függvények prototípusainak definiálása. A ’turtlebot3_core’ program valósítja meg a fő funkciókat, amelyek főként a ROS node-ok működése, perifériák (motorok, buzzer, visszajelző led-ek,stb) kezelése. A megfelelő működéshez azonban ezeket módosítani szükséges a jármű fizikai kialakítása szerint.[21][22]

2.2.4 Ackermann bicikli modell

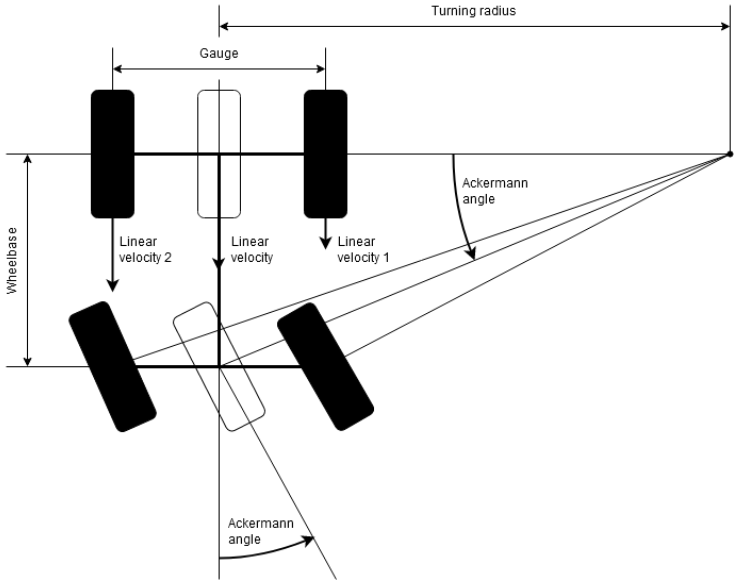
A jármű későbbi fő felhasználási célja önvezető jármű algoritmusok tesztelése. Ezért fontos, hogy képes legyen egy előre meghatározott útvonalat követni, vagy a környezetet figyelembe véve – például egy sávban – haladni. Az útvonal követésének matematikai leírása lényegesen egyszerűbb pontszerű testek vizsgálatakor. Azonban a hagyományos felépítésű, négykerekű közúti járművek esetében már összetettebb a feladat. A járművet mozgó pontként modellezve a pont csak abban az esetben esik egybe annak középpontjával (pontosabban a jármű tengelyeinek felező pontját összekötő szakasz felező pontjával), ha mindkét tengelye egyenlő mértékben, szimmetrikusan kormányzott. Azonban a legtöbb esetben az egyik tengely kerekei rögzítettek és a jármű hossztengelyével párhuzamosak, míg a másik tengely kerekei a kormányzott kerekek.

Az útvonal tervezésnél és a korrekciós számításoknál rendkívül fontos figyelembe venni a jármű sajátosságait és fizikai korlátait. Ilyen például a legkisebb fordulókör sugara. Ha egy olyan ív kerül meghatározásra az útvonalon, amely ennél a sugárnál kisebb, akkor a robot nem lesz képes a tervezett vonalon haladni. A jármű fizikai komplexitása megnehezíti az útvonal eltérés korrekciós számításait, ezért a fentebb említett pontszerű test modell közelítése a cél. Az Ackermann kormányzást használó járművek esetében egy általános módszer annak kerékpárrá történő leegyszerűsítése, modellezése.

Az Ackerman bicikli modell esetében a kerékpár első kereke a jármű első kerekei, míg a hátsó kereke a jármű hátsó kerekei kombinációjaként hozható létre. Ennek megfelelően a helyettesítő kerekek az egy tengelyen lévő kerekeket összekötő szakasz felezőpontjában helyezkednek el. Ebben a modellben a két helyettesítő kerék tengelyének metszéspontja fogja megadni a fordulókör középpontját. A két tengely által bezárt szög és az első Ackermann kerék, valamint a jármű hossztengelye által bezárt szög merőleges szárú szögek, tehát egyenlő nagyságúak. Kerékpár esetében a hátsó kerék halad a jármű tényleges útvonalán, míg az első kerék állása párhuzamos a hátsó kerék által leírt ívnek, az első kerék tengelye által kimetszett pontjába húzott érintőjével. Ennek megfelelően a jármű forduló sugara a tengelyek metszéspontja és a hátsó Ackermann kerék középpontjának távolságával egyenlő. A derékszögű háromszögek trigonometriai azonosságait felhasználva kiszámítható az adott forduló ívhez tartozó első kerék kitérési szöge:

$$\tan \delta = \frac{L}{R} \tag{1}$$

ahol L (Wheelbase) a bicikli modell tengely távolsága (megegyezik a jármű tengelytávolságával), R (Turning radius) a fordulókör sugara, δ (Ackermann angle) pedig a két tengely által bezárt szög, ami egyenlő az első kerék adott R sugarú fordulókörhöz tartozó első kerék kitérítési szögével. Itt a hányados várhatóan 1-nél jóval kisebb érték lesz minden esetben, mivel a robot maximális kerék kitérítése 45° alatt van (tan 45°=1).[2][23]



22. ábra Az Ackermann bicikli helyettesítő modell

A projekt jármű esetében a tengelytávolság 140 mm, a nyomtáv szélesség a hajtott hátsó kerekeknél 160 mm, míg az első kormányzott kerekeknél 170 mm. A két tengely közti nyomtáv szélesség esetében tapasztalható eltérés nem okoz problémát, mivel az Ackermann bicikli modell kormányzott kerekének szög számításánál csak az első tengely, míg a hajtott kerekek – nem lineáris mozgás esetén fellépő – sebesség különbség számításnál csak a hátsó tengely nyomtáv szélességével kell számolni.

A robot járművel szemben támasztott követelmények közt szerepelt annak könnyű reprodukálhatósága, valamint egyszerű javíthatósága, ezért a kormányzás mechanikai kialakítása során kereskedelmi forgalomban kapható alkatrészek kerültek felhasználásra. Ennek eredményeként sajnos nem sikerült elérni, csak közelíteni az ideális Ackermann kormányzási geometriát. Ugyanis Charles Jentaud egy 1878-ban, az Ackermann-elv alapján épített járműve fejlesztése során azt tapasztalta, hogy a legpontosabb kerék szöget abban az esetben éri el, ha a tengely-kormányösszekötő rúd-kormány csapok alkotta trapéz szárainak meghosszabbítása éppen a hátsó tengely felező pontjában metszik egymást. A robot autó esetében ez a metszéspont a hátsó tengely előtt helyezkedik el.

A kormányzáshoz több olyan alkatrész került felhasználásra, amely tervezése nem a projekt keretein belül történt és fizikai kialakításuk is eléggé összetett, így azok CAD modellezése nem minden esetben történt meg. Ezért a modell lapján nem határozhatók meg a különböző kormány állásokhoz tartozó kerék szögek. Ennek megfelelően a kormányzott kerekek kitérését méréssel határoztam meg. Ez a módszer ugyan nehézkesebb mint a modell alapján történő mérés, azonban így meghatározhatók a valós szögek, amelyekre hatással vannak az alkatrészek méretpontossága és az azok közti illeszkedési hibák. Természetesen itt számolni lehet a mérési hibákkal is, azonban azokat igyekeztem minimálisra csökkenteni.

A mérés egy sík asztal felületen történt, amelyre az autó rögzítésre került. A robot hossztengelyére merőlegesen kimérésre és felrajzolásra került egy bázis vonal. Azt követően a kerekek szöge öt pozícióban került lemérésre. A mérés előtt a kormányzott kerekekről a gumiabroncsok eltávolításra kerültek, hogy azok rugalmas felülete ne befolyásolja a mérést. Egy 300 mm-es fém vonalzót a kerekek külső felületéhez illesztve, felrajzolásra került az asztalra a vonalzó mentén az aktuális kerék szög. Így a bázis vonalhoz képest mérhetővé vált a kerekek kitérési szöge.

1. Táblázat A kormányzott kerekek mért kitérési szögei

Kerék szög	
Bal	**Jobb**
18,0°	24,0°
12,0°	15,0°
7,5°	9,5°
4,0°	5,5°
-0,5°	0,5°
-5,5°	-4,0°
-9,5°	-7,5°
-15,0°	-12,0°
-24,0°	-18,0°

A kerekek mért szögét tartalmazó táblázatból látható, hogy középállásban a kerekek egymással nem párhuzamosak, kitérésük egyenként 0,5° egymásnak ellentétes irányban. Azért választottam ezt a megoldást, mert ebben az esetben a jármű egyenes vonalú mozgásánál a kerekek – a talajjal történő súrlódásának következtében – kis mértékben a haladási iránytól távolodni igyekeznek ellentétes irányban. Így a kormány mechanika alkatrészeinek illeszkedési hibája, az esetleges kotyogások nem befolyásolják a jármű egyenes tartását, mivel a kerekek folyamatosan egymásnak feszülve tartják a kormányrudakat. A kerekek maximális kitéríthetőségét az autó fizikai kialakítása korlátozza. A mérés során ez a maximális szög 24°-ban került meghatározásra.

A kormányzott kerekek szöge alapján és a kormányzott tengely nyomtávjának ismeretében az (1)-es képletet felhasználva, valamint azt átrendezve megállapítható az adott kerék kitérési szöge által meghatározott forduló sugár nagysága:

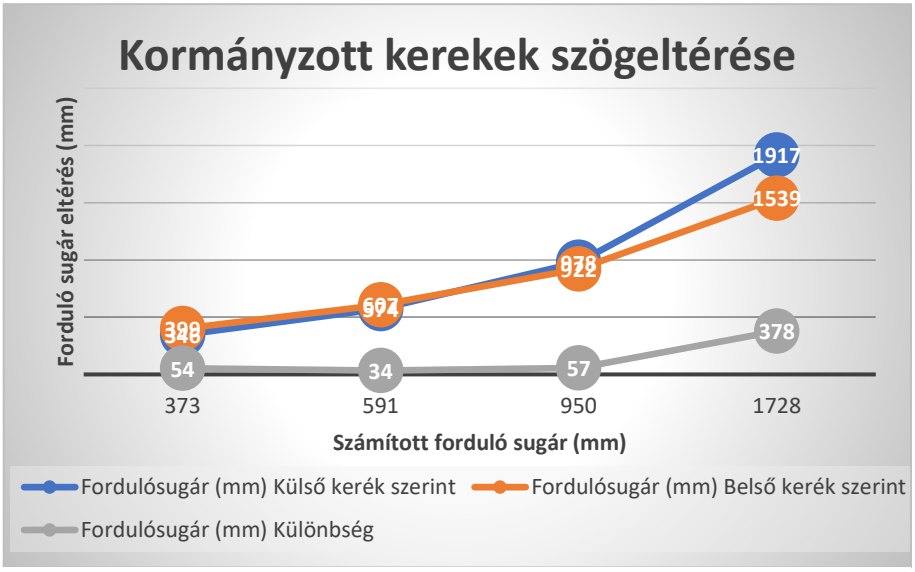
$$r = \frac{Tengelytáv}{tg\ \alpha} \pm Nyomtáv/2 \qquad (2)$$

A 20. ábra szerint a forduló belső oldala felé eső kerék esetében a nyomtáv felét hozzá adni, míg a külső kerék esetében kivonni kell a tengelytáv és kerék kitérési szög tangensének hányadosából.

2. táblázat A jármű forduló sugara az 1. táblázat kerék szögekből számítva

Fordulósugár (mm)		
Külső kerék szerint	Belső kerék szerint	Különbség
346	399	54
574	607	34
978	922	57
1917	1539	378

A 2. táblázat számított adataiból látszik, hogy a kerekek szöge nem felel meg teljes mértékben a várható értékeknek, ami így a két kerék állásából számított forduló sugár különbséghez vezet. Ez két okra vezethető vissza: a kormány mechanika kialakításakor nem a projekt járműhöz tervezett alkatrészek használatából eredő - az optimális kialakítástól való - geometriai eltérés, valamint a kerekek egyenes vonalú mozgáshoz beállított 0,5°-os deviancia.



23. ábra A kerekek szög eltérésének szemléltetése

A nagyobb eltérés az egyenes vonalú mozgáshoz közeli szögek esetében jelentkezik, azonban ezekben az esetekben ez a forduló sugár nagyságához viszonyítva elenyésző, így várhatóan ez nem fog trajektória követési hibát okozni. Ezen felül a legkisebb forduló sugár tartományában tapasztalható nem számottevő differencia növekedés, amely az Ackermann

kormányzás és az ideális kormányzás közti eltérésnek tudható be nagyobb kitérési szögek esetében.[2]

Adott kerék szög állás esetén – a kerekek mért kitérési szöge alapján számolt – a kerekekhez tartozó forduló sugár érték pár középértékét véve, kiszámítható az Ackermann helyettesítő kerék szöge az alábbi képlet segítségével, amelyet az (1)-es képlet átrendezésével kapunk:

$$\alpha = \text{arctg}\left(\frac{Tengelytáv}{r}\right) \tag{3}$$

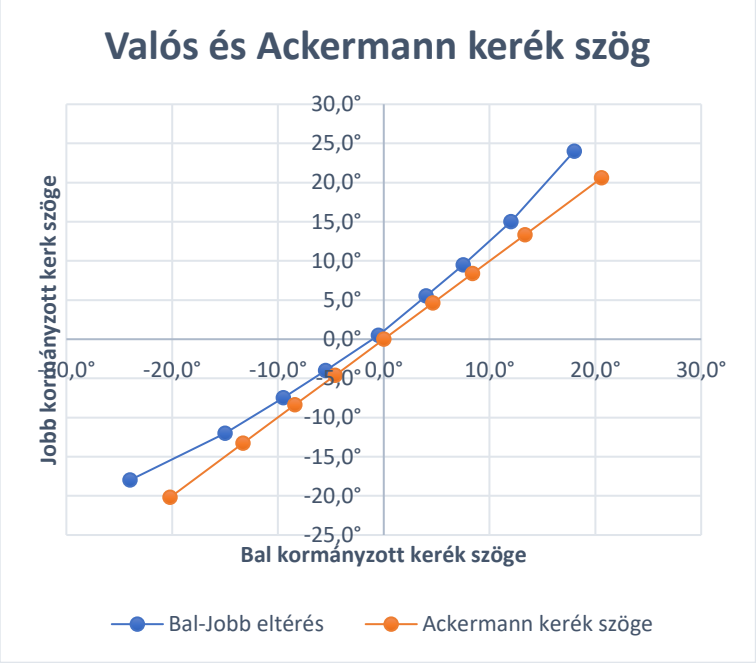
A számított adatokból már megállapítható két fontos adat amely az autó későbbi használata során elengedhetetlen ahhoz, hogy az útvonal tervező és követő algoritmusok megfelelően működjenek:

- legkisebb forduló sugár: 373mm
- legnagyobb Ackermann szög: 20,2°

3. Táblázat Forduló sugár és Ackermann kerék szög számítás

Kerék szög		Kerék forduló sugár (mm)		Számított forduló sugár (mm)	Ackermann kerék szög
Bal	Jobb	Bal	Jobb		
18,0°	24,0°	431	314	373	20,2°
12,0°	15,0°	659	522	591	13,3°
7,5°	9,5°	1063	837	950	8,4°
4,0°	5,5°	2002	1454	1728	4,6°
0,5°	0,5°	-	-	-	0,0°

Amennyiben a későbbiekben ezek a paraméterek túl kicsinek bizonyulnak, úgy abban az esetben növelni kell a kormányzott kerekek nyomtáv szélességét annak érdekében, hogy a kerekek kitérése növekedjen, ezzel párhuzamosan a forduló sugár nagysága pedig csökkenjen. Azonban úgy gondolom, hogy a jelenlegi paraméterek méretarányosan megfelelnek egy valós méretű közúti autó jellemzőinek.



24. ábra A valós kerék szögek és a számított Ackermann helyettesítő kerék szöge.

A jármű vezérlése a kívánt sebesség és szögelfordulás megadásával történik. Ehhez azonban ismerni kell, hogy ezen értékeknek milyen mértékű Ackermann bicikli kerék szög beállítás felel meg. Ha egy jármű egy körív mentén halad, valamint annak kerekei és az útfelület között megfelelő a tapadás azaz nincs oldalirányú sodródása, akkor a sugár irányú, azaz a centripetális sebessége nullával egyenlő:

$$v_r = r' = 0 \tag{4}$$

Ebből következik, hogy a jármű lineáris sebessége megegyezik a kerületi sebességével:

$$v = \sqrt{V_r^2 + v_\omega^2} = \sqrt{0 + v_\omega} = v_\omega \tag{5}$$

A kerületi sebesség függ az autó által leírt fordulósugártól és a szögsebességétől, azok szorzata:

$$v_\omega = r\omega \tag{6}$$

Ebből kifejezhető a sugár:

$$r = \frac{v}{\omega} \tag{7}$$

Ahhoz, hogy a kormányzó motornak megfelelő pozíció paramétert tudjunk átadni, elengedhetetlen a forduló sugár ismerete. A jármű a felhasználói vezérlő szoftvertől a lineáris sebességi és szögelfordulás adatokat várja. Ezek ismeretében az (7)-es képlet segítségével számítható a kívánt trajektória ívhez tartozó sugár. Az (1)-es képletet átalakítva

és a sugár helyére az (7)-es képletet behelyettesítve a lineáris sebesség, a szögelfordulás és a jármű tengelytávjának ismeretében már számítható az Ackermann bicikli modell kormányzott kerekének szükséges kerék kitérítési szöge:

$$\alpha = \text{arctg}\left(\frac{Tengelytáv}{v/\omega}\right) \tag{8}$$

Az OpenCR modulra a Robotis által nyújtott Turtlebot burger szoftver került feltöltésre. Azonban a burger verzió fizikai kialakításából adódóan ez a kód csak két kereket kezel. Ezért azt ki kell egészíteni a harmadik (kormányzó) tengelyre vonatkozó utasításokkal. Ehhez pár konstans értéket definiálni kell. A turtlebot core szoftverébe illesztve van egy config header file, ami a szükséges változók deklarálását és függvények prototípusainak definiálását végzi, ebbe a config fájlba pedig egy másik header fájl került beillesztésre, amely főként alapvető konstansok definiálását végzi. A harmadik tengely szabályzásához szükséges konstansok definiálására a leginkább megfelelő a config fájl, mivel a core program törzsébe kerülnek a tengelyhez tartozó függvények.

A kormányzó motor megfelelő működéséhez szükséges meghatározni az alábbi konstansokat:

- maximális Ackermann kerék szög – *ack_max*
- az autó tengelytávolsága – *wheelbase*
- a kormányzott kerekek közép állásához tartozó motor pozíció – *ax3_ctr*
- a kormányzott kerekek bal oldali teljes kitérítéséhez tartozó motor pozíció – ax3_ll
- a kormányzott kerekek jobb oldali teljes kitérítéséhez tartozó motor pozíció – ax3_lr

Ezen felül szükséges segéd változók deklarálása is, ami inkább a core programba célszerűbb, ahol a szabályzáshoz szükséges számítások is implementálásra kerülnek:

- számított Ackermann kerék szög radiánban – ax3_ang_rad
- maximális Ackermann kerék szög radiánban – ack_max_rad
- számított (relatív) motor pozíció eltérés a középálláshoz képest az Ackermann szög szerint – ax3_rel_pos
- számított (abszolút) motor pozíció képest az Ackermann szög szerint – ax3_pos

```
#define ack_max     20.2    //fok
#define wheelbase   0.14    //méter
#define ax3_ctr     1200    //axis3 center - nem valós adat, lekérdezés
                                szükséges
#define ax3_ll      688     //axis3 limit_left - nem valós adat, lekérdezés
                                szükséges
#define ax3_lr      1712    //axis3 limit_right - nem valós adat, lekérdezés
                                szükséges
```

A konstansként definiált paraméterek mindegyike a jármű fizikai kialakításától függ, így ha olyan változtatást eszközölünk az autón ami ezekre befolyással lehet, úgy ezen paramétereket újra meg kell határozni és a definiált értéket a programkódban meg kell változtatni. *Egy motor csere esetén ügyelni kell arra, hogy a működési tartományba semmiképp se essen bele annak nullpontja, mert azesetben a motor pozíció számítás nem lesz megfelelő.*

```
double imp_rad;         //motor encoder impulzus / radián
double ax3_ang_rad;     //számított Ackermann szög radiánban
double ack_max_rad;     //maximális Ackermann szög radiánban
double ax3_dif;         //impulzus eltérés a középállástól
double ax3_abs_pos;     //relatív motor pozíció a középálláshoz képest
```

A változók között van tizedes tört, valamint pozitív és negatív egész. Azért választottam double típust, hogy elkerüljem a számítások során szűkséges sdsttípus-konverziót. A változók a core szoftver setup() függvényébe kerültek, hogy ne legyenek a loop() függvényben újra és újra definiálva a program futása során. A motor impulzus / radián konverzióhoz tartozó változó azért nem került a konstansok közé, mert értéke függ a maximális kerék kitéréstől és attól, hogy a kormányzó motor encoder-e mekkora felbontású.

Egy ív mentén történő mozgás során a négy kerekű járműveknek nem csak a kormányzott kerekei haladnak eltérő sugarú körív mentén, hanem a hajtott kerekei is (22-es ábra). Hagyományos (nem kerékagy motoros) járműveknél, akár belső égésű motorral, akár központi elektromos motorral szerelt, ezt a sebesség különbséget a gyártók differenciálmű beépítésével küszöbölik ki. A robot autó esetében ez a megoldás nem kivitelezhető a fizikai kialakítás miatt, mivel a két hajtott kerék külön motorokkal van ellátva. Így a nem egyenes vonalú mozgás esetén jelentkező sebesség eltérést a meghajtás sebességének szabályzásával kell megoldani. A korábban megadott (7)-es képletet átrendezve látható, hogy a sebesség a forduló sugár és a szögsebesség szorzata. Ez az összefüggés nem csak az autó útvonala

esetén igaz. A képlet felírható a kerekek nyomvonalára is. A 22-es ábrán látható, hogy az íven haladó jármű külső kereke nagyobb sugarú köríven halad, azonban a szögsebessége megegyezik az autóéval és a belső íven haladó kerékével. Továbbá az is megállapítható, hogy a belső íven haladó kerék kisebb sugarú körív mentén halad. Mivel a kerekek és a jármű szögsebessége azonos, a kerekek sebessége arányos az általuk leírt körív sugarával.

$$v' = r' \cdot \omega = \left(\frac{v}{\omega} \pm \frac{Gauge}{2}\right) \cdot \omega \tag{9}$$

A fenti képletben a *v'* a hajtott kerék sebessége, míg az *r'* az adott kerék által leírt körív sugara. Mivel a kerék körívének sugara nem ismert, a (7)-es képlet behelyettesítésével átszámítható az autó sebességéből, szögsebességéből és a nyomtáv szélességből. A 22-es ábráról megállapítható, hogy a külső kerék ívének sugara pontosan a nyomtáv felével nagyobb, míg a belső kerék íve ugyan annyival kisebb mint az autó ívének sugara. Ennek megfelelően a (9)-es képletben a külső íven haladó kerék sebesség számításánál a nyomtáv fele hozzá adandó, míg a belső kerék esetében kivonandó a sebesség-szögsebesség hányadosból. A jármű nem egyenes vonalú mozgását kivéve négy eset fordulhat elő: előre haladás mellett jobbra vagy balra kanyarodás, vagy hátramenetben jobbra vagy balra kanyarodás.

Minden esetet levezetve belátható, hogy ha az autó előre haladó sebességét pozitív, míg a hátrameneti sebességet negatív előjellel, továbbá a balra fordulás szögsebességét pozitív és a jobbra fordulás szögsebességét pedig negatív előjellel kezeljük, akkor a két kerék ívben történő haladás melletti sebességét az alábbi kódsorokkal implementálhatjuk:

```
if (cmd_vel_msg.angular.z == 0){

    lin_vel1 = cmd_vel_msg.linear.x;
    lin_vel2 = cmd_vel_msg.linear.x;
}else{
    lin_vel1 = ((cmd_vel_msg.linear.x / cmd_vel_msg.angular.z) -
(WHEEL_SEPARATION / 2)) * cmd_vel_msg.angular.z;
    lin_vel2 = ((cmd_vel_msg.linear.x / cmd_vel_msg.angular.z) +
(WHEEL_SEPARATION / 2)) * cmd_vel_msg.angular.z;

}
```

A szögsebesség vizsgálata azért szűkséges, hogy megállapítsuk, hogy egyenes vonalú-e a mozgás. Amennyiben igen, úgy a két hajtott kerék sebessége egyenlő az autó sebességével. A hajtott kerekek nyomtáv szélessége megegyezik a Turtlebot3-éval amely a Robotis szoftverében már deklarálásra került *WHEEL_SEPARATION* néven és *0.160*-as értékkel. A *lin_vel1* változó a bal oldali hajtott kerék sebessége, míg a *lin_vel2* a jobb oldali hajtott kerék sebessége. Ehhez azonban előbb ezt a két változót deklarálni kell a setup() függvényben:

```
double lin_vel1, lin_vel2;
```

A hajtott kerekek sebességének ismeretében implementálható a kormányzott kerekek motorjának szabályzó algoritmusa. Amennyiben a mozgás egyenes vonalú, úgy a kormányzó motor középállásba kell mozgatni, ami az alábbi feltétel vizsgálattal és adat megadással lehetséges:

```
if (cmd_vel_msg.angular.z == 0){

    ax3_rel_pos = ax3_ctr;

}else{
      .
      .
      .
```

A robot a szögsebesség értékét rad/s-ban várja, azonban a maximális Ackermann szög fokban kerül definiálásra konstansként, ezért azt át kell váltani radiánba. Ezt az alábbi számítás segítségével valósíthatjuk meg:

```
ack_max_rad = (ack_max * PI) / 180;
```

Az Ackermann kerék kitérési szögének számítása meghatározásra került a (8)-as képlettel. Ennek implementálása a rendelkezésre álló változók függvényében az alábbi:

```
ax3_ang_rad = arctg(wheelbase / (cmd_vel_msg.linear.x /
cmd_vel_msg.angular.z));
```

A kormány szerkezet miatt, a kormányzó motor kitérési szöge nem egyenlő az Ackermann kerék kitérési szögével. Csak abban az esetben egyezne meg a két szög, ha a kormányzó motor beépítése olyan pozícióban történt volna, hogy annak tengelye és az autó függőleges tengelye egymással párhuzamosak lennének. Jelen felépítés mellett a pontos működéshez szükséges a motor mozgási tartományának ismerete. A kerék középállása és szélső helyzete közötti szögtartományt meg kell feleltetni az ugyan ebben a tartományban lévő motor encoder impulzus tartománynak. Azaz át kell konvertálni a szögállást impulzusszámmá. Az alábbi kódsorral ez megvalósítható:

```
if (cmd_vel_msg.angular.z < 0){

    imp_rad = (ax3_ctr / ax3_lr) / ack_max_rad;

}else{

    imp_rad = (ax3_ctr / ax3_ll) / ack_max_rad;

}
```

Így a kívánt szögelfordulás már meghatározható a középállástól való motor encoder impulzus eltérésként. Ehhez a kívánt Ackermann szöget meg kell szorozni a kapott váltószámmal:

```
ax3_dif = ax3_ang_rad * imp_rad;
```

Ezzel az ax3_dif változóba kerül az a motor impulzus szám, amennyivel a motort el kell fordítani a középállástól. Ez még csak egy relatív pozíció, nem az elérni kívánt kerékszöghöz tartozó abszolút pozíció. Azonban abban az esetben, ha a kerék középállása a motor null pozíciójánál lenne, ez az érték már megfelelne a valós értéknek.

Az alábbi kód arra az esetleges problémára nyújt megoldást, ha egy jövőbeni motor csere során más típusú motor kerül beszerelésre és annak encoder értéke ellenkező forgásirányba inkrementálódik:

```
if (cmd_vel_msg.linear.x > 0){
    if (cmd_vel_msg.angular.z < 0){
        if (ax3_ctr < ax3_lr){

            ax3_dif = ax3_dif * -1;

        }
    }
    if (cmd_vel_msg.angular.z > 0){
        if (ax3_ctr < ax3_ll){

            ax3_dif = ax3_dif * -1;
        }
    }
}
if (cmd_vel_msg.linear.x < 0){
    if (cmd_vel_msg.angular.z < 0){
        if (ax3_ctr < ax3_lr){

            ax3_dif = ax3_dif * -1;

        }
    }
    if (cmd_vel_msg.angular.z > 0){
        if (ax3_ctr < ax3_ll){

            ax3_dif = ax3_dif * -1;

        }
    }
}
```

Ahhoz, hogy az Ackermann szögnek megfelelő valós motor pozíciót megkapjuk, a középállás pozícióhoz hozzá kell adni az attól való eltérést. Ezzel megkapjuk az abszolút motor pozíciót:

```
ax3_abs_pos = ax3_dif + ax3_ctr;
```

2.2.5 Pozíció visszajelzés

Az önvezető funkció működéséhez feltétlenül szűkség van a megfelelő visszacsatolásra a szabályzó szoftver felé arról, hogy a jármű hogyan helyezkedik el a térben. Ezen adatok birtokában lehet csak képes bármilyen útvonal tervező vagy követő algoritmus megfelelően működni. Mivel az autó irányítása csak két dimenziós, így elegendő három információ folyamatos visszajelzése: x irányú elmozdulás, y irányú elmozdulás, valamint a jármű orientációja, azaz a függőleges tengelye körüli elfordulása. A Robotis Turtlebot3 vezérlő szoftverében a calcOdometry() függvény folyamatosan számolja a jármű elmozdulását és elfordulását három tengely körül a szenzor és motor adatok alapján. Ennek köszönhetően csak egy publisher node-ot kell a kódhoz adni, amely *geometry_msgs::PoseStamped* üzenetben közli a kívánt adatokat. Ehhez elsősorban a config fájlba kell a *geometry_msgs/PoseStamped.h* header fájlt illeszteni, a csomópontot megvalósító függvény prototípusát deklarálni, valamint a core fájl setup() függvényébe a *position_feedback* topic inicializáló függvényét beilleszteni:

```
void publishPoseStamped(void);
```

```
nh.advertise(position_feedback);
```

Azt követően a calcOdometry() függvény után implementálni kell az orientációs adatokat közlő csomópontot:

```
void publishPoseStamped(void){

    ros::Publisher position_feedback =
    nh.advertise<geometry_msgs::PoseStamped>("position", 1000);

    ros::Rate loop_rate(30);

    while (ros::ok()){

        geometry_msgs::PoseStamped poseStamped;
        poseStamped.header.frame_id="/position";
        poseStamped.header.stamp = ros::Time::now();

        goal.pose.position.x = odom.pose.pose.position.x;
        goal.pose.position.y = odom.pose.pose.position.y;
        goal.pose.orientation = odom.pose.pose.orientation;

        poseStampedPub.publish(poseStamped);
    }
```

A *position_feedback* nevű csomópont így a *position* topicba továbbítja folyamatosan az autó x, y koordináták menti elmozdulását, valamint a Θ (theta) orientációját. A későbbiekben a felhasználói szoftverben szükséges lesz egy subscriber node implementálása amely a position topic-ra feliratkozik és geometry_msgs::PoseStamped üzenet strukturájának megfelelően kiolvassa az adatokat.[24]

3 Összegzés

A projekt során törekedtem arra, hogy a feladat kiírás szerinti minden kritériumnak megfeleljen a robotjármű. Továbbá célom a feladat túlteljesítése volt plusz punkciók implementálásával. A tervezési, építési és kódolási folyamat során azonban ráébredtem, hogy ez egy nagyon összetett, rendkívül sok és széles körű ismereteket igénylő projekt, ami nagy mennyiségű időráfordítást is igényel. A feladat előtt csekély ismereteim voltak a beágyazott rendszerek terén, azonban a projekt előrehaladtával ezek gyorsan gyarapodni kezdtek a komplex munkának köszönhetően.

Mechanikai oldalról egy stabil, logikusan felépített robot járművet sikerült összeépíteni, amely hallgatói munkákhoz megfelelő. Egyszerűen és gyorsan javítható, könnyen fejleszthető. Az egyedi hardver kialakításnak köszönhetően megjelent számos akadály, amelyek okát túlnyomó többségében sikerült felderíteni, így egy későbbi javítás vagy fejlesztés folyamán ezek már könnyen kikerülhetők.

Az autó szoftveres oldalról még munkát igényel ahhoz, hogy megfelelő alapot nyújtson különböző önvezető algoritmusok teszteléséhez. Implementálni kell a motorok mozgatásához szükséges függvényt, valamint tesztelni kell a node-okat. A LIDAR és a motordriver node tesztelése megtörtént, az ROS rendszer megfelelően működik.

Mindent összevetve egy stabil felépítésű, számos lehetőséget rejtő autonóm autó platform jött létre, amely megfelel a feladat kiírásban szereplő célnak, a hallgatói projektek alapjaként.

Irodalomjegyzék

[1] "Robotis e-Manual - Dynamixel" [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/. [Accessed 01. október 2021].

[2] "Data Genetics – Ackerman Steering," [Online]. Available: https://datagenetics.com/blog/december12016/index.html. [Accessed 02. szeptember 2021.].

[3] "Robotis e-Manual - OpenCR 1.0" [Online]. Available: https://emanual.robotis.com/docs/en/parts/controller/opencr10/. [Accessed 01. október 2021].

[4] "Raspbery Pi – Raspberry Pi 3 Model B+" [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/. [Accessed 01. október 2021].

[5] "Nvidia developer – Jetson Nano Developer Kit" [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed 01. október 2021].

[6] Verőné Dr. Wojtaszek Małgorzata. „Fotointerpretáció és távérzékelés 3., A lézer alapú távérzékelés.” -Nyugat-magyarországi Egyetem, 2010

[7] "Robotis e-Manual – LDS-01" [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/. [Accessed 01. október 2021].

[8] "Slamtec – RPLIDAR A3" [Online]. Available: https://www.slamtec.com/en/Lidar/A3. [Accessed 01. október 2021].

[9] "Autodesk" [Online]. https://www.autodesk.com/solutions/cad-software. [Accessed 10. november 2021].

[10] "Onshape – ROBOTIS TurtleBot3 Burger CAD elements" [Online]. https://cad.onshape.com/documents/2586c4659ef3e7078e91168b/w/14abf4cb615429a14a2732cc/e/6c94f199b347f8785a67b6f8. [Accessed 20. szeptember 2021].

[11] "Nvidia developer – Getting Started with Jetson Nano Developer Kit" [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit. [Accessed 05. november 2021].

[12] Szögi Gábor. „Robot operációs rendszer a „ROS 2.0" lehetőségeinek tükrében." - Óbudai Egyetem Bejczy Antal iRobottechnikai Központ, 2017

[13] Cserép Máté. „Szoftver technológia – Build systems." - ELTE Informatikai kar, 2019

[14] "GitHub - horverno / ros-gyakorlatok" [Online]. Available: https://github.com/horverno/ros-gyakorlatok. [Accessed 20. szeptember 2021.].

[15] "Robot Operating System – Understanding Nodes" [Online]. Available: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes. [Accessed 20. szeptember 2021.].

[16] "Robot Operating System docs – geometry_msgs::Twist meassage" [Online]. Available: http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html. [Accessed 30. szeptember 2021.].

[17] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachnis, Wolfram Burgard. „A Tutorial on Graph-Based SLAM." - IEEE Intelligent Transportation Systems Magazine, 2010

[18] "Robot Operating System – rplidar" [Online]. Available: http://wiki.ros.org/rplidar. [Accessed 05. október 2021.].

[19] "Robot Operating System – rviz User Guide" [Online]. Available: http://wiki.ros.org/rviz/UserGuide. [Accessed 05. október 2021.].

[20] "Robot Operating System – sensor_msgs/LaserScan" [Online]. Available: http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html. [Accessed 05. október 2021.].

[21] "Arduino" [Online]. Available: https://www.arduino.cc/. [Accessed 20. október 2021.].

[22] " Robotis e-Manual – OpenCR setup" [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/#opencr-setup. [Accessed 05. október 2021.].

[23] Xiaohui Li, Zhenping Sun, Qingyang Chen, Daxue Liu. „An adaptive preview path tracker for off-road autonomous driving.” - National University of Defense Technology, 2013

[24] "Robot Operating System – geometry_msgs/PoseStamped" [Online]. Available: http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseStamped.html [Accessed 05. október 2021.].

Ábrajegyzék

Táblázatjegyzék

Mellékletek

Motor pozíció meghatározás (2.2.2)

```
#include <DynamixelWorkbench.h>

#define BAUDRATE  1000000
#define DXL_ID    3

DynamixelWorkbench dxl_wb;

void setup() {
  Serial.begin(57600);

  const char *log;
  bool result = false;

  uint8_t dxl_id = DXL_ID;
  uint16_t model_number = 0;

  result = dxl_wb.init(DEVICE_NAME, BAUDRATE, &log);
  if (result == false)
  {
    Serial.println(log);
    Serial.println("Failed to init");
  }
  else
  {
    Serial.print("Succeeded to init : ");
    Serial.println(BAUDRATE);
  }

  result = dxl_wb.ping(dxl_id, &model_number, &log);
  if (result == false)
  {
    Serial.println(log);
    Serial.println("Failed to ping");
  }
  else
  {
    Serial.println("Succeeded to ping");
    Serial.print("id : ");
    Serial.print(dxl_id);
    Serial.print(" model_number : ");
    Serial.println(model_number);
  }

  result = dxl_wb.jointMode(dxl_id, 0, 0, &log);
  if (result == false)
  {
    Serial.println(log);
    Serial.println("Failed to change joint mode");
  }
  else
  {
```

```
    Serial.println("Succeed to change joint mode");
  }
  result = dxl_wb.addSyncReadHandler(dxl_id[0], "Present_Position", &log);
  if (result == false)
  {
    Serial.println(log);
    Serial.println("Failed to add sync read handler");
  }
}
void loop()[

  const char *log;
  bool result = false;

  int32_t present_position = 0;

  do
  {
    result = dxl_wb.syncRead(handler_index, &log);
    if (result == false)
    {
      Serial.println(log);
      Serial.println("Failed to sync read position");
    }

    result = dxl_wb.getSyncReadData(handler_index, &present_position[0], &log);
    if (result == false)
    {
      Serial.println(log);
    }
    else
    {
      Serial.print(" Present Position of the steering motor: ");
      Serial.print(present_position);
    }
  }
}
```

Mozgás vezérlő node (2.1.2.1)

```
#include <ros/ros.h>
#include <stdlib.h>
#include <serial/serial.h>
#include <std_msgs/Empty.h>
#include <std_msgs/Bool.h>
#include <geometry_msgs/Twist.h>

serial::Serial ser;

extern float vel = 0;
extern float avel = 0;

int main(int argc, char **argv){

//*************************Init*****************************
```

```
    ros::init(argc, argv, "motordriver_pub");

    ros::NodeHandle n;

    ros::Publisher motordriver_pub =
    n.advertise<geometry_msgs::Twist>("cmd_vel", 1000);

    //ros::Publisher motordriver_pub =
    n.advertise<std_msgs::Bool>("motor_power", 1000);

    ros::Rate loop_rate(30);

    float velocity = 0;
    float avelocity = 0;

//**********************USB connection **********************

    try{
        ser.setPort("/dev/ttyACM0");
        ser.setBaudrate(115200);
        serial::Timeout to = serial::Timeout::simpleTimeout(2000);
        ser.setTimeout(to);
        ser.open();
    }

    catch (serial::IOException& e){
        ROS_ERROR_STREAM("Unable to open port ");
        return -1;
    }

    if(ser.isOpen()){
        ROS_INFO_STREAM("Serial Port initialized");
    }else{
        return -1;
    }

//************************Publishing****************************

    while (ros::ok()){

        //libusb_handle_events_timeout(...);

        //std_msgs::Bool;
        //msg.data = true;

        velocity = vel;
        avelocity = avel;

        geometry_msgs::Twist msg;
        msg.linear.x = velocity;
        msg.angular.z = avelocity;

        motordriver_pub.publish(msg);
    }
    return 1;
}
```