UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA

FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE DIN TÂRGU-MUREȘ

PROGRAMUL DE STUDII CALCULATOARE

APLICAREA COMENZII VOCALE PENTRU CONDUCEREA ROBOȚILOR MOBILI

Coordonator științific:

Conf. dr. ing. Domokos József

Absolvent:

Katona Andrea Izabella

2023

UNIVERSITATEA „SAPIENTIA” din Cluj-Napoca **Viza facultății:**

Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Programul de studii: **Calculatoare**

LUCRARE DE DIPLOMĂ

Coordonator științific:	Candidat: Katona Andrea Izabella
Conf. dr. ing. Domokos József	Anul absolvirii: **2023**

a) Tema lucrării de licență:

APLICAREA COMENZII VOCALE PENTRU CONDUCEREA ROBOȚILOR MOBILI

b) Problemele principale tratate:

- Studiu bibliografic privind sistemele de recunoaștere vocală
- Realizarea unei aplicații pentru recunoașterea comenzilor vocale din vorbirea continuă, în limba maghiară
- Evaluarea sistemului de recunoaștere utilizat
- Aplicarea comenzii vocale prin intermediul unui simulator pentru roboți mobili
- Construirea unui robot mobil și aplicarea comenzilor vocale pentru dirijarea acestuia

c) Desene obligatorii:

- Schema bloc a sistemului realizat
- Diagramele de proiectare software

d) Softuri obligatorii:

-Aplicație de recunoaștere a comenzilor vocale
- Softul de conectare a sistemului de comandă vocală cu simulatorul pentru roboți mobili
- Softurile pentru conectarea aplicației de recunoaștere a comenzilor cu robotul mobil construit
- Softurile necesare pentru funcționarea robotului mobil

e) Bibliografia recomandată:

- Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, Spoken language processing: a guide to theory, algorithm and system development, 2001

-Németh Géza, Olaszy Gábor, A magyar beszéd: beszédkutatás, beszédtechnológia, beszédinformációs rendszerek, 2010

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca, Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, laboratorul de electronică 113

Primit tema la data de: 31.03.2022

Termen de predare: 27.06.2023

Semnătura Directorului de departament

Semnătura conducătorului științific

Semnătura responsabilului programului de studiu

Semnătura candidatului

Declarație

Subsemnata/ul Katona Andrea Izabella, absolvent(ă) al/a programului de studii Calculatoare, promoția 2023 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia din Cluj-Napoca cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data: Absolvent

Semnătura……

Aplicarea comenzii vocale pentru conducerea roboților mobili

Extras

Oamenii au fost întotdeauna interesați de roboți. Întâlnim adesea roboți, nu numai în știință, ci și în diferite domenii ale vieții. Deși obișnuiam să ne gândim la interacțiunea om-robot ca la un subiect de science-fiction (de exemplu, R2D2 din Războiul Stelelor), în zilele noastre a devenit din ce în ce mai mult o realitate.

Deoarece comunicarea umană naturală se realizează prin vorbire, nu este dificil să ne imaginăm controlul prin comenzi vocale. Controlul vocal este un subiect fascinant care permite roboților să fie controlați prin comenzi vocale. Există diferite metode în acest sens, dar cu toate acestea cercetările în acest domeniu sunt încă în curs de desfășurare. Acest domeniu al științei poate duce la multe realizări și aplicații interesante în toate domeniile vieții (știință, divertisment, industrie etc.).

Scopul lucrării mele de diplomă este de a demonstra avantajele și potențialul controlului vocal în controlul roboților mobili. În acest scop, voi prezenta o implementare care permite controlul unui robot într-un mediu de simulare, precum și al unui robot fizic, folosind comenzi vocale. În cele ce urmează, puteți citi despre implementarea detaliată, tehnologiile utilizate, precum și despre rezultatele măsurătorilor efectuate, concluziile și posibilitățile de dezvoltare ulterioară.

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

HANGVEZÉRLÉS ALKALMAZÁSA MOBILIS ROBOTOK IRÁNYÍTÁSÁRA

Témavezető:
Dr. Domokos József
egyetemi docens

Végzős hallgató:
Katona Andrea Izabella

2023

Kivonat

Az emberek mindig is érdekeltek voltak a robotokban. Gyakran találkozhatunk robotokkal, nem csak a tudományokban, hanem az élet különböző területein is. Bár régebb úgy tekintettünk az ember-robot interakcióra, mint sci-fi (pl. R2D2 a Csillagok háborúja című filmből) napjainkban ez egyre inkább valósággá vált.

Mivel az emberek természetes kommunikációja beszéden keresztül történik, nem nehéz elképzelni a beszéden keresztül történő irányítást sem. A hangvezérlés egy izgalmas téma, ami a robotok irányítását teszi lehetővé hangparancsok által. Erre különböző módszerek léteznek azonban a területen mai napig is kutatások zajlanak. Ez a tudományág nagyon sok izgalmas megvalósítást és alkalmazást hozhat magával a továbbiakban az élet minden területén (tudomány, szórakoztatás, ipar stb.).

A dolgozatom célja, hogy bemutassam a hangvezérlés előnyeit és potenciálját a mobilis robotok irányításban. Ennek érdekében egy megvalósítást fogok bemutatni, amely egy szimulációs környezetben való robot, valamint egy fizikai robot irányítását teszi lehetővé hangparancsokkal. A továbbiakban olvashatnak a részletes megvalósításról, a felhasznált technológiákról, valamint az elvégzett mérések eredményéről, következtetésekről és továbbfejlesztési lehetőségeiről.

Kulcsszavak: mobilis robot, hangvezérlés

Abstract

People have always been interested in robots. We often meet robots, not only in science, but also in various areas of life. Although we used to think of human-robot interaction as science fiction (e.g. R2D2 from the movie Star Wars), nowadays it has become more and more a reality.

Since people's natural communication is through speech, it is not difficult to imagine control through speech. Voice control is an exciting topic that allows robots to be controlled by voice commands. There are different methods for this, however, research is still ongoing in the field. This branch of science can lead to many exciting realizations and applications in all areas of life (science, entertainment, industry, etc.).

The purpose of my work is to present the advantages and potential of voice control in controlling mobile robots. To this end, I will present an implementation that allows controlling a robot in a simulation environment as well as a physical robot with voice commands. Further on, you can read about the detailed implementation, the technologies used, as well as the results of the measurements, conclusions and further development possibilities.

Keywords: mobile robots, voice control

Tartalomjegyzék

Ábrák jegyzéke

Kódrészletek jegyzéke

Táblázatok jegyzéke

1. Bevezető

A beszéd az emberi kommunikáció legrégebbi és leggyakoribb formája. Ennek segítségével kapcsolatokat teremtünk, információkat közvetítünk, valamint gondolatainkat és érzelmeinket fejezzük ki. A beszéd segítségével tanítunk és tanulunk. Fontos szerepet játszik a társadalomban, így nem meglepő, hogy felkeltette a tudósok és kutatók érdeklődését is. Az elmúlt évtizedekben tanulmányozva a hangok tulajdonságait sikerült automatizálni a beszéd felismerését. Így születtek olyan algoritmusok és alkalmazások a mesterséges intelligencia területén, amelyek képesek kinyerni a hanghullámokból a beszéd tartalmi információit. Ez egy nagy előrelépést jelentett a technológia fejlődésében.

Napjainkban nem idegenek azok az alkalmazások vagy rendszerek, amelyek képesek a beszédfelismerésre. Vegyük példának az okos asszisztenseket (pl. Amazon Alexa, Apple Siri, Cortana, Google Assistant), amelyek jelen vannak az emberek mindennapjaiban. Ezek az eszközök képesek kérdésekre válaszolni (pl. Hány óra van?) és utasításokat végrehajtani (pl. Termék hozzáadása a bevásárló listához). Ezt hangvezérlés segítségével képesek kivitelezni, azonban ez nem csak az okos asszisztensek világára korlátozódik.

A hangvezérlés egy olyan alkalmazása a beszédfelismerésnek, amely lehetővé teszi, hogy a felhasználó hangparancsokkal vezéreljen egy eszközt. Ezek az eszközök lehetnek háztartási eszközök, okostelefonok, autók vagy robotok. Az eszközök képesek érzékelni a beszédet és a beszédből kiszűrt információ alapján végrehajtják az ennek megfelelő feladatot. (pl. “Turn ON” utasítás esetén bekapcsol a TV). A hangvezérlés lehetővé teszi, hogy a felhasználók intuitív módon használják a rendszer.

Dolgozatomban egy mobilis robot hangvezérlését fogom bemutatni, ennek részletes tervezési fázisát, valamint elvégzett kísérleteket és a belőle levont következtetéseket.

2. Célkitűzések

A dolgozatom fontosabb célkitűzései a következők:

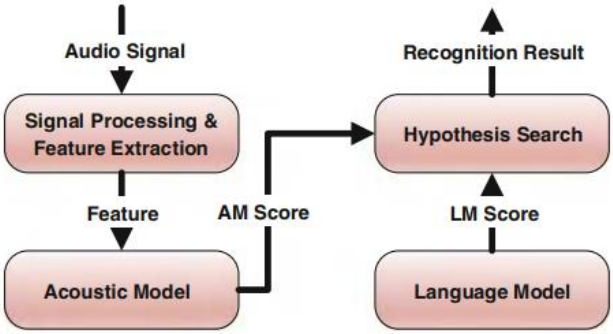
- Egy olyan rendszert megtervezni, amely képes egy vezérelt robot mozgását irányítani hangparancsok segítségével;
- Az előző pontban említett rendszert szimulálni és megfigyelni;
- A megfigyelt viselkedés nagyon fontos része a prototípus fejlesztésnek, ezért érdemes figyelembe venni a további tervezés során;
- Célunk úgyszintén egy valódi vezérelt robot tervezése, valamint megvalósítása oly módon, hogy ezt be tudjuk üzemeltetni és hangvezérléssel irányítani;
- A hangirányítás megvalósítása magyar nyelvre;
- Magasabb szintű programozási nyelven megvalósítani a robot irányítását;
- Oly módon megtervezni és dokumentálni a rendszert, hogy ezt lehessen tovább fejleszteni, illetve optimalizálni.

3. Elméleti megalapozás és szakirodalmi tanulmány

A tervezést megelőző legfontosabb lépés az elméleti megalapozás, valamint módszerek, és technológiák megismerése. A projekt megvalósítása előtt megvizsgáltam a beszédfelismerés, illetve a hangvezérlés témaköröket, valamint azokat a létező megvalósításokat, amelyek egy robot irányítását teszik lehetővé.

3.1. Beszédfelismerés

Ez a fejezet egy álltalános áttekintést nyújt egy beszédfelismerő rendszer működéséről és alkotó elemeiről. [1]



1. Ábra - Beszédfelismerő rendszer általános architektúrája [1]

A fenti ábrán (1. Ábra - Beszédfelismerő rendszer általános architektúrája [1]) látható egy általános beszédfelismerő rendszer felépítése. Ezt a rendszert különböző alrészek alkotják, mint például a jelfeldolgozó és jellemző kinyerő (Signal processing & Feature Extraction), az akusztikai modell (Acoustic Model, rövidítve AM), a nyelvi modell (Language Model, rövidítve LM) és a hipotézis keresési (Hypothesis Search) algoritmus. Ezek az alrészek külön-külön fontos szerepet játszanak a felismerési folyamat során. A rendszer bemenete rendszerint egy hangállomány, míg a kimenete egy szöveges állomány a felismert és átírt szöveggel.

A rendszer működése az alábbi lépésekből áll, amint azt a szakirodalom ismerteti:

1. A jelfeldolgozó és jellemző kiszűrő alrész a hangállományt veszi bemenetként, majd ebből eltávolítja a zajokat és a csatornatórzításokat, ezzel javítva a beszédminőséget. Az így kapott jelet leképezi időtartományból frekvenciatartományba és továbbá ezt fogjuk jellemzővektornak (feature vector) nevezni.
2. Az előző lépésben kapott jellemzővektor lesz az akusztikai modell bemenete. Az akusztikai modell tartalmazza a felismerés nyelvéhez tartozó fonetikai és akusztikai jellemzőket. A kinyert jellemzők alapján a rendszer különböző hipotéziseket generál a

lehetséges szavak vagy mondatok formájába. A hipotézisek alapján számol egy valószínűségi értéket, amit AM pontosságnak (AM score) nevezünk. Ez egy valószínűségi eloszlást jelent, amely megadja, hogy a bemenet milyent hang vagy fonéma lehet a modell alapján.

3. A nyelvi modell a kontextust és a nyelvi összefüggéseket veszi figyelembe. Ez a modell a kontextus alapján számolja ki egy lehetséges eredmény valószínűségét a jellemzővektorból, ezt nevezzük LM pontosságnak (LM score). Ez nagy mértékben javíthatja a beszédfelismerés eredményét.
4. Az utolsó fázisban, a rendszer összeadja az AM és LM pontszámokat a feltételezett szószekvenciákhoz. A legmagasabb pontszámú szószekvencia kerül kiválasztásra, és ez lesz a felismert szöveg vagy mondat eredménye.

Dale Isaacs kutatásaira alapozva [1], a mai beszédfelismerő és a szöveg alapú beszédszintézis rendszerek jól kiépítettek és a legújabb technológiákat alkalmazva a 90%-os pontosságot is meghaladhatják. Számos nyílt és zárt forrású beszédfelismerő alkalmazás létezik és nehéz lehet ezek közül a megfelelőt választani. A Veton Këpuska és Gamal Bohouta tudományos cikkében [3] a Microsoft API, Google API és CMU Sphinx beszédfelismerő rendszerek kerültek összehasonlításra. A mérések során arra a következtetésre jutottak, hogy a Google API WER-je (Word Error Rate) a legkisebb (9%), pontossága meghaladva a Microsoft API-t (18%), illetve a Sphinx-4-et (37%).

3.2. Google Cloud Speech-to-Text API

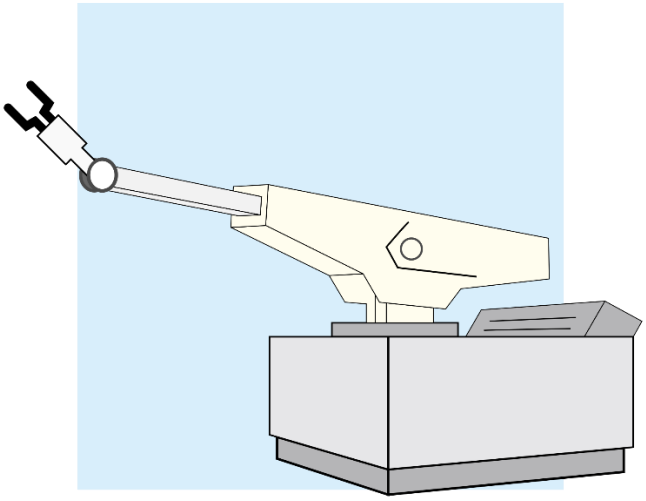
A Google Cloud Speech-to-Text API [4] egy nagyon népszerű beszédfelismerő szolgáltatás, amelyet a Google fejlesztett ki. A beszéd-szöveg átíró funkció legfeljebb 1 perces, szinkron kérésben küldött hanganyagot tud feldolgozni. Ez a funkció blokkolt, ami azt jelenti, hogy egy újabb kérés kielégítése előtt választ kell adnia. A Google Cloud Speech-to-Text API számos nyelvet támogat, többet között a magyar nyelvet is. További előnye, hogy több programozási platformon keresztül is elérhető így sok izgalmas alkalmazás fejlesztésére nyújt lehetőséget.

A Google Cloud Speech-to-Text API három fő módon hajthatja végre a beszédfelimerés folyamatát:

- Szinkron felismerés - Hangadatokat fogad, végrehajtja a felismerést és miután az összes hang feldolgozásra kerül egy eredményt küld vissza. Ebben az esetben legfeljebb 1 perces hangadatokra korlátozott;
- Aszinkron felismerés - Hangadatokat fogad, és elindít egy hosszan futó műveletet, ezzel a művelettel időnként le lehet kérdezni a felismerés eredményét. Ezt legfeljebb 480 perces időtartamokra lehet használni;
- Streaming felismerés - Hangfelismerést hajt végre és felismerés közben ideiglenes eredményeket szolgáltat, miközben a hang rögzítésre kerül. Ez lehetővé teszi, hogy az eredmény megjelenjen ameddig a felhasználó beszél

3.3. Robotok

Az emberi munka megkönnyítése érdekében olyan gépeket fejlesztettek, amelyek képesek emberi feladatokat elvégezni. Az emberek számára veszélyes vagy kimerítő fizikai munka elvégzésére is alkalmasak. Az első robot, amely az ipari munka megkönnyítésére szolgált a General Motors által használt Unimate *(*2. Ábra Unimate robot vázlata [6]*)* volt, amelyet 1958-ban az autók gyártására használták. [5]



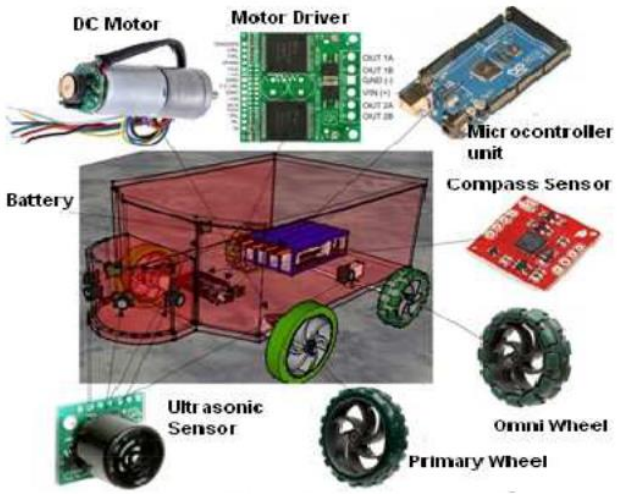
2. Ábra Unimate robot vázlata [6]

Bár a robotok remek megoldást biztosítanak az ipari munka könnyítésére, jelen vannak az orvostudományban, katonaságban és a mindennapjainkban is. Meghatározás szerint “A robot elektromechanikai szerkezet, amely előzetes programozás alapján képes különböző feladatok végrehajtására. Lehet közvetlen emberi irányítás alatt, de önállóan is végezheti a munkáját egy

számítógép felügyeletére bízva” [7]. A robotokat különböző szempontok szerint lehet csoportosítani. Ha mobilitás szempontjából szeretnénk megkülönböztetni őket, léteznek mobil (mobilis) robotok (amelyek képesek mozogni környezetükben akár emberi irányítás alatt akár autonóm módon) és helyhez kötött robotok (nem képesek helyváltoztatásra, viszont végezhetnek mozgást) [[7].

3.4. Mobilis robotok

Ahogy már említettem az előző fejezetben, a mobilis robotok képesek mozogni a környezetükben. A mozgást lábak, propellerek vagy kerekek meghajtásával érjük el. Álltalánosan egy mobilis robot felépítése egy vezérlő, szenzorok, aktuátor és egy energiaellátó rendszerből áll. [8] A vezérlő álltalában egy mikroprocesszor, beágyazott mikrovezérlő vagy akár egy személyi számítógép. Mivel a robot mozgásra van tervezve, így a robot táplálására akkumulátort szoktak használni.[8]



3. Ábra Mezőgazdasági mobilis robot alkotó részei [9]

3.5. Hasonló megvalósítások

A hasonló alkalmazások utáni kutatásom alatt találtam egy érdekes megvalósítást [[10]. Ez az alkalmazás egy IoT (Internet of Things) alapú beszédfelismerő robot távolról történő vezérlésének egy lehetséges megvalósítását mutatta be. A robot Python beszédfelismerő könyvtárat használ és alapvetően 4 utasítást képes elvégezni (előre, hátra, jobbra, balra mozdul) valamint a felhasználó által meghatározott távolságra is képes elmozdulni (pl. “Go 3 meter” – előre mozdul 3 métert). A felismerés két nyelven történik: angol és bangla. Ebben a megvalósításban a Raspberry Pi 3 volt a fő feldolgozó egység. A mikrofon, aminek segítségével a hangot rögzítik az a Raspberryhez van csatlakoztatva. Ez számomra nem jelentett megfelelő architektúrai megvalósítást, mivel a beszélő és a robot közötti távolság nagy mértékben befolyásolhatja a rögzített hanganyag minőségét.

Egy másik megvalósítás egy nagyobb projekt egyik alrészét képezi. Az EarthRover [11] egy Raspberry Pi alapú robot, aminek számos funkcionalitása közé tartozik a hangirányítás. A robot egy weboldalon keresztül implementált felhasználói felületen keresztül kapja az angol nyelvű utasításokat (pl. “robot go forward” utasítás esetén a robot előre megy) és végzi el a feladatát. Ebben a megvalósításban a beszédfelismerés az a weboldalon keresztül implementált Web Speech API segítségével történik és a mikrofon a személyi számítógéphez van csatlakoztatva. Így a felhasználó beszédéből kiszűrt szöveget az interneten keresztül továbbítja a robotnak, ez fogadja és feldolgozza az üzenetet. Ha az üzenet az utasításkészletnek része akkor a robot végrehajtja az utasítást, különben a felhasználói felületen jelzi, hogy ezt az utasítást nem ismerte fel. Ebben a megvalósításban az előzővel ellentétben létezik egy felhasználói felület, amelynek segítségével a felhasználó láthatja a felismert szöveget. Ezzel a felhasználó egy visszajelzést kap a rendszertől.

Harmadik hasonló alkalmazásként megemlíteném Xiaoling Lv és Minglu Zhang [12] által készített robot vezérlését. Az általuk megvalósított rendszerben két számítógép jelenik meg, az egyik munkafelületként működik (ez rendelkezik hangkártyával és mikrofonnal) a másik pedig a robotba van építve. A vezérlés két részből áll: egy beszédfelismerő modul és egy vezérlő modul. A beszédfelismerő modul célja a beszéd elemzése annak érdekében, hogy a robotnak milyen műveletet kell elvégeznie. A robot hét műveletet képes elvégezni: "menj előre, menj hátra, fordulj balra, fordulj jobbra, nyújtsd ki, helyezd át és állj meg" és ezeket a parancsokat kínai nyelven ismeri fel.

Következtetésként levonhatjuk, hogy bár a cél azonos (egy robot hangparancsokkal való irányítása) a megvalósítások különböztek egymástól. Ezek után úgy döntöttem, hogy az első tervezési fázisban csak egy robot szimulációt használok, annak érdekében, hogy a rendszeremet könnyebben tudjam beüzemelni. Ez abban is segített a tervezés során, hogy a robot viselkedését tudjam megfigyelni fizikai robot megvalósítása nélkül. A fentebb említett alkalmazások abban is segítettek, hogy meghatározzam, hogy milyen műveleteket szeretném, hogy elvégezzen a robotom. Az általam választott műveletek a *jobbra, balra, előre* és *hátra* való elmozdulás.

3.6. Felhasznált technológiák

3.6.1. CoppeliaSim

A CoppeliaSim egy robot szimulációs környezet, amelynek legfőbb tulajdonsága az eloszlott vezérlési architektúra, aminek segítségével minden modell/objektum egyedileg vezérelhető beágyazott szkripten, bővítményen, plugin-okon, ROS (Robot Operating System) /ROS2 csomópontokon vagy távoli API klienseken keresztül [13].

Jelenleg több mint 400 különböző API függvényt támogat, valamint 6 programozási nyelvet (C/C++, Python, Java, Lua, Matlab, Octave). További előnye, hogy számos oktatóanyagot bíztosítanak, valamint részletes magyarázatot, így a kezdők számára remek eszközt jelent.
Lehetőséget ad saját robotok megtervezésére és modellezésére, valamint biztosit előre megtervezett robotokat, valamint akadályokat, mindennapi tárgyakat, természeti elemeket és sok mást is. Remek eszköz az érzékelők és motorok modellezésére, fizikai interakciók és szimulációkra, valamint robotvezérlésre és navigációra. Platformfüggetlen (Windows, MacOS, Linux) és jelenleg legfrissebb verzió a 4.5.1-es, ami 2023 március 29.-én jelent meg.

3.6.2. ZeroMQ Remote API

A ZeroMQ Remote API segítségével külső alkalmazásokkal kapcsolódhatunk a CoppeliaSim-hez. Ez lehetővé teszi a szimuláció és egy másik folyamat vagy egy másik gépen futó alkalmazás közötti kommunikációt. Lehetővé teszi a szimuláció (vagy maga a szimulátor) vezérlését külső alkalmazásról vagy távoli hardverről (pl. valódi robotról, távoli számítógépről stb.). Mindez a felhasználó számára rejtett módon történik. A ZeroMQ Remote API lehetővé teszi, hogy egy vagy több alkalmazás lépjen interakcióba a szimulációval, sőt a távoli vezérlés is

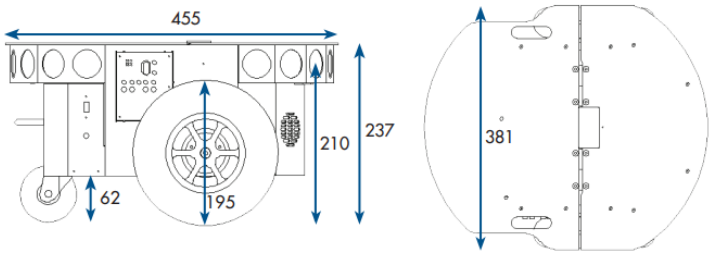
támogatott (pl. egy jelenet betöltése, szimuláció indítása, szüneteltetése vagy leállítása). [14] Jelenleg a kliens alkalmazásokat Python, C++, Matlab, Octave, Lua és Rust segítségével lehet megvalósítani.

3.6.3. Pioneer P3-DX

A CoppeliaSim számos előre meghatározott robotmodellt tartalmaz, és az egyik közöttük a Pioneer P3-DX mobilis robot. Ez a modell a valóságban is használt robotot szimulálja, és különböző funkciókkal rendelkezik a CoppeliaSim környezetben. Ez a modell lehetővé teszi a felhasználók számára a mobil robot irányítását és navigációját a szimulációban. A következő jellemzőkkel rendelkezik [15]:

- Átlag tömeg: 9 kg
- Max. Előre/hátra sebesség: 1,2 m/s
- Üzemidő: 8-10 óra 3 akkumulátorral (kellék nélkül)
- Töltési idő: 12 óra (standard) vagy 2,4 óra (opcionális nagy kapacitású töltővel)
- Áthidalható terep: Beltéri, akadálymentesített területek

4. Ábra Pioneer P3-DX [15]



5. Ábra Robot méretei mm-ben [15]

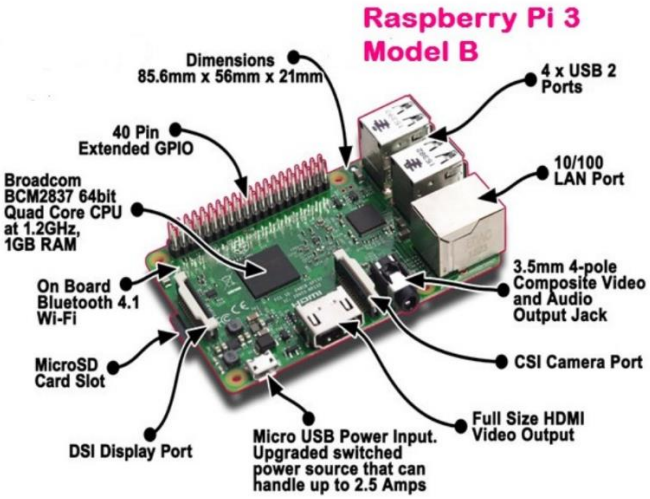
3.6.4. Raspberry Pi 3 model B

A Raspberry Pi 3 model B a Raspberry Pi harmadik generációja. Ez egy hitelkártya méretű számítógép. Egy BDM2837 system-on-chip-re (SoC) alapszik, amely egy 1,2 GHz-es négymagos ARMv8 64 bites processzort és egy nagy teljesítményű VideoCore IV GPU-t (Graphics Processing Unit) tartalmaz [16]. Ahogy már említettem, a Raspberry Pi 3 egy kis méretű számítógép, így telepíthető a Raspberry Pi OS (korábbi nevén Raspbian) és különböző perifériaeszközök segítségével (billentyűzet, egér, kijelző) asztali számítógépként is használható. Úgy volt tervezve, hogy használható legyen olyan tevékenységekre, mint amire használunk egy asztali számítógépet, így alkalmas szövegszerkesztésre, játékokra és programozásra is.

Jellemzők [16]:

- 1,2 GHz-es négymagos BCM2837 ARMv8 bites CPU
- 1 GB RAM
- VideoCore IV 3D grafikus mag
- Ethernet port
- Vezetél nélküli hálózati csatlakozás (WiFi)
- Bluetooth 4.1
- Video és audio kimeneti csatlakozó
- Négy USB port
- HDMI kimenet
- Kamera interfész (CSI)
- Kijelző interfész (DSI)
- 40 pin GPIO (General Purpose Input/Output)

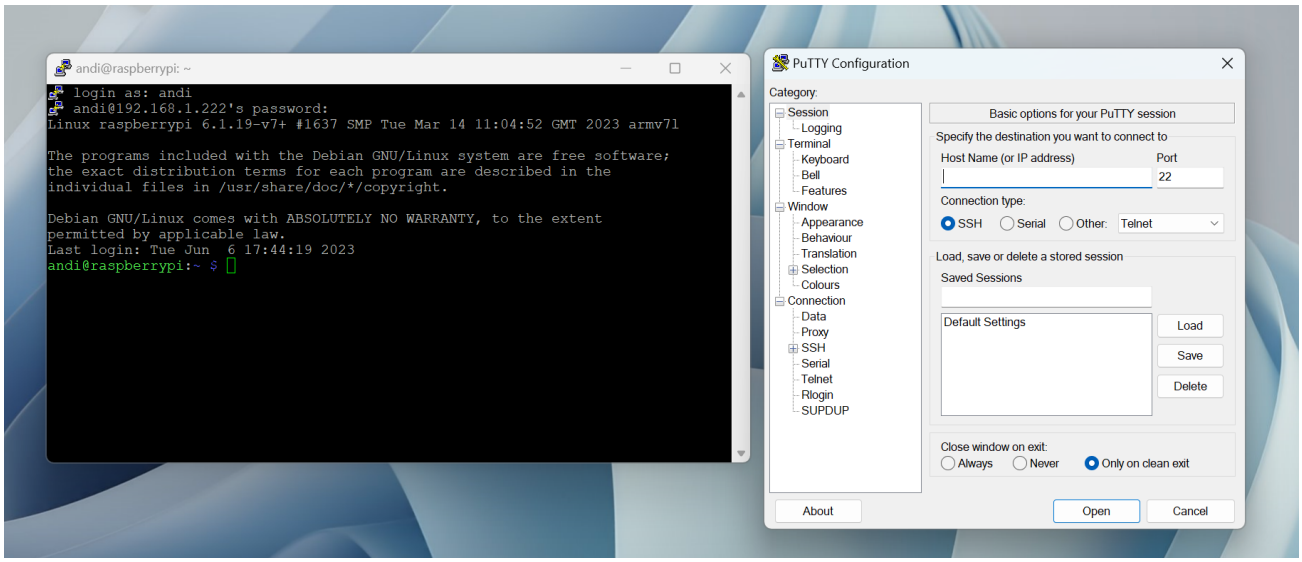
Kialakításának köszönhetően remek választás egy robot vezérlésére. A Raspberry Pi-t nem csak számítógépként lehet használni, hanem headless módban is, ami azt jelenti, hogy perifériaeszközök hiányában is használható.



6. Ábra Raspberry Pi 3 modell B szerkezete [17]

3.6.5. PuTTY

A PuTTY egy SSH és Telnet kliens, amelyet Simon Tatham fejlesztett ki Windows platformra [18]. A PuTTY egy nyílt forráskódú szoftver, amely támogatja a biztonságos távoli terminál elérését [19].



7. Ábra Raspberry Pi terminál elérése PuTTY-val

A fenti ábrán (*7. Ábra Raspberry Pi terminál elérése PuTTY-val*) látható a PuTTY-val való csatlakozás a Raspberry Pi 3 termináljához (bal oldali terminál) saját számítógépről. Ebben az esetben a Raspberry Pi headless módban van, valamint a megfelelő működéshez engedélyezve kell legyen az SSH (Secure Shell) hálózati protokoll. A PuTTY-nak meg kell adni az eszköz IP címét, valamint a SSH protokollt kiválasztani. Ajánlott a port-ot az alapértelmezett 22 értéknek hagyni. Ezek után az "Open" gomb lenyomása után felugrik a terminál, ahol a Raspberryn használt felhasználó nevét, valamint jelszavát kell begépelnünk. Ha a bejelentkezés sikeres a terminál használható a Raspberry vezérlésére.

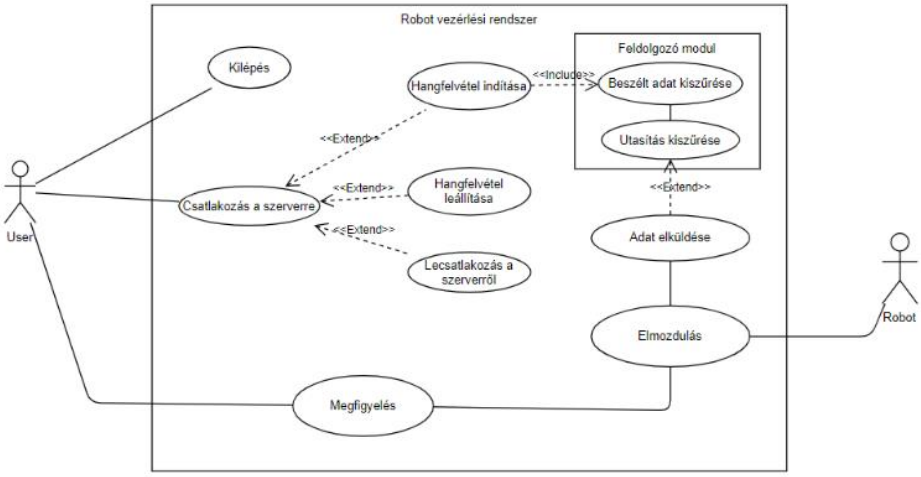
3.6.6. Python

A Python egy objektumorientált, magas szintű programozási nyelv [20]. A Pythont gyakran használják webes alkalmazások, szoftverek fejlesztésére, valamint feladatok automatizálására és adatelemzésre [21]. Gyakran használják oktatásra is, mivel szintaxisának köszönhetően beszédes, valamint könnyedén olvasható és használható. A Python aktív fejlesztői közösséggel rendelkezik, amely a könyvtárak, eszközök és keretrendszerek folyamatos fejlesztésével foglalkozik. Ennek köszönhetően a Python gazdag könyvtárkészlettel büszkélkedhet. Python programokat könnyedén lehet fejleszteni szövegszerkesztőkben (pl. Notepad++), IDE-kben (Integrated Development Enviornment) (pl. PyCharm, Spyder, Visual Studio Code) valamint online felületeken (pl. Jupyter Notebook). A választásom azért esett a Pythonra, mert nem csak a CoppeliaSim irányítását teszi lehetővé, hanem a Raspberry Pi GPIO partjainak a beállítását is támogatja.

4. Követelmény specifikáció

A rendszer zökkenőmentes működésének érdekében bizonyos követelményeknek kell megfeleljen. Ebben a fejezetben a követelményekről olvashatnak, különböző szempontból megközelítve. Fontos megjegyezni, hogy a követelmény specifikáció a fizikai robot megvalósítására érvényes, a szimulációs környezetben levő robot vezérléséről részletesebben az Első tervezési fázisban fogok beszélni.

4.1. Felhasználói követelmények



8. Ábra A rendszer Use Case diagramja

Ahogy a fenti ábrán *(*8. Ábra A rendszer Use Case diagramja*)* is látható, a felhasználónak a rendszer használatához el kell indítania a rendszer, ami lehetővé teszi a továbbiakban hangparancsok fogadását. Ennek következtében képes kell legyen a lecsatlakozásra is, amikor a rendszert már nem szeretné használni. A továbbiakban képes kell legyen hangparancsokat rögzíteni, amelyeket majd a rendszer feldolgoz és kiszűri az utasítást. Ez az utasítás fogja meghatározni a robot viselkedését. Abban az esetben, ha a felhasználó nem szeretné a felvétel készítését vagy nem akar több utasítást adni a robotnak, akkor képes kell legyen a felvételezés leállítására. Amennyiben a felhasználó mégis küld egy utasítást képes kell legyen a robot mozgásának a megfigyelésére.

4.2. Rendszer követelmények

4.2.1. Funkcionális követelmények

- A felhasználó el kell indítsa a robot szervert a PuTTY segítségével;
- Biztosítania kell, hogy a robot és az asztali számítógép ugyanarra a WiFi hálózatra van csatlakoztatva;
- A felhasználónak csatlakoznia kell a robot szerverére és csak utána küldhet hangparancsokat;
- A felhasználó környezetében nem lehetnek zajforrások (pl. hangos beszéd, zúgás vagy bármi, ami a hang minőségét nagy mértékben rontaná);
- A felhasználó a mikrofonba értelmesen és megfelelő hangsúllyal kell beszéljen, hogy a felismerés eredménye pontos legyen.

4.2.2. Nem funkcionális követelmények

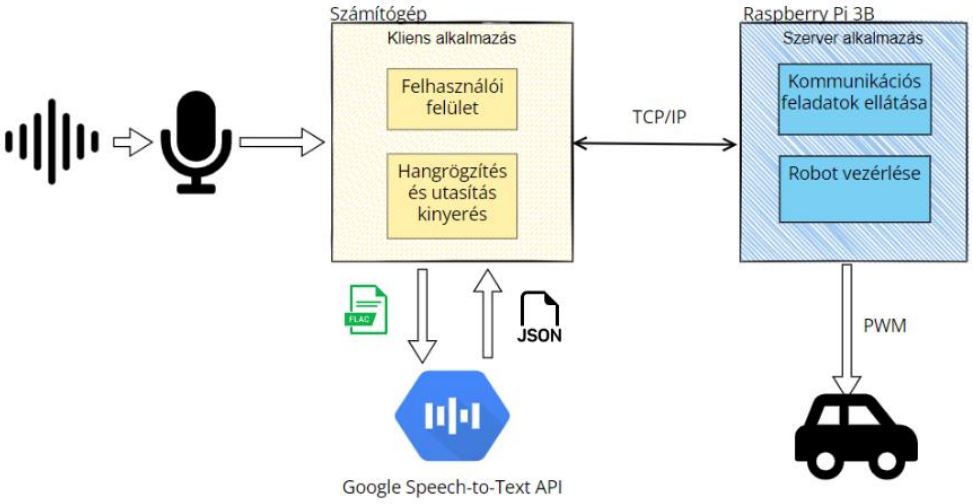
4.2.2.1 Számítógépes alkalmazás esetén

- Python 3.9.6
- Megfelelő könyvtárak telepítése (PyAudio, PySimpleGUI, SpeechRecognition – a requirements.txt tartalmazza a pontos verziószámokat)
- Operációs rendszer – Windows 11
- Stabil WiFi hálózathoz való kapcsolódás
- Legalább 10 Mbps letöltési és feltöltési sebesség
- PuTTY alkalmazás telepítése
- Mikrofon

4.2.2.2 Robot alkalmazás esetén

- Raspberry Pi 3 modell B
- Raspberry OS operációs rendszer
- Felhasználó a Raspberry OS-ben
- Python
- Robothoz tartozó komponensek
- Stabil WiFi hálózathoz való kapcsolódás
- Legalább 10 Mbps letöltési sebesség

5. A rendszer architektúrája



9. Ábra rendszer architektúrája

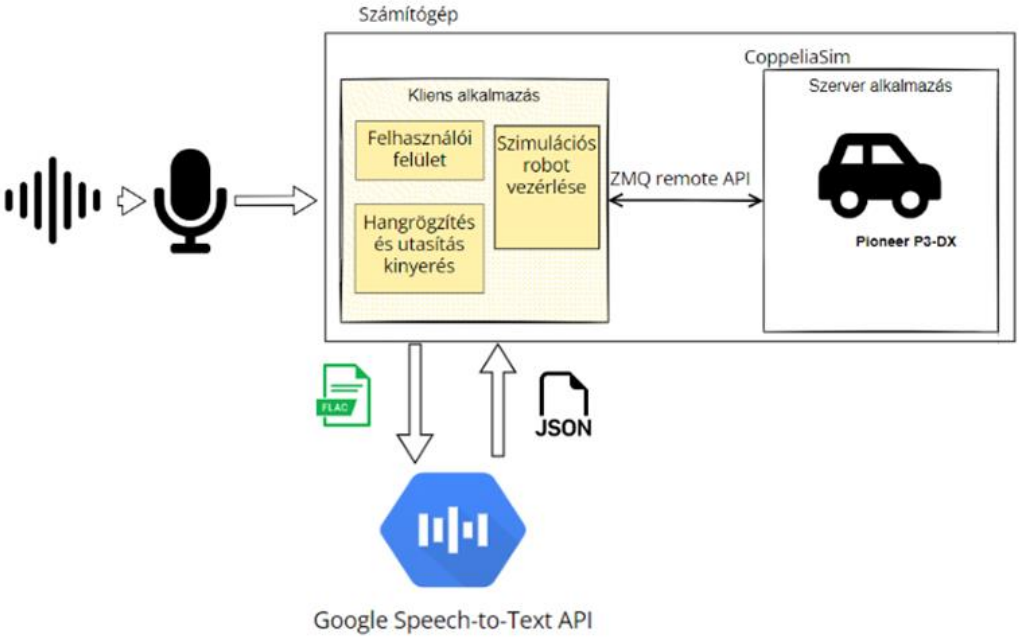
Az fenti ábrán *(*9. Ábra rendszer architektúrája*)* látható a tervezett rendszer architektúrája. Az architektúra a kliens-szerver modellt követi, amelyben a kliens alkalmazás a felhasználó álltak kiadott parancsokat feldolgozza, valamint továbbítja a szerver oldalnak végrehajtásra. A kommunikáció megvalósítására TCP/IP protokollt használtam, aminek segítségével egy kommunikációs socketet hoztam létre Python programozási nyelvben.

A rendszer fontos szereplője a felhasználó, aki a számítógéphez kapcsolt mikrofonba beszélve egy utasítást vagy utasítás kombinációt tud megadni a rendszernek folyamatos beszéd formájában, mondatokban. A felhasználó a felhasználói felület segítségével visszajelzést kap a felismerés eredményéről.

A kliens alkalmazásnak 3 fontos szerepe van. Az egyik a felhasználói felület biztosítása a felhasználó számára, valamint a felhasználói események kezelése, a második az utasítás rögzítése és feldolgozása, a harmadik meg a szerver alkalmazással való kommunikáció és adatküldés. A kliens alkalmazás a beszédből való információk szöveggé alakítását a Google Speech-to-Text API segítségével kapja meg, valamint a válaszként kapott szövegből kiszűri az utasításokat sorrendben és továbbítja a szervernek. A hangfelvétel FLAC (Free Lossless Audio Codec) hangformátumban kerül a HTTP (Hypertext Transfer Protocol) kérésbe, a válaszban kapott feldolgozott adat JSON formátumban kerül vissza.

A szerver alkalmazást a robot vezérlő szoftvere alkotja. Ezen az oldalon a Raspberry Pi eszközön fut egy szerver, ami a kliens csatlakozását, valamint a robot vezérlését teszi lehetővé. A Raspberry Pi PWM (Pulse Width Modulation) jeleket küld egy H hídnak a robot motorjainak a vezérléséhez.

5.1. Első tervezési fázis



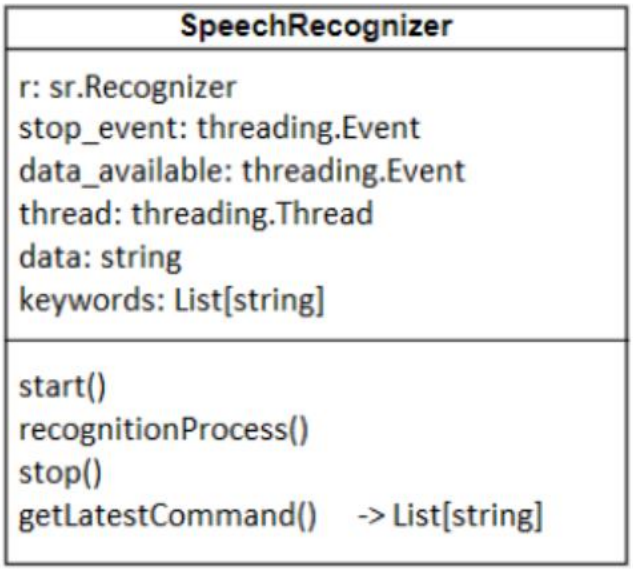
10. Ábra Szimulációs robot vezérlési rendszer architektúrája

Az első tervezési fázis során a kliens alkalmazás kiépítésére összpontosítottam, emiatt a rendszer architektúrája is megváltozott. A fenti ábrán (10. Ábra Szimulációs robot vezérlési rendszer architektúrája) látható a szimulációs robot vezérlésére szolgáló rendszer architektúrája. A kliens alkalmazás a CoppeliaSim szimulációs környezettel kommunikál, ahol a Pioneer P3-DX robot modell található. A kliens és a szerver alkalmazás is a számítógépen található. A kliens alkalmazás a ZMQ remote API segítségével vezérelheti a CoppeliaSim-ben betöltött szimulációt. A kliens alkalmazás 4 fő feladatnak kellett megfeleljen:

- Feldolgozó egységet valósít meg, amelynek szerepe a hang rögzítése, felismerési eredmény feldolgozása, valamint az utasítások kiszűrése;

- Felhasználói felületet kell kezeljen, amelynek célja a felismerés eredményének a követése, valamint a felismerés és kommunikáció vezérlése;
- Kommunikációs alegység, ennek szerepe a kapcsolat megvalósítása, valamint az adat küldés;
- Szimulációs robot vezérlése.

5.1.1. Beszédfelismerő és feldolgozó egység



11. Ábra SpeechRecognizer osztály

Létrehozott attribútumok és feladatai:

- **r**: A speech_recognition csomag objektuma, a felismerés folyamatára szolgál.

- **stop_even**t: Egy esemény, amely a felismerés folyamatának futását jelzi.

- **data_available**: Egy esemény, amely jelzi, hogy a felismerésnek van adata.

- **thread**: Egy új szál, amelyben a felismerési folyamat fut.

- **data**: A felismerésből kiszűrt adat, kezdetben üres.

- **keywords**: Lista a kulcsszavakkal, tartalma: ["előre", "hátra", "jobbra", "balra"]

Metódusok és feladataik:

– **__init__( ):** Az osztály konstruktora, inicializálja az attribútumokat.

– **recognitionProcess( ):** Ez a metódus felelős a hang rögzítésért, adatfeldolgozásért és külön szálon fut.

```
29      def recognitionProcess(self):
30
31          with sr.Microphone() as source:
32              while not self.stop_event.is_set():
33                  audio = None
34                  print("Kérlek beszélj, hallgatlak.")
35                  try:
36                      audio = self.r.listen(source, timeout=10, phrase_time_limit=8)
37
38                  except sr.WaitTimeoutError:
39                      print("Nem volt érzékelve beszéd, kezd újra!")
40                      continue
41
42                  if audio is not None and not self.stop_event.is_set():
43                      try:
44                          print("Felismerés folyamatban")
45                          text = self.r.recognize_google(audio, language="hu-HU")
46                          if text:
47                              self.data = text.lower()
48                              self.data_available.set()
49                              print("Felismert szöveg: " + self.data)
50
51                      except sr.UnknownValueError:
52                          print("Nem sikerült felismerni a beszédet.")
53                      except sr.RequestError as e:
54                          print("Hiba történt a Google Speech API hívása közben: {0}".format(e))
55
```

1. Kódrészlet – recognitionProcess() metódus

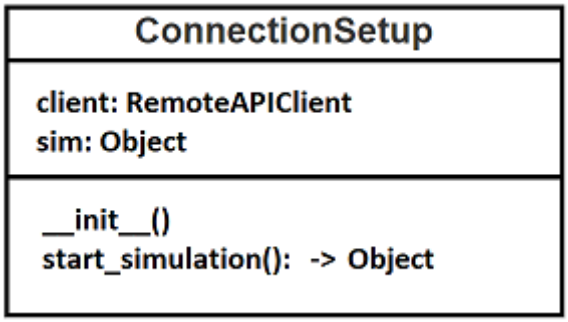
A fenti ábrán *(1. Kódrészlet – recognitionProcess() metódus)* látható a recognitionProcess metódus megvalósítása. A 31. sorban látható “with” blokkban a **Microphone** objektumnak a használata. Ennek a segítségével a mikrofon bemenetét rögzíteni tudjuk. A továbbiakban egy ciklusba lépünk, amely megvalósítja a folytonos beszédfelismerést - minden utasítás felismerése után automatikusan jöhet egy következő hangparancs feldolgozása. Ez a ciklus addig hajtódik végre ameddig a felhasználó nem állítja le a felismerés folyamatát, vagyis a **stop_event** esemény beállításra kerül. A cikluson belül a következők történnek:

- A mikrofon bemenete rögzítődik (36. sor). A timeout paraméter jelzi, hogy hány másodpercig várunk arra, hogy a felhasználó megszólaljon, a phrase_time_limit meg azt jelzi, hogy hány másodpercig beszélhet a felhasználó. Megtörténhet, hogy a felhasználó nem beszél, ebben az esetben WaitTimeoutError (38.sor) keletkezik.

- Ha sikerült a hangrögzítés és a beszédfelismerést közben nem állították le (42. sor), akkor megpróbáljuk a rögzített hanganyagot átadni a **recognize_google** metódusnak (45. sor) és a felismerés eredményét mentjük egy **text** nevű változóban. A megfelelő feldolgozás érdekében a karaktereket kisbetűkre változtatjuk, és ezután jelezzük, hogy van feldolgozásra váró adat (48. sor).
- A recognize_google segítségével hajtódik végre a felismerés. Ez a metódus a speech_recognition csomag Recognition osztályának a metódusa. A paraméterei közé tartozik a hangfelvétel, ez tartalmazza a felismerésre szánt adatot (audio), valamint a nyelv (language=”hu-HU”), amelyen szeretnénk a felismerési folyamatot. A függvény előkészíti a hangadatokat FLAC formátumba. Ha nincs megadva API kulcs, akkor használ egy generikus kulcsot. Ezután létrehoz egy URL-t az API kérés számára, majd elküldi a HTTP kérésben a hangrögzítést. A válaszból a legvalószínűbb eredmény kimenti. A függvény visszatérési értéke attól függ, hogy a show_all paraméter igaz vagy hamis. Ha igaz, akkor a nyers választ adja vissza JSON formátumban. Ha hamis, akkor a legjobb találat transzkripcióját adja vissza. Alapértelmezetten hamis.
- Előfordulhat, hogy a felismerés nem sikerül (nem értelmezhető), valamint az API hívása közben hiba történik. Ezeket a kivételeket a felhasználó számára olvasható módon jelenítjük meg (44.-47. sorok).

- **start():** Ez a metódus felelős a beszédfelismerő folyamat elindításáért. Ellenőrzi, hogy a folyamat már el van indítva, és ha nincs akkor elindítja egy új szálban és törli a megállítási eseményt.
- **stop():** Ez a metódus felelős a beszédfelismerő folyamat leállításáért. Ellenőrzi, hogy a folyamat fut, és ha igen beállítja a megállási eseményt és bevárja a szálat.
- **getLatestCommand():** Vár ameddig van feldolgozásra szánt adat, ezt eléri az osztály “data” adattagjából, feldarabolja és kiszűri a kulcsszavakat a felismerés. Visszatéríti a kulcsszavak listáját.

5.1.2. Kapcsolat létrehozása a szimulációs környezettel



12. Ábra ConnectionSetup osztály

Létrehozott attribútumok:

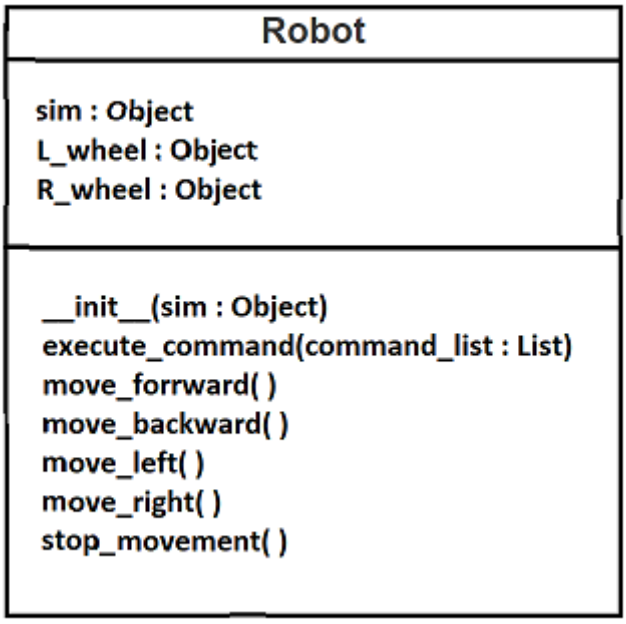
- **client:** Kliens a CoppeliaSim ZMQ Remote API-jához való csatlakozáshoz.
- **sim:** Objektum, amely a szimulációs környezet objektumait tartalmazza.

Metódusok és feladataik:

- **__init__( ):** Az osztály konstruktora, inicializálja az adattagokat.
- **start_simulation( ):** A szimuláció képkockáinak a beállítására használható, valamint elindítja a szimulációt a **sim** objektum **startSimulation()** metódusának segítségével, majd visszatéríti a **sim** objektumot.

A **sim** objektum tartalmazza a szimuláció paramétereit, illetve a vezérlő metódusokat. Ennek segítségével kinyerhető a robot összes alkotó egysége (pl. motorok vagy szenzorok) vagy a színhelyt alkotó elemek.

5.1.3. Robot vezérlő



13. Ábra Robot osztály

Létrehozott attribútumok:

- **sim:** Objektum, amely tartalmazza a szimulációs paramétereket.
- **L_wheel:** Objektum, amely tartalmazza a szimulációs robot bal motorjának a fogantyúját.
- **R_wheel:** Objektum, amely tartalmazza a szimulációs robot jobb motorjának a fogantyúját.

Metódusok és feladataik:

- **__init__():** Az osztály konstruktora, inicializálja az adattagokat.
- **execute_command(command_list : List):** A **command_list** tartalmazza azokat az utasításokat, amelyeket a robot végre kell hajtson.

```
27      def execute_command(self,command_list):
28
29          for command in command_list:
30              if command == 'előre':
31                  self.move_forward()
32              if command == 'hátra':
33                  self.move_backward()
34              if command == 'jobbra':
35                  self.move_right()
36              if command == 'balra':
37                  self.move_left()
38
39              self.stop_movement()
40              pass
41
```

2. Kódrészlet - execute_command() metódus

A fenti ábrán *(2. Kódrészlet - execute_command() metódus)* látható a metódus kódsora. A **command_lis**t-et iterálja, és figyeli, hogy a jelenlegi elem megfelel-e az "előre", "hátra", "jobbra" vagy "balra" utasításoknak. Ha igen, akkor meghívodik a neki megfelelő metódus. A **stop_movement()** metódus arra szolgál, hogy amennyiben a mozgás megtörténik, utána a robot mozgása álljon le.

– **move_forward():**

```
47      def move_forward(self):
48
49          self.sim.setJointTargetVelocity(self.L_wheel,1)
50          self.sim.setJointTargetVelocity(self.R_wheel,1)
51          sleep(1)
```

3. Kódrészlet - move_forward() metódus

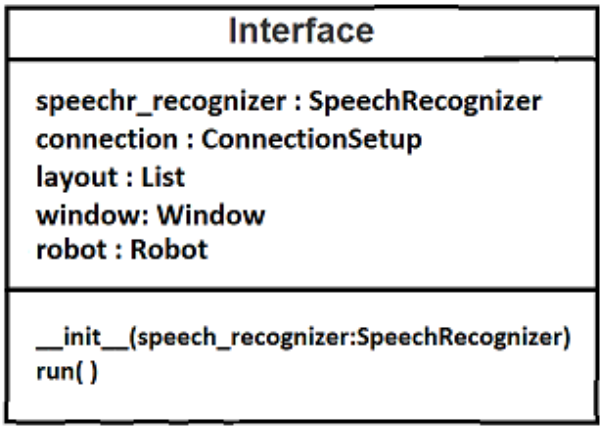
A fenti ábrán (*3. Kódrészlet - move_forward() metódus*) látható a move_forward metódus kódsora. Ebben azt figyelhetjük meg, hogy a szimulációs objektum sebességét hogyan állíthatjuk be a **setJointTargetVelocity** metódus segítségével. Ennek paraméterei a "handle", a mi esetünkben, a jobb, valamint a bal motor és a sebesség. Ha azt szeretnénk, hogy előre mozduljon a robot, akkor a két motor sebességét egyenlő értékre állítjuk be. A metódusban történik egy késleltetés (51. sor) is, hogy a robotnak legyen ideje elmozdulni.

Ennek mintájára a többi mozgást meghatározó metódus a következő értékeket használja a robot vezérlésére:

Metódus	Végrehajtott mozgás	Értékek
move_backward()	**Hátra**	**(-1 , -1)**
move_left()	**Balra**	**(0 , 1)**
move_right()	**Jobbra**	**(1 , 0)**
stop_movement()	**Megáll**	**(0 , 0)**

1. Táblázat Mozgást végrehajtó metódusok

5.1.4. A felhasználói felület



14. Ábra Interface osztály

Létrehozott attribútumok:

- **speech_recognizer : SpeechRecognizer** osztály példánya;
- **connection : ConnectionSetup** osztály példánya;
- **layout :** Lista, amely tartalmazza a felhasználói felület ablakában szereplő elemeket;
- **window :** A layout segítségével készített Window (ablak);
- **robot :** Robot osztály példánya.

Metódusok és feladataik:

- **__init__( ):** Az osztály konstruktora, inicializálja az adattagokat;

– **run( ):** - futtatja a metódust.

```
33      def run(self):
34
35          while True:
36
37              event, _ = self.window.read(timeout=100)
38
39              if event == sg.WINDOW_CLOSED or event == 'EXIT':
40                  self.speech_recognizer.stop()
41                  self.sim.stopSimulation()
42                  break
43              if event == 'SIM-START':
44                  self.sim = self.connection.start_simulation()
45                  self.robot = Robot(self.sim)
46                  self.window.Element('SIM-START').Update(disabled=True)
47                  self.window.Element('SIM-STOP').Update(disabled=False)
48                  self.window.Element('START').Update(disabled=False)
49
50              if event == 'SIM-STOP':
51
52                  self.sim.stopSimulation()
53                  self.window.Element('SIM-START').Update(disabled=False)
54                  self.window.Element('SIM-STOP').Update(disabled=True)
55                  self.window.Element('START').Update(disabled=True)
56                  self.window.Element('STOP').Update(disabled=True)
57
58              if event == 'START':
59                  self.speech_recognizer.start()
60                  self.window.Element('SIM-STOP').Update(disabled=True)
61                  self.window.Element('START').Update(disabled=True)
62                  self.window.Element('STOP').Update(disabled=False)
63
64              if event == 'STOP':
65                  self.speech_recognizer.stop()
66                  self.window.Element('SIM-STOP').Update(disabled=False)
67                  self.window.Element('START').Update(disabled=False)
68                  self.window.Element('STOP').Update(disabled=True)
69
70              if self.speech_recognizer.data_available.isSet():
71                  command=self.speech_recognizer.getLatestCommand()
72                  if command:
73                      self.robot.execute_command(command)
74
75
76          self.window.close()
77
```

*4. Kódrészlet – run() met*ódus

A fenti ábrán () látható a run () metódus, ez felelős a felhasználói felület ablakában létrejött események kezelésére. A következő eseményeket kezeli:

- WINDOW_CLOSED: A felhasználó lezárja az ablakot, a felismerés lezárul és kilép.

- EXIT: A felhasználó a 'Kilépés' gombra kattintott, a felismerés lezárul és kilép.

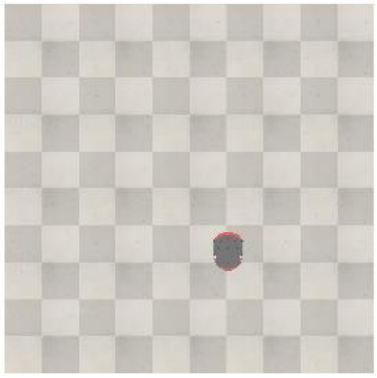
- CON-START: A felhasználó a 'Szimuláció indítása' gombra kattintott és elindítja a szimulációt és inicializáljuk a robotot a szimulációs paraméterrel.

- CON-STOP: A felhasználó a 'Szimuláció leállítása' gombra kattintott és leállítja a szimulációt.

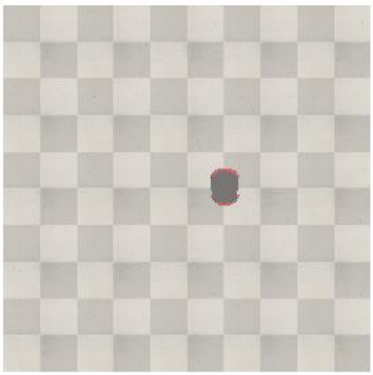
- START: A felhasználó a 'Felvétel indítása' gombra kattintott és elindul a beszédfelismerés.

- STOP: A felhasználó a 'Felvétel leállítása' gombra kattintott és leállítja a beszédfelismerést.

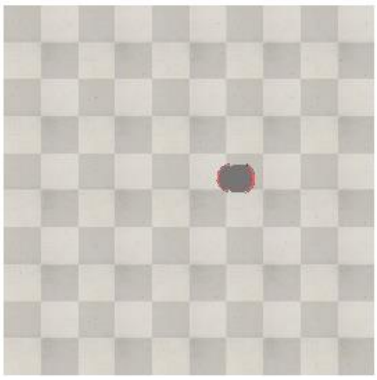
5.1.5. Eredmény



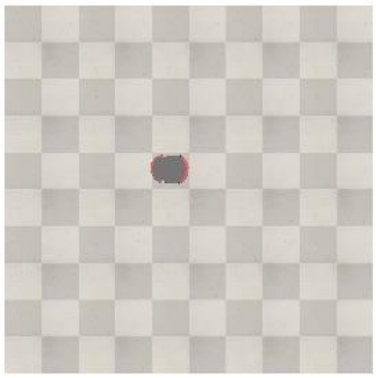
15. Ábra Kezdeti állapot



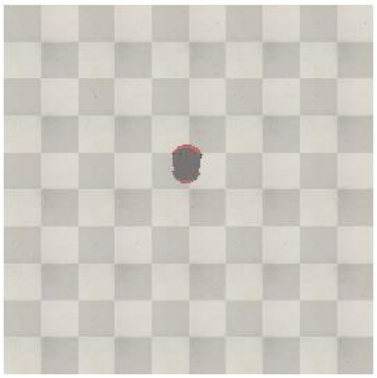
16. Ábra Előre haladás



17. Ábra Jobbra fordulás

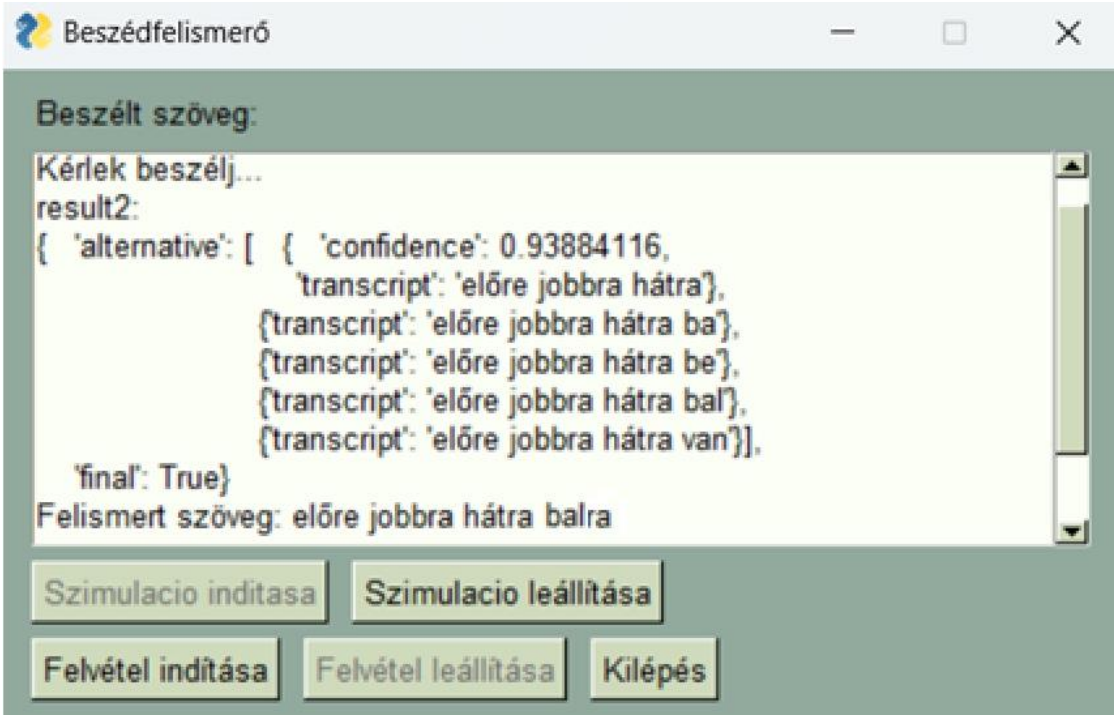


18. Ábra Hátra ment



19. Ábra Balra fordulás

A fenti képsorozat részletezi a robot mozgását a szimulációs környezetben, ha a felhasználó a “előre, jobbra, hátra, balra” utasítás sort adja meg. Az első ábra (*15. Ábra Kezdeti állapot*) bemutatja a robot kezdeti pozícióját a szimuláció elindítása után. A hangutasítások feldolgozása után a robot az adott sorrendben végrehajtja a mozgási műveleteket. Az ábrákon észrevehetjük, ahogy a robot lépéseket tesz előre, jobbra, hátra és balra.



20. Ábra Felismerés eredménye

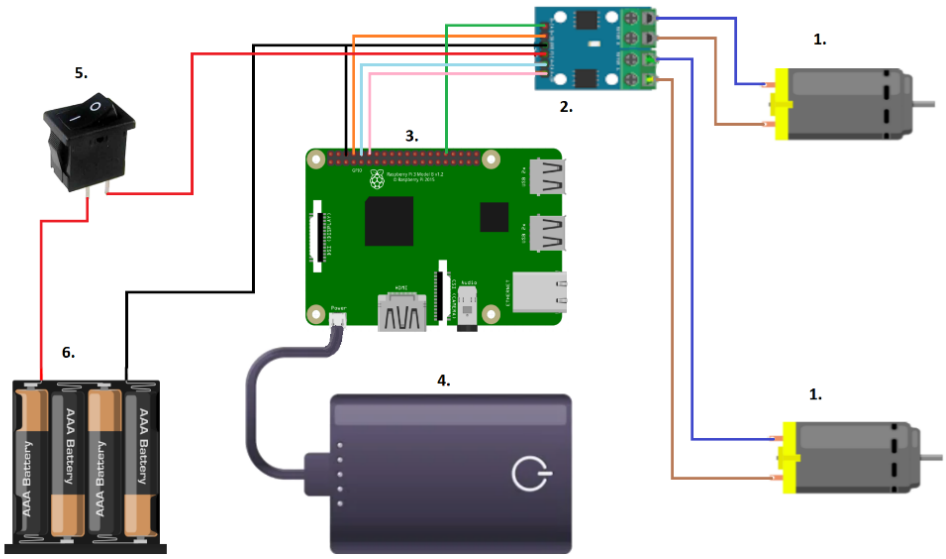
A felhasználói felületen (20. Ábra Felismerés eredménye) látható a felismerés eredménye, valamint a Google Speech to Text API által visszatérített metadata. A metadata tartalmazza az 5 legvalószínűbb átírás eredményét.

A megfelelő működés érdekében a felhasználónak be kell tartania egy sorrendet a felhasználói felület használata közben. Emiatt, azok a funkciók, amelyek bezavarhatnák a megfelelő működést, a felhasználás különböző fázisaiban inaktívak. Ezzel biztosítva, hogy a felhasználó akkor is helyesen használja a felületet, ha nem is ismeri előzetesen a működését.

5.2. Második tervezési fázis

A második tervezési fázis során a fizikai robot megvalósítására fektettem a hangsúlyt, valamint a kommunikációra és a robot vezérlésre.

5.2.1. Robot megvalósítása



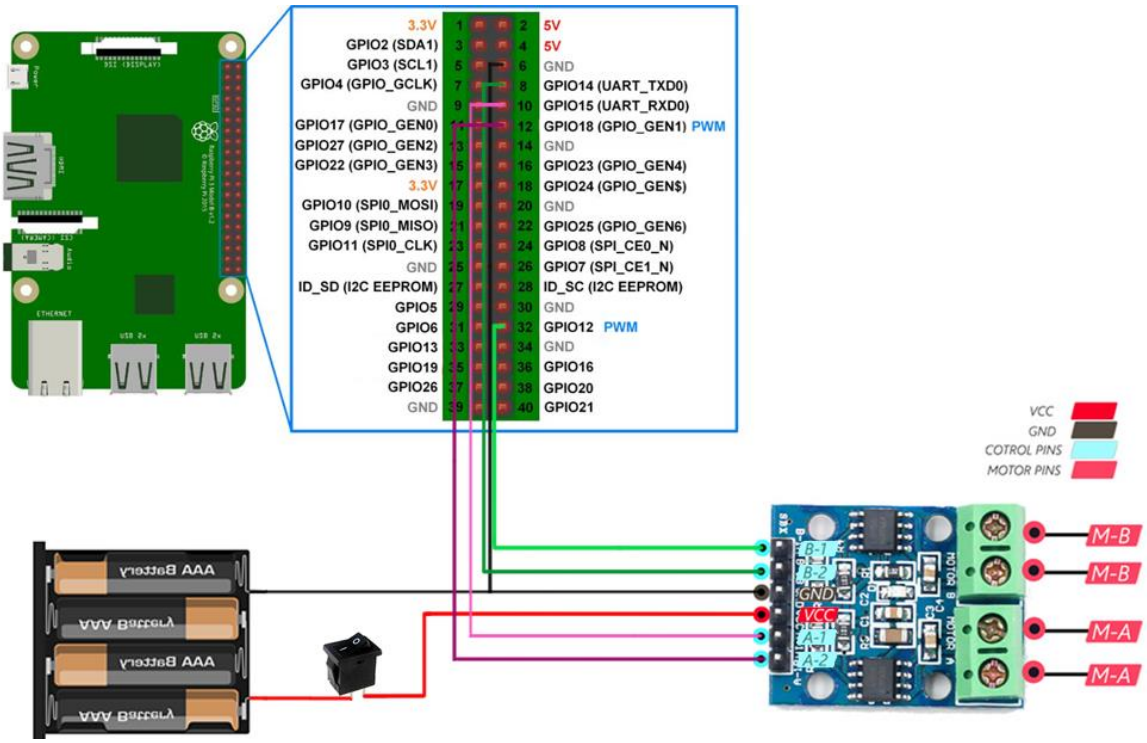
21. Ábra Robot bekötési rajz

A fenti ábrán (*21. Ábra Robot bekötési rajz*) látható a robot bekötési rajza és alkatrészei.

1. Motor (IG220019*00015R - Digilent)
2. H híd (L9110)
3. Raspberry Pi 3
4. Hordozható külső akkumulátor
5. Kapcsoló
6. Elemhordozó + 4 x elem (1.5 V 200mA)

A motorok irányítására PWM jeleket generálunk a Raspberry Pi segítségével. A H híd szerepe a PWM jel polaritásának a megfordítása, ezzel együtt megfordítva a motoroknak a forgási irányát. A két motor esetében szükségünk volt két PWM jel generálására, valamint két iránybit meghatározására. Ezeket állíthatjuk be a Raspberry Pi-on futó program segítségével. A bekötés a következő módon van megvalósítva:

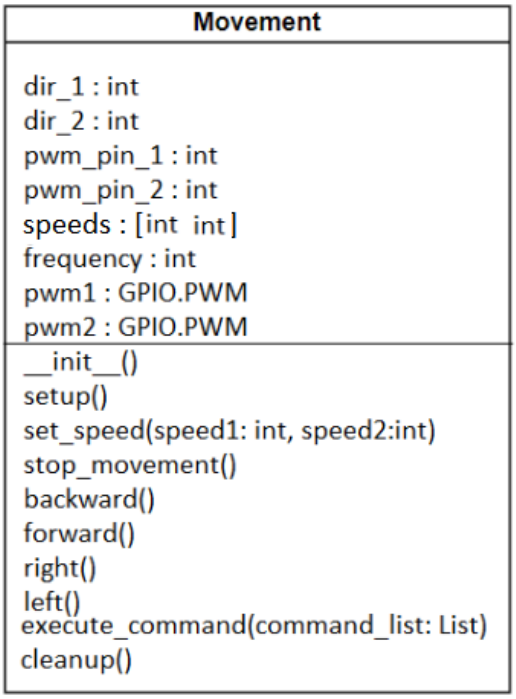
- A motrok pozitív és negatív pólusai a H hídba vannak kötve;
- A Raspberry Pi GPIO pinjei, valamint a GND a H hídba vannak kötve;
- A hordozható külső akkumulátór a Raspberry Pi-t táplálja;
- Az elemhordozó pozitív pólusa egy kapcsolón keresztül, míg a negatív pólusa közvetlenül csatlakoztatva van a H hídhoz;



22. Ábra GPIO pin kiosztás

A fenti ábrán (*22. Ábra GPIO pin kiosztás*) látható, hogy melyek azok a GPIO pinek, amelyek lehetővé teszik a robot vezérlését. A 8-as (GPIO14) valamint a 10-es (GPIO15) számú pinek az iránybitek, a 12-es és 32-es számú pinek a PWM generálására szolgáló pinek. Ezek páronként meghatározzák egy-egy motor vezérlését. A motor táplálása 4 elem által biztosított és egy kapcsoló segítségével megállítható a motorok táplálása. Az elemhordozó és Raspberry Pi földelése (GND) is a H híd földeléséhez van kötve, ezzel biztosítva a földelési potenciál egységességét.

5.2.2. Robot vezérlése



23. Ábra Movement osztály

A Movement osztály felelős a Raspberry Pi alapú robot irányításáért.

Létrehozott attribútumok:

- dir_1, dir_2: Meghatározzák az iránybiteket a Raspberry Pi pin száma szerint

- pwm_pin1, pwm_pin2: Azok a pinek, amelyeken generálni szeretnénk a PWM jeleket

- speeds: A két motor sebességét tárolja

- frequency: A PWM jel frekvenciája, ez egy állandó érték.

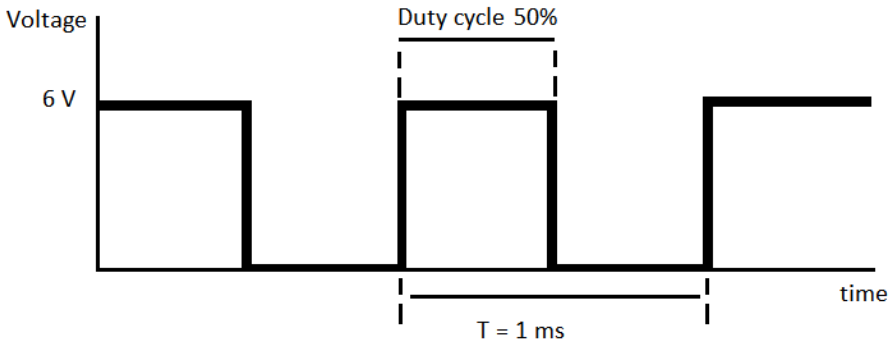
- pwm1, pwm2: Generált PWM jelek

Metódusok:

__init__(): Osztály konstruktora, inicializálja a pinek értékét, valamint a frekvenciát.

setup(): Beállítja a pinek irányát kimenetre (OUT) és generálja a PWM jeleket kikapcsolt állapotban.

set_speed(speed1, speed2): Metódus, amely a paraméterként megkapott értékre állítja a PWM jeleknek a kitöltési tényezőjét (duty cycle) (0-100 között), ezáltal meghatározza a bekapcsolt állapotban töltött időtartamot.



24. Ábra PWM jel kitöltési tényezője (duty cycle)

A PWM jel a motor forgási sebességét határozza meg. A duty cycle a periódus idejének azt a részét jelzi, amikor a PWM jel HIGH állapotban van. A motor forgási sebessége arányos a duty cycle értékével, amennyivel nagyobb ez az érték, annál nagyobb a forgási sebesség, és ez fordítva is igaz. A fenti ábrán (*24. Ábra PWM jel kitöltési tényezője (duty cycle)*) látható egy PWM jel, aminek a duty cycle értéke 50%-ra van állítva. Ez azt jelenti, hogy a motor a teljes feszültség felét kapja meg, ami 3 V ideális esetben.

stop_movement(), forward(), backward(), left(), right(): Ezek a metódusok határozzák meg a mozgást.

Iránybitek (dir_1, dir_2)	Duty cycle (pwm1, pwm2)	Mozgás
LOW, LOW	0%, 0%	Megállás
HIGH, HIGH	20%, 20%	Előre
LOW, LOW	90%, 90%	Hátra
LOW, HIGH	95%, 5%	Jobbra
HIGH, LOW	5%, 95%	Balra

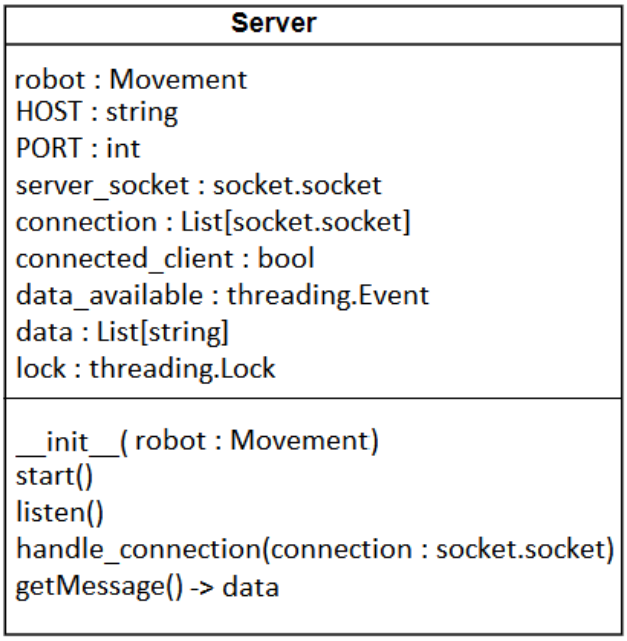
2. Táblázat Mozgási értékek

A fenti táblázatban **(**2. Táblázat Mozgási értékek**)** vannak összefoglalva a duty cycle értékek, valamint az iránybitek állapota, amelyek szerint végrehajtódik a mozgás.

execute_command(command_list): Ellenőrzi az utasítás lista tartalmát és annak megfelelően meghívja a mozgást megvalósító metódusokat.

cleanup(): Az erőforrások felszabadítására szolgál. A program végén leállítja a PWM jelek generálását, és felszabadítja a GPIO-kat.

5.2.3. Szerver megvalósítása



25. Ábra Server osztály

Létrehozott attribútumok:

- robot: Movement osztály objektuma, tartalmazza a robot vezérléséhez szükséges metódusokat
- HOST: Szerver IP címe
- PORT: Kapcsolódási port száma
- server_socket: Socket objektum a szerver oldali kommunikációhoz.
- connection:Lista a kapcsolatok tárolására.
- connected_client: Logikai változó, amely jelzi, hogy van-e aktív kapcsolat.
- data_available: Esemény objektum, amely jelzi, hogy elérhető az adat.

- data: A legutóbb kapott üzenetet tartalmazó változó.
- lock: Zár objektum, a szál biztonság érdekében a kliens tárolásánál használjuk.

Metódusok:

start(): Elindítja a szerver működését. Egy másik szálat indít el a listen metódussal, amely figyeli a kliensek csatlakozását. Egy végtelen ciklusban várja, hogy elérhető legyen az utasítás és továbbítja ezt a robot objektum execute_command() metódusának. KeyboardInterrupt esetén felszabadítja az erőforrásokat és lezárja a szerver_socketet.

listen(): Ez a metódus kezeli a kliensek csatlakozását.

```
75          self.server_socket.listen(1)
76          print('Várakozás a csatlakozásra...')
77          while True:
78              if not self.connected_client:
79               connection, client_address = self.server_socket.accept()
80               print('Csatlakozott kliens:', client_address)
81
82               self.connected_client=True
83               connection_thread = threading.Thread(target = self.handle_connection,args=(connection,))
84               connection_thread.start()
85
86               with self.lock:
87                   self.connection.append(connection)
```

5. Kódrészlet listen metódus

Beállítjuk a szerver socketet, hogy figyelje a kapcsolatokat, a paraméternek megadott szám jelzi, hogy hány aktív kapcsolatot engedélyezünk. A self.server_socket.accept() blokkolja a ciklust, ami azt jelenti, hogy ameddig nincs ú kliens, addig a program megáll ezen a soron, és várakozik. Ha van csatlakozott kliens, akkor a paramétereit átadjuk a handle_connection metódusnak, amit egy új szálban futtatunk. A végén mentjük a kapcsolatot a connection listába.

handle_connection(connection):

```
96          with connection:
97              while True:
98
99                  data = connection.recv(1024)
100                 if data:
101
102                     self.data = pickle.loads(data)
103                     self.data_available.set()
104
105                 else:
106
107                     break
108
109
110             with self.lock:
111                 self.connection.remove(connection)
112                 self.connected_client=False
113                 print('Kapcsolat lezárva')
114
```

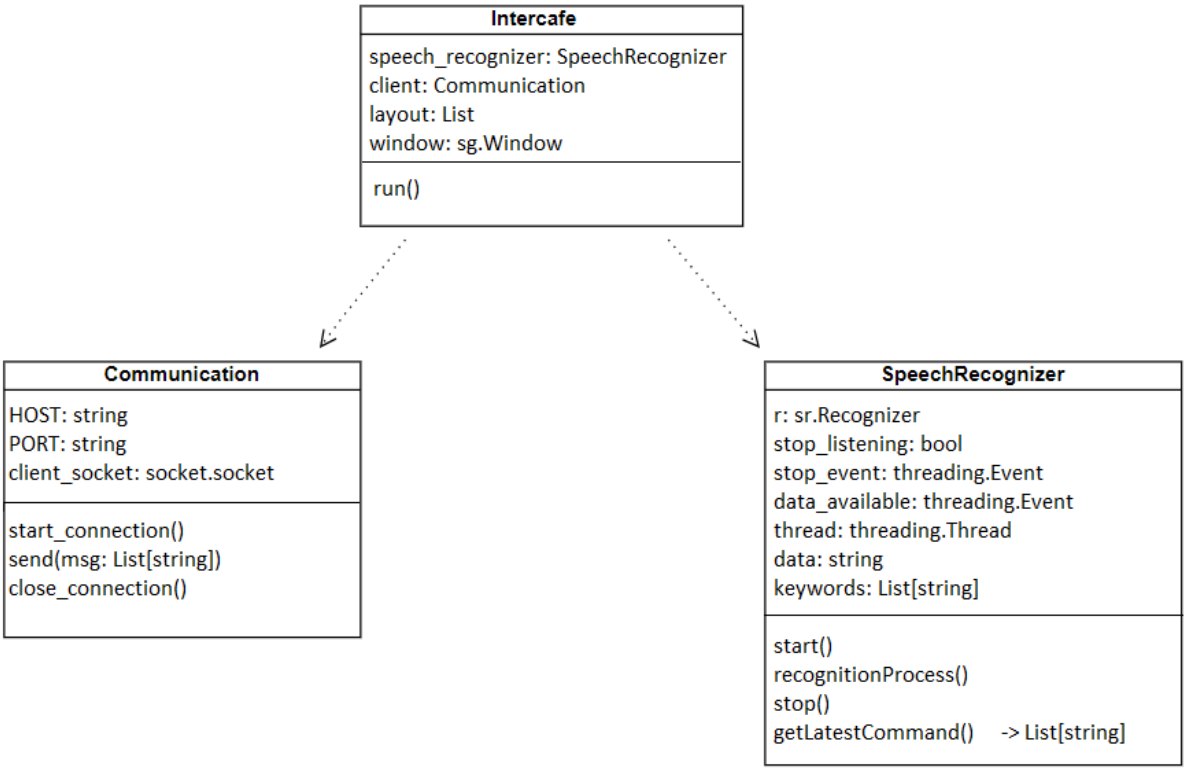
6. Kódrészlet handle_connection metódus

A “with” blokk a connection objektumot kezeli, ennek előnye, hogy gondoskodik az erőforrás felszabadításáról amikor a blokk befejeződött. A ciklusban folyamatosan várja a kliens üzenetét. Ha az üzenet megérkezett, akkor ezt a pickle.loads() metódus szerint deszerializáljuk, mentjük majd beállítjuk a data_available eseményt. Ha nem érkezett üzenet, vagy a kapcsolat megszakadt, akkor kilépünk a ciklusból. A kritikus szekcióban töröljük a kapcsolatot a listából és jelezzük, hogy nincs aktív kliens.

getMessage(): A metódus várakozik a kliens üzenetére és blokkolja a programot. Ha az adat megérkezett az üzenetet kimentjük a data változóba, töröljük az event értékét és visszatérítjük az üzenetet.

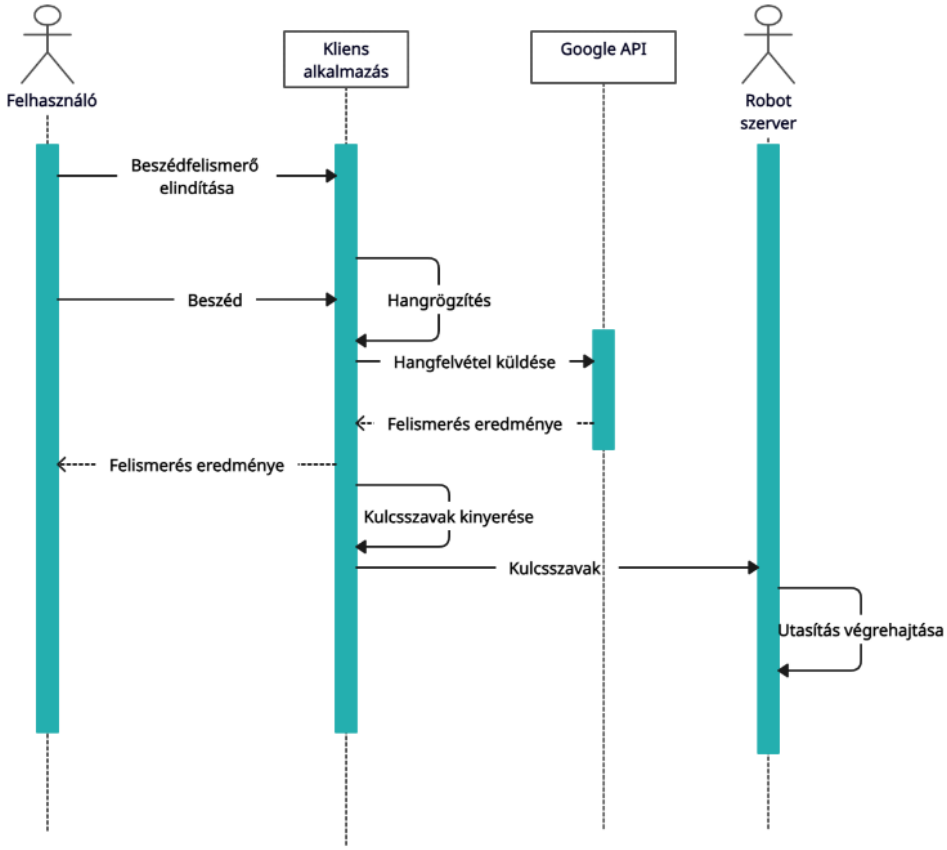
5.2.4. Kliens alkalmazás

Az első tervezési fázishoz képest, itt a robot vezérlése már a szerver oldalon történik, emiatt csak az adatok feldolgozására és elküldésére kell fokuszálni. Ebben az esetben a kommunikáció megvalósítása a Communication osztály attribútumain és metódusain keresztül történik.



26. Ábra kliens alkalmazás osztály diagram

A fenti ábrán (26. Ábra kliens alkalmazás osztály diagram) látható a kliens oldalon használt osztályok. Az Interface osztály kezeli a kommunikációért és a hangfeldolgozásért felelős osztályok objektumait. Funkcionalitás szempontjából csak a kommunikációért felelős osztály változott az előző megvalósításhoz képest. A kliens oldalon a kommunikáció a szerverhez való csatlakozásból és adat küldésből áll. Az üzenetet a felhasználói felületen keresztül kapja meg, amikor van elérhető adat a SpeechRecognition osztály feldolgozó egységétől. Ez az üzenet egy lista, amit szerializálni kell annak érdekében, hogy megfelelő formátumba jelenjen meg a szerver oldalán.



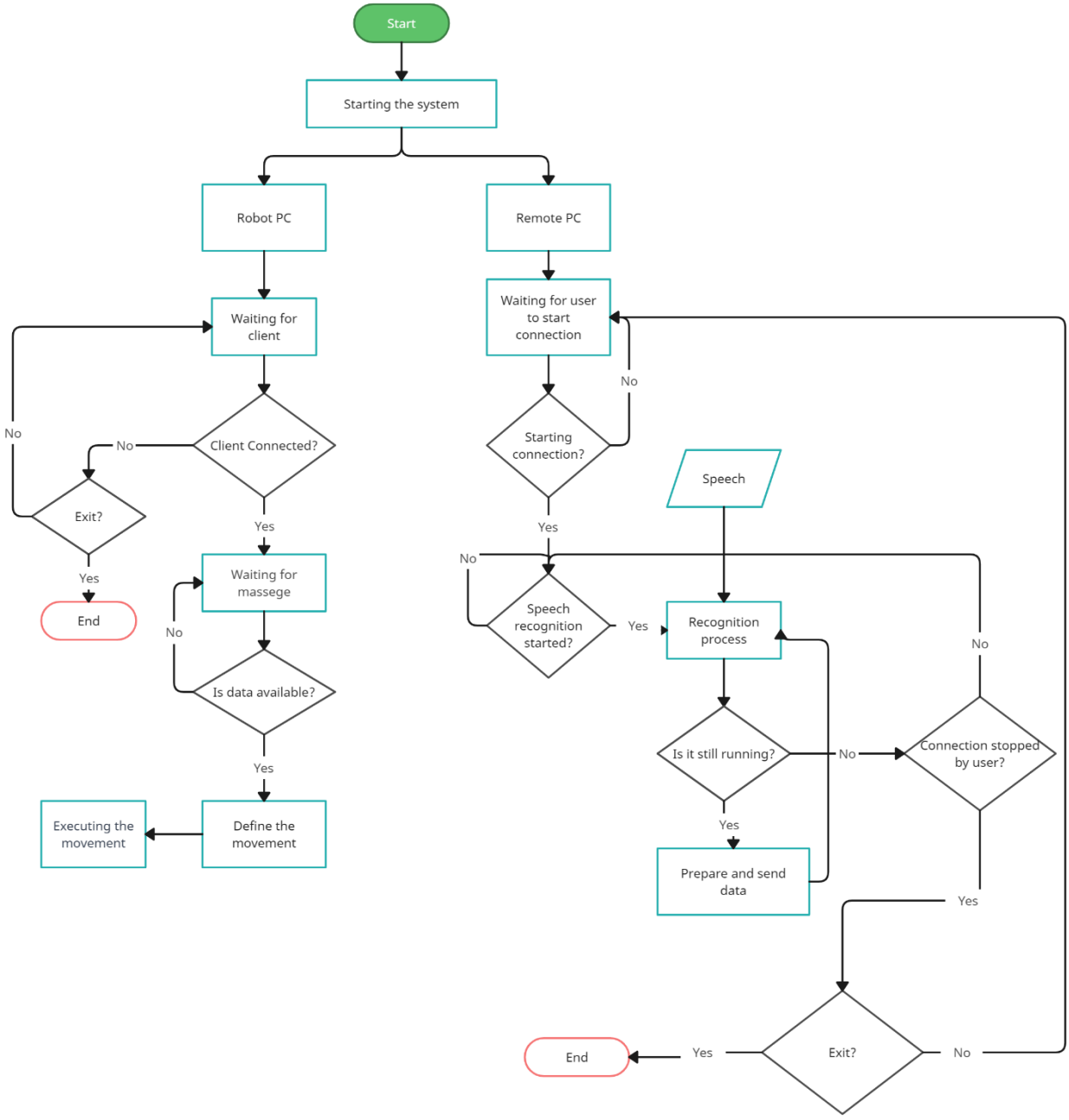
27. Ábra Kliens alkalmazás szekvencia diagramja

A fenti ábrán (27. Ábra Kliens alkalmazás szekvencia diagramja) látható szekvencia diagram ábrázolja a felhasználó és a rendszer közötti műveleteket és adatfolyamot. A felhasználó a rendszer műveleteinek a végrehajtási eredményeit a felhasználói felületen keresztül látja. Ugyancsak a felhasználói felületen keresztül interakcióba léphet a rendszerrel és minden egyes utasítása során más állapotba kerül a rendszer. Fontos megjegyezni, hogy bizonyos események csak akkor fognak bekövetkezni, ha a felhasználó erre utasítja a rendszert.

Első sorban a felhasználó elindítja a kapcsolódási folyamatot a felhasználói felületen keresztül. Ha sikeres a kapcsolódás akkor el lehet indítani a beszéd felismerését is. A beszéd a kliens alkalmazásban kerül rögzítésre, majd a Google Speech to Text API segítségével megkapjuk a felismerés eredményét. A felismerés eredménye a felhasználó számára is látható.

A felismerés eredményéből kinyerődnek azok a kulcsszavak, amelyek azonosítják a mozgásokat. Ezeket a kulcsszavakat továbbítjuk a robot szerverének, majd ezek alapján azonosítok és megtörténik a mozgás.

5.2.5. A rendszer működése



28. Ábra Rendszer folyamat ábra

A fenti ábra (*28. Ábra Rendszer folyamat ábra*) magába foglalja a teljes rendszer lépéseit és döntéseit. Látható, hogy a teljes rendszer két részből áll, a robot és a távoli számítógép.

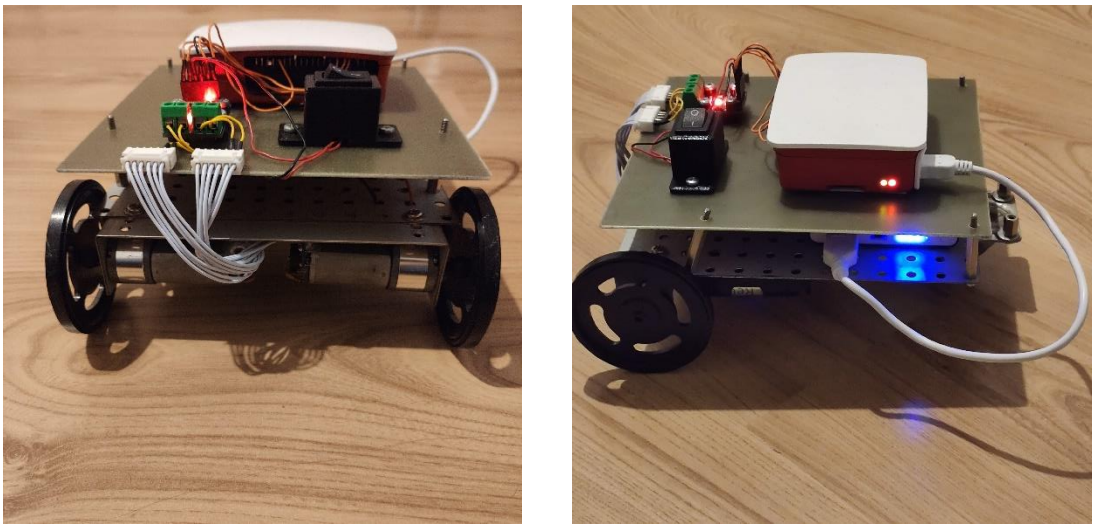
Robot PC folyamatok:

- Várakozik a kliens csatlakozására;
- Ha nincs kliens csatlakozva akkor megállítható a szerver;
- Ha van kliens csatlakozva akkor várakozik az üzenetre;
- Ha van elérhető üzenet, akkor azonosítsa a mozgási műveletet/műveleteket és végrehajtsa; a motorok vezérlését az utasítások függvényében;
- Várja a következő üzenetet.

Távoli számítógép folyamatok:

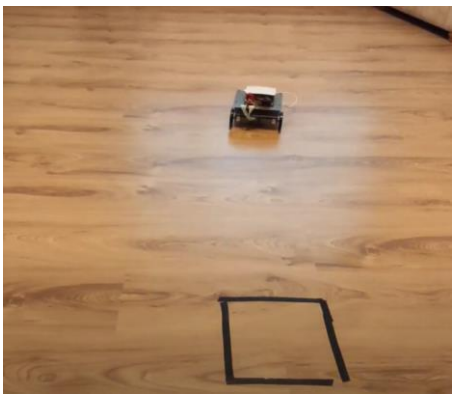
- Várakozik, ameddig a felhasználó elindítja a kapcsolatot;
- Ha sikeres csatlakozás van, akkor várakozik ameddig a felhasználó elindítja a beszédfelismerőt;
- Ha a beszédfelismerő el van indítva, akkor elindul a felismerési folyamat, amelynek bemente a felhasználó beszéde;
- A felismerési folyamatban beletartozik az adatnak a feldolgozása: ha nem egy utasítást vagy utasítás sorozat akkor nem releváns és nem küldjük el a robot félnek;
- Ameddig a felhasználó nem állítja le a beszédfelismerőt addig folyamatosan fogadja a hangparancsokat, feldolgozza, és továbbítja végrehajtásra;
- Ha a beszédfelismerés megáll akkor két lehetséges eset van:
    - a felhasználó újra indítsa a beszédfelismerést, ebben az esetben újra elindul a beszédfelismerési folyamat;
    - a felhasználó leállítsa a kommunikációt;
- A kommunikáció leállítása esetében a felhasználó kiléphet és ezzel véget ér a folyamat, ellenkező esetben újra elindíthatja a kommunikációs csatornát és folytathatja a robot irányítását.

5.2.6. Eredmény

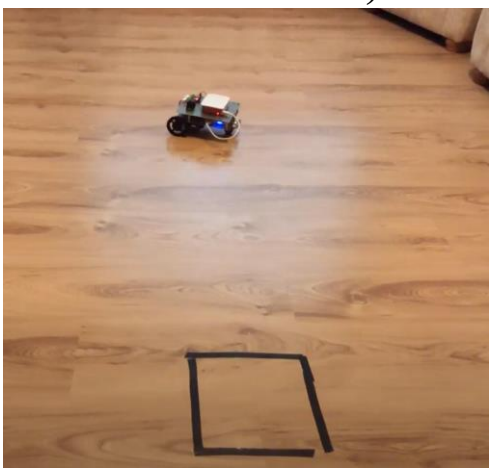


29. Ábra Raspberry Pi robot

a.)



b.)



c.)

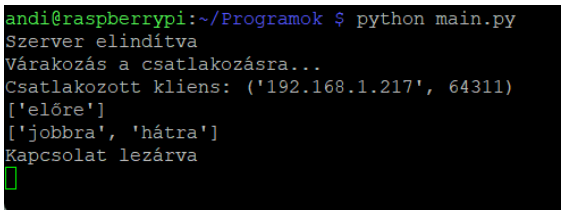
d.)

3. Táblázat Prototípus működése

A fenti táblázat (3. Táblázat Prototípus működése) foglalja össze a prototípus működését.

a) Az ábrán a rendszer inicializálva van, és várja a felhasználó utasítását;
b) A felhasználó kiadta a “Menj előre” parancsot és a robot elmozdul előre;
c) A felhasználó kiadta a “Fordulj jobbra és menj hátra” parancsokat és a robot elfordult jobbra;
d) A jobbra fordulás után végrehajtsa a hátrafele menetet.

30. Ábra Felhasználói felületen kijelzett eredmények



31. Ábra Raspberry Pi által kijelzett eredmények

6. Üzembe helyezési lépések

6.1. Szimulációs robot vezérlése

1. Lépés - Kódbázis

A kódbázis elérhető GitHub-on ezen a [1]linken. A kódbázist vagy klónozni vagy .zip formátumban le kell tölteni. A kódbázisban található a requirements.txt, ami tartalmazza a szükséges csomagokat. A telepítést ajánlott egy virtual enviorment alatt végezni, ezzel elkerülhető a csomag vagy verzió ütközés.

A következő lépések Windows alatt érvényesek:

- Ha nincs telepítve a virtualenv csomag, akkor a következő utasítás segítségével telepíthető:

  pip install virtualenv

- Egy terminál segítségével a gyökérkönyvtárban a következő utasításokkal lehet létrehozni egy vitruális környezetet és a megfelelő könyvtárakat telepíteni:

  virtualenv venv
  venv\Scripts\activate
  pip install -r requirements.txt

2.Lépés - CoppeliaSim beüzemelése

- Telepíteni kell a CoppeliaSim szimulációs környezetet, ezt a [2]hivatalos weboldalról lehet megtenni.
- Elindításkor, a CoppeliaSim betölti a ZMQ pluginokat, ha ez nem történik meg akkor látogass el [3]erre a weboldalra.
- A kódbázisban található proba.ttt elnevezésű állományt be kell tölteni a következő módon:

[1] https://github.com/AndreaKatona/Szimulacios-robot-vezerlese.git

[2] https://www.coppeliarobotics.com/

[3] https://github.com/CoppeliaRobotics/simZMQ

File > Open scene.. > proba.ttt

- Hozzá kell adni a megfelelő elérési útvonalat a python kódba, pontosabban robot.py, valamint a connectipn.py állományokban a következő módon, ahol a "…" jelöli a CoppeliaSim telepítési helyét:

  sys.path.append(...\CoppeliaSimEdu\programming\zmqRemoteApi\clients\python)

3. Lépés - Program futtatása

A program futtatására a virtual enviorment aktiválva kell legyen, ezt megtehető manuálisan vagy egy fejlesztői környezetben beállítható, hogy ez automatikusan elinduljon. Ezt a [4]Visual Studio Code segítségével a következő módon lehet megtenni:

- Visual Studio Codeban a gyökérkönytárat nyissuk meg.
- Amikor létező virtual enviormentet szeretnénk használni, akkor a Visual Studio Code érzékeli és ajánlani fogja, hogy ezt használd munkakörnyezetnek. Ha ez nem történik meg akkor manuálisan beállítható a [5]dokumentáció szerint.

Ha a virtual enviorment aktiválva van, akkor a main.py fájl futtatásával indítható el a program.

6.2. A fizikai robot vezérlése

1. Lépés – Kódbázis

A kódbázis elérhető [6]ezen a linken. A kódbázis két részből áll, az egyik a robot felén fog futni, a másik pedig a kliens számítógépén. A kliens számítógépén az alkalmazás telepítése hasonló módon történik, mint az előző fejezet első lépése ().

A robothoz tartozó kódbázist a Raspberry Pi 3-ra kell letölteni. A Raspberry Pi OS biztosit fejlesztői környezeteket is, ezek közül használható a Thonny is, vagy más fejlesztői környezetet is lehet használni. Ha a Raspberry Pi operációs rendszere egy régebbi verzió, ajánlatos ezt egy frissebb verzióra változtatni:

[4] https://code.visualstudio.com/

[5] https://code.visualstudio.com/docs/python/environments

[6] https://github.com/AndreaKatona/Raspberry-Pi-robot-vezerlese.git

A Raspberry Pi-ra le kell tölteni “Raspberry” mappát a kódbázisból. A connection.py fogja tartalmazni a szerver futtatásához szükséges adatokat. Itt a “HOST” -hoz tartozó IP címet be kell állítani a Raspberry Pi-nak megfelelő IP címnek. Ezt a következő módon kaphatod meg:

- Egy terminál segítségével írjuk be a következő parancsot: ifconfig. Ezután látni fogjuk az összes hálózati konfigurációt, valamint az IP címet is
- Az IP címet a connection.py HOST=’...’ sorába kell megadni, ezzel jelezve a szerver

Ha ezt megtetted, akkor a számítógépeden is a kódbázisban meg kell add a szerver IP címét, ez a Rasperry IP címe lesz. Nyisd meg a **communication.py** fájlt és cseréld ki a HOST=’...’ sorban az IP címet.

2. Lépés - Robot

- Robot alkatrészeinek a beszerzése (6. Táblázat Alkatrész lista)
- Robot alkatrészeinek az összekötése (21. Ábra Robot bekötési rajz)

3. Lépés - Szerver elindítása

A szerver elindítása a számítógépen történik és erre szükségünk van a [7]PuTTY szoftverre. Az előző lépésben megszerzett IP címet a PuTTY ablakának a Host Name helyére beillesszük és a SSH (Secure Shell) kapcsolódást kiválasszuk (Ha ezt nincs engedélyezve a Raspberry-n akkor: Preferences > Raspberry Pi Configuration > Interfaces > SSH -Enable).
A megnyitás után meg fog jelenni egy terminál, ebben a következő műveleteket hajtjuk végre:

- Bejelentkezés a Raspberry Pi felhasználóba (felhasználónév + jelszó);
- Elnavigálunk a gyökérkönyvtárba (cd PéldaNév);
- Futtassuk a main.py-t: python main.py utasítással.

[7] https://www.putty.org/

6.3. Felmerült problémák és megoldásaik

A tervezés során számos érdekes kihívással találkoztam, amelyeket sikerült megoldanom a prototípusok kialakítása során.

Az egyik fő probléma, amivel szembesültem, az a beszédfelismerő által okozott felhasználói felület blokkolás volt. A mikrofon meghatározatlan ideig való hallgatása gyakran blokkolta a felimerési folyamatot. Emiatt úgy döntöttem, hogy 10 másodpercig várok a felhasználóra, hogy beszéljen és ha ez idő alatt nem történik meg az utasítás átadása, akkor kivétel keletkezik és várakozik a következő utasításra.

Egy másik probléma, amit meg kellett oldanom, az a Google Speech to Text API válaszának formátuma volt. Az API-val való kísérleteim során észrevettem, hogy a válaszként kapott üzenetsor nagy és kisbetűkből is állhatott, de ez nem mindig volt következetes. Hogy elkerüljem a kiszűrés során a nagy és kisbetű ellenőrzést is, úgy döntöttem, hogy minden üzenetet előbb átalakítok kisbetűs formába.

Egy harmadik probléma az volt, hogy a felismerés eredménye nem mindig helyes vagy nem része az utasítás sorozatnak. Ez abban az esetben jelentett volna problémákat, ha a robot minden utasítást addig hajtana végre ameddig nem kap egy másikat. Annak érdekében, hogy a mozgás pontosabb legyen, minden utasítást végén a robot megáll majd folytatja a következő utasítás végrehajtását.

Ezekkel a problémákkal sikerült megbirkóznom a prototípusok tervezése során. Folyamatos fejlesztések és finomítások során remélhetőleg egy még jobb és felhasználói felületet tudok kialakítani a jövőben.

7. Mérések és következtetések

A rendszerrel elvégzett kísérletek során két fő szempontot vizsgáltunk. Az egyik az volt, hogy a zajos környezet milyen mértékben zavarja be a felismerés folyamatát, és hogy segíthet a mikrofon érzékenységének a csökkentése ebben az esetben. A másik fontos tényező, amit megfigyeltünk, hogy a rendszer reakció ideje milyen mértékben függ a környezeti zajoktól és hogy mennyi az átlagos reakció idő. Az az alábbi (4. Táblázat Mérési táblázat) táblázatban vannak összefoglalva a felismerés mérési eredmények. A mérések során a zajmentes környezetben nem voltak a rendszert bezavaró hangok, a felhasználó volt az egytelen hangforrás. A zajos környezetben a felhasználótól megközelítőleg fél méterre helyeztünk egy hangforrást, ezen magyar nyelvű hangrögzítéseket játszódtunk le. Ugyanezt a környezetet biztosítottuk a harmadik típusú mérés esetén, viszont itt a mikrofon érzékenységét csökkentettül 20%-kal.

Elhangzott utasítás	**Zajmentes környezetben**			**Zajos környezetben**			**Zajos környezetben, érzékenység csökkentésével**		
	Elhangzott	Felismerve		Elhangzott	Felismerve		Elhangzott	Felismerve	
“előre”	50	47	94%	50	49	98%	50	49	98%
“hátra”	50	48	96%	50	47	94%	50	44	88%
“jobbra”	50	49	98%	50	49	98%	50	47	94%
“balra”	50	48	96%	50	45	90%	50	42	84%

4. Táblázat Mérési táblázat

A mérésbe csak azokat az adatokat vettük figyelembe, amelyek a beszédfelismerő feldolgozási keretein belül voltak megadva. Ha a beszédfelismerő adatot dolgozott fel, akkor nem adtunk ki más utasítást ameddig azt be nem fejezte. A parancsokat egyenként adtuk a rendszernek, minden parancs kimondása nem haladhatta meg a 6 másodpercet. A mérések során csak azokat az eredményeket figyeltük, amikor a felhasználó adott ki parancsokat. Ha a rendszer nem érzékelte vagy a felismerés eredménye nem egyezett a felhasználó által beszélt szöveggel akkor a felismerést sikertelennek minősítettük.

Elhangzott utasítás	Átlag reakció idő		
	Zajmentes környezetben	Zajos környezetben	Zajos környezetben, érzékenység csökkentésével
“előre”	3.18 s	4.23 s	4.53 s
“hátra”	3.19 s	4.05 s	4.83 s
“jobbra”	3.36 s	4.54 s	4.47 s
“balra”	3.65 s	4.40 s	4.85 s
Átlag	3.35 s	4.31 s	4.67 s

5. Táblázat Átlag reakció idő

A mérések során figyeltük a rendszer reakció idejét is. A reakció időt úgy határoztuk meg, mint a hangparancs kiadásának pillanatától a robot mozgásának kezdetéig eltelt idő. Megfigyeltük, hogy a zaj mennyire befolyásolja a teljes rendszer működésének gyorsaságát. Annak érdekében, hogy az eredmény pontosabb legyen, minden esetben ugyanannyi értéket vettünk (összesen 600).

Következtetések:

- Bár azt gondolnánk, hogy a mikrofon érzékenységének csökkentése segítene a felismerési folyamaton zajos környezetben az eredmények az ellenkezőjét mutatják. Nem csak a felismert adatok száma csökkent, hanem a rendszer reakció ideje is megnövekedett.
- A zajos környezetben elért eredmények azt mutatják, hogy a rendszer működőképes zajos körülmények között is.
- Annak ellenére, hogy a mérések 2/3-ad része nem kedvező körülmények között történt, a rendszer reakció képessége átlagosan 4.11 századmásodperc.
- A Google Speech to Text által nyújtott beszédfelismerő rendszer valóban jó eredményeket szolgált, emiatt levonhatjuk azt a következtetést is, hogy nem szükséges saját beszédfelismerő rendszert tanítani. Fontos megjegyezni, hogy abban az esetben, ha gyorsabb rendszert szeretnénk megvalósítani vagy nem szeretnénk fizetni a szolgáltatásért, akkor szükséges saját beszédfelismerő rendszert készíteni.

8. Összegzés

A dolgozatom során sikerült két prototípust készíteni, amelyek robotok hangvezérlését teszik lehetővé. Az egyik rendszer egy szimulációs környezetben létező robot vezérlését valósítsa meg. Ez remek lehetőséget biztosít további kísérletekre, mivel nem annyira költséges, mint egy valódi robot elkészítése és üzemeltetése. A második prototípus egy valódi robot vezérlését megvalósító rendszer, amely egy Raspberry Pi alapú robot hangvezérlését teszi lehetővé.

A két rendszer közös vonásai közé tartozik a Google Speech to Text API használata, amelynek segítségével a beszédből kinyert információk szöveges megjelenítését kaptuk meg. A beszédből kinyert információk alapján sikerült azonosítani azokat a kulcsszavakat, amelyeknek segítségével vezérelhetjük a robotot.

Továbbá sikerült megvalósítani mind a két rendszer esetében egy felhasználói felületet, aminek segítségével a felhasználó irányíthassa a rendszert, valamint visszajelzést kap a rendszer állapotáról és a felismerés eredményéről.

Összességében sikerült két olyan prototípust fejleszteni, amelyeknek segítségével sok izgalmas felhasználása lehet a jövőre nézve.

8.1. Továbbfejlesztési lehetőségek

A továbbfejlesztési lehetőségek sok izgalmas alkalmazást biztosíthatnak a rendszer számára. Az első továbbfejlesztési lehetőség lehet a robotnak a távoli irányítása, bárhonnan a világban. Ebben az esetben egy kamera segítségével megfigyelhető a robot helyzete. Ez egy remek lehetőséget jelenthet különböző veszélyes helyzetek megfigyelésére.

Egy másik továbbfejlesztési lehetőséget jelenhet az autonóm mozgás beüzemeltetése. Ebben a rendszerben a robot képes lenne saját maga mozogni a környezetében, a hangirányítás kizárólag beavatkozásra szolgálna.

Egy harmadik továbbfejlesztési lehetőséget jelent a távolságmérés. Ennek segítségével a robot pontos távolságokat tudna bejárni, (pl. “előre 5 métert” esetében a robot 5 métert megy előre). Így a robot irányítása pontosabb lenne és akadályok kikerülésére is remek megoldást jelenthet.

A negyedik továbbfejlesztési lehetőség a paraméterezhetőség. Be lehet vezetni a rendszerbe olyan parancsokat, amelyek a mozgási paramétereket határozzák meg. Ha azt

szeretnénk, hogy a robotunk gyorsabban vagy lassabban hajtsa végre a mozgásokat, valamint, hogy a kanyarokat élesebben vegye vagy visszaforduljon helyben, akkor csak a megfelelő kulcsszavakat adjuk meg.

Összességében a rendszer továbbfejleszthető és sok új alkalmazási területe lehet minden egyes fejlesztett prototípusnak.

9. Függelék

Alkatrész	Ár
2 X motor	~ 145 RON x2
2 X Kerék	~ 6 RON x2
Pivot kerék	~ 6 RON
H híd	~ 7 RON
Raspberry Pi 3	~ 155 RON
Hordozható külső akkumulátor + mikro USB	~ 100 RON
Kapcsoló	~ 1.50 RON
Elemhordozó + 4 x elem (1.5 V 200mA)	~ 10 RON
Egy alap (fém vagy műanyag)	~ 0 – 55 RON

6. Táblázat Alkatrész lista

10. Hivatkozások

[1] Yu, D., & Deng, L. (2016). *Automatic speech recognition* (Vol. 1). Berlin: Springer

[2] Isaacs, D. (2010). *A comparison of the network speech recognition and distributed speech recognition systems and their effect on speech enabling mobile devices* (Master's thesis, University of Cape Town)

[3] Këpuska, V., & Bohouta, G. (2017). Comparing speech recognition systems (Microsoft API, Google API and CMU Sphinx). *Int. J. Eng. Res. Appl*, *7*(03), 20-24.

[4] Google Cloud Speech-to-Text. (2023, május 30). Speech-to-Text API Documentation. Elérhető: https://cloud.google.com/speech-to-text/docs/speech-to-text-requests

[5] Hockstein, N. G., Gourin, C. G., Faust, R. A., & Terris, D. J. (2007). A history of robots: from science fiction to surgical robotics. Journal of robotic surgery, 1, 113-118.

[6] Wikipedia contributors. (2023, May 31). Unimate. In *Wikipedia, The Free Encyclopedia*. Elérhető: https://en.wikipedia.org/w/index.php?title=Unimate&oldid=1157826703

[7] BME weboldala Elérhető: https://mogi.bme.hu/TAMOP/robotmechanizmusok/ch14.html

[8] Wikipedia contributors. (2022, december 4). Mobile robot. In Wikipedia, The Free Encyclopedia.Elérhető:https://en.wikipedia.org/w/index.php?title=Mobile_robot&oldid=1125548051

[9] Celen, I. H., Onler, E., & Kiliç, E. (2015, január). A Design of an Autonomous Agricultural Robot to Navigate between Rows

[10] Rahat, S. A., Imteaj, A., & Rahman, T. (2018, október). An IoT based interactive speech recognizable robot with distance control using Raspberry Pi. In 2018 International Conference on Innovations in Science, Engineering and Technology (ICISET) (pp. 480-485). IEEE.

[11] Saini, J. Robotics-Level-3 [GitHub repository]. Hozzáférési URL: https://github.com/jiteshsaini/robotics-level-3

[12] Lv, X., Zhang, M., & Li, H. (2008, szeptember). Robot control based on voice command. In 2008 IEEE International Conference on Automation and Logistics (pp. 2490-2494). IEEE.

[13] CoppeliaSim felhasználói kézikönyv Elérhető: https://www.coppeliarobotics.com/helpFiles/

[14] CoppeliaRobotics ZeroMQ Remote API Overview. Elérhető: https://www.coppeliarobotics.com/helpFiles/en/zmqRemoteApiOverview.htm

[15] Generation Robots Pioneer 3DX P3DX RevA User Manual. Elérhető: https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf

[16] Pololu Corporation QTR-8RC Reflectance Sensor Array. Elérhető: https://www.pololu.com/product/2759

[17] Binary Updates. (2016, March 21). Introduction of Raspberry Pi 3 Model B. Elérhető: https://binaryupdates.com/introduction-of-raspberry-pi-3-model-b/

[18] PuTTY: a free SSH and Telnet client. Elérhető: https://www.putty.org/

[19] PuTTY Wikipedia oldala Elérhető: https://hu.wikipedia.org/wiki/PuTTY

[20] Python Software Foundation. Python: Introduction and Overview Elérhető: https://www.python.org/doc/essays/blurb/

[21] Coursera. What is Python Used For? A Beginner's Guide to Using Python. Elérhető: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python