Pannon Egyetem

Műszaki Informatikai Kar

Mérnökinformatikus BSc

**Projekt labor első beszámoló**

**Csomagszállító, önvezető autó megvalósítása Webots szimulációs környezetben**

**Karácsony Máté, Szász Áron Zsigmond, Wéninger Zalán Tibor**

Konzulens: Dr. Magyar Attila

2025.

Témakiírás

A cél különböző útvonaltervező és lokalizációs algoritmusok megvalósítása Matlab-ban vagy Python-ban, a mobil robotikához kapcsolódó eszköztárak/könyvtárak felhasználásával. A szimulációs környezet a Webots, vagy a CoppeliaSim, egy ingyenes 3D szimulációs környezet. A fizikai világot (robot, ütközés, környezet, szenzorok, aktuátorok stb.) a szimulációs környezet adja, az irányítási és útvonaltervezési, lokalizációs algoritmusok pedig a programozási környezetben fut.

**Cél**

A projekt célja egy Webots szimulációs környezetben futó, önvezető csomagszállító jármű teljes megvalósítása: útvonaltervezés (A\*), környezetkészítés JOSM módszerrel, járművezérlés controllerrel és szenzorokkal, valamint supervisor alapú csomagkezeléssel.

**Megvalósított feladatok az első beszámolóra**

Az első beszámolóra a csapat megvalósította az A\* algoritmus előkészítését az útvonaltervezéshez, a kezdetleges Webots környezetet kialakítását, a jármű kézi vezérlését és a csomagok kezelését.

**Csapattagok hozzájárulása a projekthez**

Karácsony Máté: Csomagok és jármű kezelése  
Szász Áron: A\* útkereső algoritmus és vizualizálás  
Wéninger Zalán: Webots szimulációs környezet

Tartalomjegyzék

[1. A\* útkereső algoritmus és vizualizálása 4](#_Toc211525758)

[1.1. Útkereső algoritmus megvalósítása 4](#_Toc211525759)

[1.1.1. Gridmap (négyzetháló) 7](#_Toc211525760)

[1.1.2. Utak, út szegmensek használata, mint állapot 8](#_Toc211525761)

[1.2. Fejlesztési tapasztalatok és megvalósítás 9](#_Toc211525762)

[1.2.1. Program grafikus megvalósítása 10](#_Toc211525763)

[1.2.2. Tapasztalatok 11](#_Toc211525764)

[2. Webots szimulációs környezet 12](#_Toc211525765)

[2.1. Úthálózat megvalósításának módszerei 12](#_Toc211525767)

[2.1.1. Manuális útépítés 13](#_Toc211525768)

[2.1.2. Valódi városrészlet beimportálása 14](#_Toc211525769)

[2.1.3. Rajz készítése JOSM-ban, majd importálás 15](#_Toc211525770)

[2.2. Úthálózat megvalósítása 16](#_Toc211525771)

[2.2.1. Ismerkedés a JOSM programmal 16](#_Toc211525772)

[2.2.2. Úthálózat szerkezetének megrajzolása 17](#_Toc211525773)

[3. Csomagok és jármű kezelése 20](#_Toc211525774)

[3.1. Protók áttekintése 20](#_Toc211525775)

[3.1.1. Általános áttekintés 20](#_Toc211525776)

[3.1.2. A projektben használt protók 21](#_Toc211525777)

[3.1.3. Tesla Model 3 21](#_Toc211525778)

[3.1.4. DeliveryMarker (Saját protó) 23](#_Toc211525779)

[3.2. Controllerek áttekintése 24](#_Toc211525780)

[3.2.1. Általános áttekintés 24](#_Toc211525781)

[3.2.2. A Teslát vezérlő controller 24](#_Toc211525782)

[3.3. Supervisorok áttekintése 28](#_Toc211525783)

[3.3.1. Supervisor vs. Controller 28](#_Toc211525784)

[3.3.2. A csomagokkezelő supervisor 29](#_Toc211525785)

[3.4. Összefoglalás 31](#_Toc211525786)

[Mellékletek 32](#_Toc211525787)

# A\* útkereső algoritmus és vizualizálása

A projekt keretében egy A\* útkereső algoritmus került implementálásra, melynek működése grafikusan is ábrázolásra került. A megvalósítás Python programozási nyelven, a Raylib grafikus könyvtár (Python-kötéssel) segítségével történt. A Python, mint magas szintű programozási nyelv, hatékony és gyors fejlesztést tett lehetővé.

A grafikus megjelenítéshez a Raylib könyvtár került felhasználásra. A Raylib egy nyílt forráskódú, eredetileg C nyelven írt könyvtár, amely Python-kötéssel is rendelkezik, és alkalmas 2D, valamint 3D-s alkalmazások és játékok fejlesztésére. Előnye, hogy számos beépített funkcióval rendelkezik, ugyanakkor teret enged az egyedi megoldások implementálásának is. Más játékmotoroktól eltérően nem rendelkezik grafikus szerkesztőfelülettel, így a teljes fejlesztési folyamat a kódban zajlik.

A képen képernyőkép, Téglalap, tervezés, Betűtípus látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

*1. ábra: Raylib python verziójának a logója*

## Útkereső algoritmus megvalósítása

Az útvonaltervezési feladathoz az A\* algoritmus került kiválasztásra. Az A\* egy széles körben alkalmazott gráf- és útvonalkereső algoritmus, amelyet többek között a játékfejlesztés, a robotika és a navigációs rendszerek területén is használnak.

Az A\* egy informált keresési algoritmus, amely a legrövidebb, illetve legkisebb költségű útvonal meghatározásához az útvonalköltségeket és egy heurisztikus becslést használ. Az algoritmus minden egyes csomóponthoz (állapothoz) egy költségértéket (súlyt) rendel a következő képlet alapján:

***f(n)= g(n) + h(n)***

ahol:

* **g(n):** A kezdőponttól az aktuális csomópontig megtett út tényleges költsége.
* **h(n):** Az aktuális csomóponttól a célpontig hátralévő út becsült költsége (heurisztika).
* **f(n)**: Az aktuális csomópont teljes becsült költsége.

Az algoritmus minden lépésben azt a csomópontot terjeszti ki, amely a legalacsonyabb f(n) értékkel rendelkezik. A h(n) heurisztikus függvény a megoldandó problémától függ. Geometriai távolság becslésére gyakran használatos az Euklideszi-távolság:

Az algoritmus lépései a következők:

1. A kezdő csomópont felvétele a nyílt listára (a potenciális következő lépések listája).
2. A nyílt listáról a legalacsonyabb ***f(n)*** költségű csomópont kivétele és megvizsgálása.
3. Ha a vizsgált csomópont a cél, az algoritmus véget ér.
4. Ellenkező esetben a csomópont kiterjesztése és a szomszédos csomópontok vizsgálata.
5. Minden szomszédra kiszámításra kerülnek az ***f(n)***, ***g(n)*** és ***h(n)*** értékek.
6. Ha egy már vizsgált szomszédos csomópont egy új útvonalon keresztül alacsonyabb költséggel érhető el, az értékei frissítésre kerülnek.
7. A folyamat a 2. lépéstől ismétlődik, amíg a cél elérésre nem kerül, vagy a nyílt lista kiürül.

Nyílt lista: Potenciális következő lépések listája/sora, aminek első eleme a legolcsóbb mind közül.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

*2. ábra: Részlet a költségszámítás megvalósításáról Python kódban*

Az A\* algoritmus előnyei:

* Megfelelő heurisztika alkalmazása esetén garantáltan megtalálja a legoptimálisabb útvonalat.
* A heurisztika használatával hatékonyan szűkíti a keresési teret.
* Széles körben alkalmazható.

És hátrányai:

* Jelentős memóriát igényelhet a csomópontok tárolása miatt.
* Nem megfelelő heurisztika esetén a teljesítménye lecsökkenhet.
* Bizonyos problémák esetén a heurisztika definiálása nehézségekbe ütközhet.

Ezeket a pontokat szem előtt tartva került kiválasztásra az A\*. A feladat megoldására két lehetséges megközelítés került feltárásra:

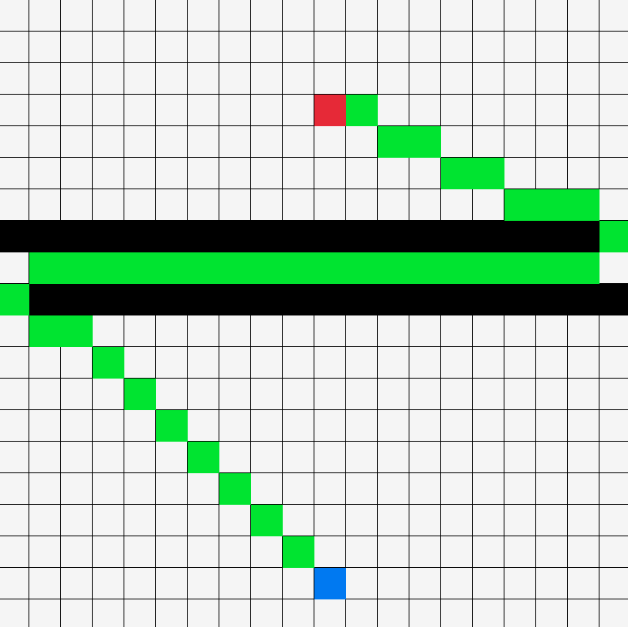
* Gridmap (négyzetháló) alapú megvalósítás.
* Útszegmensek használata csomópontokként.

### Gridmap (négyzetháló)

Az A\* algoritmust gyakran alkalmazzák négyzethálós rendszereken, ahol minden egyes négyzet egy lehetséges állapotot (csomópontot) képvisel. Ez a módszer alkalmazható diszkrét, rács alapú térképeken, vagy egy folytonos térre (pl. úthálózatra) vetített hálón, amelyet "navmesh"-nek is neveznek.

Egy csomópont (négyzet) a következő attribútumokat tárolhatja:

* Az algoritmus működéséhez szükséges ***f***, ***g*** és ***h*** értékek.
* A négyzethálóban elfoglalt sor- és oszlopindex.
* A négyzet x és y koordinátái a grafikus megjelenítéshez.
* Hivatkozás a szülő csomópontra, amelyből az aktuális állapot elérhető volt (az útvonal rekonstrukciójához).
* Speciális paraméterek, mint például a járhatóság vagy a mozgás költségét módosító tényezők (pl. akadályok, nehezebben járható terep).



3. ábra: Egy négyzetrácsos pályán a kék és piros pont közötti legrövidebb út zölddel jelölve, ahol a fekete négyzetek nem átjárhatóak

Bár ez a megoldás egyszerű, kétdimenziós térképek esetén hatékony, kanyargós, valósághű úthálózatok modellezéséhez pontatlan lehet.

A képen tér, Színesség, pixel, sor látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

*4. ábra: Próbálkozás egy kanyargós úttal egy négyzetrácsos pályán*

Lehetséges lenne a négyzethálót a bejárni kívánt útszakaszra vetíteni, majd az útvonal meghatározása után a pontatlanságokat utólagosan korrigálni, azonban ez a megközelítés nem tekinthető hatékonynak, mivel létezik ennél elegánsabb megoldás.

### Utak, út szegmensek használata, mint állapot

Egy alternatív megközelítés a négyzetháló elhagyása, és ehelyett a már meglévő úthálózat csomópontjainak (pl. útkereszteződések, útszakaszok végpontjai) állapotokként való kezelése. Ez a módszer feltételezi egy előre definiált úthálózat meglétét, ahol az összeköttetések és azok koordinátái ismertek.

A képen művészet látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

*5. ábra: Úthálózat felsőnézete, láthatóak az út szegmensek összeillesztései, melyekhez koordináta is tartozik*

Az algoritmus működési elve változatlan marad: a keresés a kezdőponttól indulva, a legkisebb költségű csomópontok (útszegmens-végpontok) kiterjesztésével halad a cél felé. Ebben az esetben a h(n) heurisztika az aktuális csomópont és a cél közötti légvonalbeli távolság alapján számítható. A két módszer összehasonlítása után kerül kiválasztásra a projektben véglegesen használt megoldás.

## Fejlesztési tapasztalatok és megvalósítás

A bemutató program fejlesztése a Raylib könyvtár Python nyelvű verziójának felhasználásával történt. A program célja az volt, hogy zárt, kontrollált környezetben lehessen tesztelni az algoritmus működését, tulajdonságait, skálázhatóságát és általános megvalósítását.

A későbbi fázisokban az algoritmus a Webots szimulációs környezetben kerül alkalmazásra, ahol fizikai modellként működő jármű fogja használni.

A Raylib könyvtár korábbi projektek során már bevált eszköznek bizonyult, így logikus választás volt a tesztprogram fejlesztéséhez. Az A\* algoritmus implementálásával korábban is történtek kísérletek, azonban a mostani megvalósítás stabilabb és funkcionálisabb alapot biztosított a további fejlesztésekhez.

### Program grafikus megvalósítása

A Raylib könyvtár az alábbi kódrészlettel teszi lehetővé egy ablak létrehozását:

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

*6. ábra: Kódrészlet egy játékablak létrehozásához*

*A képen képernyőkép, szöveg, diagram, tervezés látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.*

*7. ábra: A kódrészlet eredménye*

* init\_window() – Létrehozza az alkalmazás ablakát a megadott paraméterek alapján (szélesség, magasság, ablak címe).
* while not window\_should\_close() – A program fő ciklusát valósítja meg; a ciklus mindaddig fut, amíg a felhasználó nem zárja be az ablakot.
* begin\_drawing() / end\_drawing() – A megjelenítéshez szükséges függvénypár; minden, ami a két hívás között szerepel, a képernyőre kerül kirajzolásra.
* clear\_background() – Minden ciklus elején törli a képernyőt, így előkészíti a következő rajzolási műveletet.
* draw\_rectangle() – Egy téglalap kirajzolását végzi a megadott paraméterekkel (x-koordináta, y-koordináta, szélesség, magasság, szín).
* close\_window() – A fő ciklus befejezése után bezárja az ablakot.

A példából jól látható a Raylib könyvtár egyszerű felépítése és használhatósága. A függvénynevek egyértelműek, a működési elv pedig könnyen átlátható, ami megkönnyíti az alapvető grafikus alkalmazások fejlesztését.

A könyvtár emellett széleskörű bővíthetőséget biztosít: támogatja különböző modellek, képek, textúrák és hangfájlok betöltését is, így a megjeleníthető grafikai elemek mennyisége és összetettsége szinte korlátlan.

### Tapasztalatok

A Python és a Raylib együttes használata kényelmes és egyszerű fejlesztési környezetet biztosít bármilyen grafikai megjelenítést igénylő programhoz.

A könyvtár C és C++ nyelvű verziói nagyobb optimalizálási lehetőséget kínálnak, ezért teljesítménykritikus alkalmazások esetében ezek előnyösebbek lehetnek. A Python előnye ugyanakkor a gyors fejlesztés és az egyszerű szintaxis, amely jelentősen csökkenti a fejlesztési időt.

Az A\* algoritmus megvalósítása a korábbi elméleti ismereteknek köszönhetően hatékonyan valósult meg. A működési elv papíron történő előzetes levezetése segítette az implementáció folyamatát, így a kód megírása már célirányosan történhetett.

A fejlesztés során a legnagyobb nehézséget az állapotok indexelése jelentette. Ennek oka, hogy a kétdimenziós adattárolók elemei [sor index][oszlop index] formában érhetők el, míg a rajzolófüggvények (x koordináta, y koordináta) elrendezést használnak. Ez az eltérés kezdetben zavart okozott az adatok kezelésében, azonban a megfelelő konverziók alkalmazásával a probléma kiküszöbölhetővé vált.

# Webots szimulációs környezet

A projekt megvalósítása a Webots szimulációs környezetben történik, az úthálózat megépítése előtt elengedhetetlen fontosságú annak megismerése.

A Webots egy nyílt forráskódú szimulátor, melyet kifejezetten robotikai kísérletekhez fejlesztettek, nagy hangsúlyt fektetve a valósághű fizikai modellezésre és részletes 3D-s környezetre. A program lehetőséget nyújt különböző robotok és járművek, illetve ezek szenzorjainak szimulációjára, valamint saját virtuális világ elkészítésére.

A környezet megismerése során különösen nagy figyelmet kapnak a következő területek:

* kezelőfelület
* Scene Tree hierarchia
* objektumok elhelyezése és azok paraméterezése
* pályaépítéséhez szükséges útobjektumok (Road, Crossroad, RoadLine)

A képen képernyőkép, szöveg, Multimédiás szoftver, Grafikai szoftver látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

1. ábra: ismerkedés a Webots környezettel



## Úthálózat megvalósításának módszerei

Az úthálózat megvalósítására három módszer került feltárásra. Ezek a következők:

* utak elhelyezése és szerkesztése manuálisan Webots környezetbe
* valódi városrészlet beimportálása OpenStreetMap segítségével
* úthálózat megrajzolása JOSM programban, majd importálása Webots környezetbe

A következő alfejezetekben mindhárom módszer bemutatásra kerül, lényegi leírásukkal, illetve a problémáikkal, amik miatt a módszer cserélve lett a soron következőre.

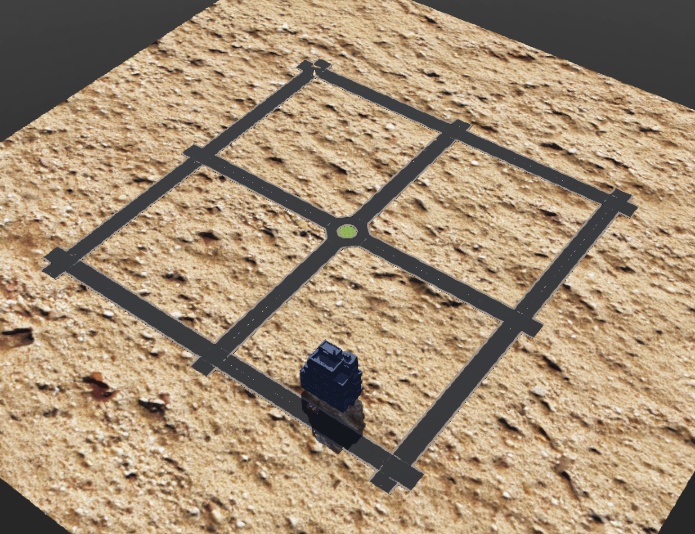
### Manuális útépítés

A legkezdetlegesebb megoldás a három módszer közül a manuális útépítés Webots környezetben. Ebben az esetben az úthálózat a „Road” és „Crossroad” típusú objektumok felhasználásával, valamint azok attribútumainak módosításával hozható létre. Az útépítés szempontjából legfontosabb paraméterek a „translation”, a „rotation” és a „wayPoints” attribútumok:

* translation: az útobjektum kezdő- vagy középpontjának elhelyezésére szolgál a világ globális koordinátarendszerében, egy koordinátapár megadásával
* rotation: az objektum elforgatását teszi lehetővé
* wayPoints: a „Road” típusú objektumoknál használható, azon koordináták listáját tartalmazza, amelyeken az útnak mindenképpen át kell haladnia

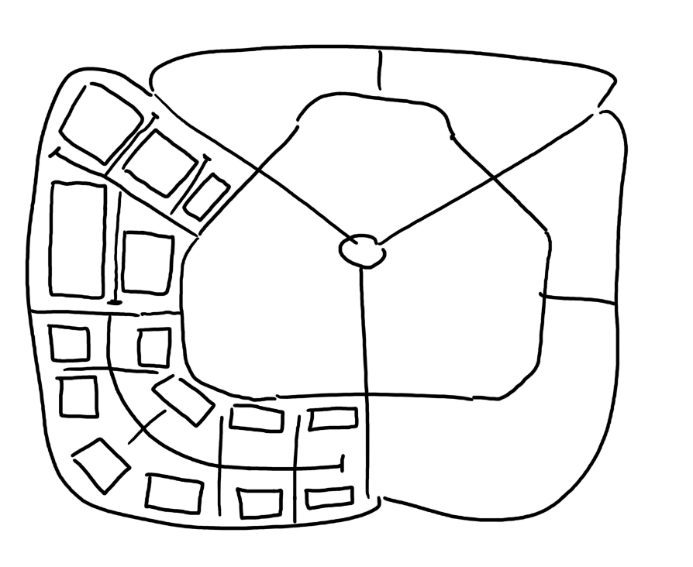
A probléma ott mutatkozik meg ebben a megoldás, hogy kimondottan időigényes

egy realisztikus és a feladat leírásának megfelelő szintű úthálózatot megalkotni, illetve amennyiben azt később bővíteni kellene, teljes utakat kell módosítani vagy legrosszabb esetben a nulláról létrehozni.



2. ábra: manuálisan összerakott úthálózat

A nehézségek közé tartozik még emellett az is, hogy a csapatban nem áll rendelkezésre olyan személy, aki megfelelő ismeretekkel rendelkezik közlekedési infrastruktúra tervezésében, ezáltal az ezzel a módszerrel létrehozott úthálózat realisztikussága korlátozott.



3. ábra: kezdeti tervrajz

### Valódi városrészlet beimportálása

A www.cyberbotics.com által nyújtott dokumentációk között található egy „OpenStreetMap Importer” nevű leírás, amely egy olyan szkritpről szól, aminek a segítségével a www.openstreetmap.org/export oldalon keresztül a világ bármely pontjáról letölthető annak virtuális mása. A letöltött részlet tartalmazza az adott területen található utakat, épületeket és egyéb objektumokat.

Az importáló szkript célja, hogy megszűntesse a szimulációs környezet manuális létrehozásával járó nehézségeket, illetve lehetővé teszi, hogy akár négyzetkilométeres területeken is valós időben lehessen szimulációt futtatni.

Az OpenStreetMap oldalról történő exportáláshoz meg kell adni a szélességi és hosszúsági fokok alsó és felső korlátait, ezt követően pedig a kijelölt terület osm (OpenStreetMap) fájlformátumban exportálható. Az így kapott fájlt ezután a Webots beépített szkriptje segítségével alakítható át egy olyan világgá, amelyet a szimulációs program képes megnyitni.



4. ábra: Budapest, IX. kerület a szimulátorban

A probléma ezzel a megközelítéssel az volt, hogy a létrehozott világ túlságosan realisztikus lett, különösen az úthálózat tekintetében. A robot működését figyelembe véve nincs szükség a többsávos utakra, sőt ezek akár az önvezető jármű hibás működését is okozhatják. Emellett előfordulhatnak olyan útszakaszok, amelyekről hiányzok a felfestések vagy színük nem megfelelő, így a robot számára az útvonal követése vagy a sávok felismerése nehézséget jelenthet. Ezen hibák kijavítása rendkívül időigényes lehet, különösen egy több négyzetkilométeres terület esetén.

### Rajz készítése JOSM-ban, majd importálás

A korábban említett dokumentációban az OpenStreetMap oldal mellett szerepel egy JOSM (Java OpenStreetMap Editor) nevű Java nyelven készült program is.

A JOSM célja, hogy az OpenStreetMap-ről exportált úthálózatok könnyen szerkeszthetők, bővíthetők legyenek, illetve lehetőséget biztosítson teljesen új úthálózatok létrehozására is.

A program használata a következő: a felhasználó egy kétdimenziós, grafikus felületen rajzolhatja meg az utakat és épületeket, hasonló módon, mint egy rajzolóprogramban.

Az így létrehozott objektumokhoz ezután különböző paraméterek rendelhetők, mint például az út típusa (főút, autópálya, mellékút), így az úthálózat gyorsan testre szabható.

A képen térkép, kereszteződés, csomópont, Városépítészet látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

5. ábra: Budapest, IX. kerület részlet, JOSM-ban megvalósítva

Az előző megoldásokban tapasztalt problémák ezzel a megközelítéssel nagyrészt megszűnnek: a program felületén egyszerűen hozzáadhatók vagy eltávolíthatók a hálózatból a nem kívánt útszakaszok, és a robot működéséhez szükséges feltételek is könnyedén biztosíthatók, anélkül, hogy a szimulációs környezet beállításait módosítani kellene.

## Úthálózat megvalósítása

Az úthálózat megvalósítására tehát a JOSM program használata bizonyult a legideálisabb megoldásnak, mivel ez felelt meg leginkább a projekt igényeinek és korlátjainak.

A hálózat elkészítésének folyamata három részre osztható:

* a JOSM programmal való megismerkedés: a kezelőfelület, eszközök és funkciók feltérképezése
* a hálózat szerkezetének megrajzolása: az utak, csomópontok és egyéb infrastruktúra-elemek vizuális felépítése a 2D-s felületen
* importálás a Webots környezetbe: a megrajzolt úthálózat exportálása megfelelő formátumban, majd annak átalakítása és tesztelése a szimulációban

### Ismerkedés a JOSM programmal

A program működését bemutató videó az OpenStreetMap Importer című Cyberbotics-dokumentációban található.

A felvételen a JOSM legfontosabb, alapvető funkciói kerülnek ismertetésre, különös tekintettel arra, hogyan módosítható egy már meglévő .osm fájl, majd hogyan importálható az a Webots környezetébe.

Bár a videó elsősorban a módosítási folyamatra fókuszál, a bemutatott eszközök segítségével a teljes úthálózat manuálisan is létrehozható, így a projekt szempontjából is jól hasznosítható ismereteket tartalmazott.

A projekt megvalósításához a következő három szerkesztőeszköz bizonyult a legfontosabbnak:

* kijelölés mód: az útszakaszok, csomópontok vagy épületek kiválasztására szolgál
* rajzolás: új útszakaszok létrehozására használható, a felhasználó egérkattintásokkal pontokat helyez el a síkon, amelyeket a program automatikusan összeköt, így határozva meg az útvonalat (később ezek a pontok fognak megjelenni „wayPoints” attribútumként a szimulációban)
* kettévágási mód: már meglévő útszakaszok felosztására szolgál

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

6. ábra: JOSM szerkeszőteszközök

### Úthálózat szerkezetének megrajzolása

A programot magyarázó videóban egy „my\_world.osm” nevű világgal mutatják be a program funkcióit és szerkesztőeszközeit.

Ez a világ szerkezeti felépítését és méretarányait tekintve is ideálisnak bizonyult a projekt igényeihez: elrendezése egyszerű, átlátható, ugyanakkor elegendő részletet tartalmaz ahhoz, hogy valósághű környezetet biztosítson a robot működésének teszteléséhez.



7. ábra: "my\_world.osm" a Webots környezetben

Az útszakaszok kialakítása kevésbé realisztikus, mint egy valódi városi úthálózaté, ami előnyös a projekt szempontjából, mivel így a robot navigációját nem befolyásolják a felesleges komplexitások, például a többsávos utak.

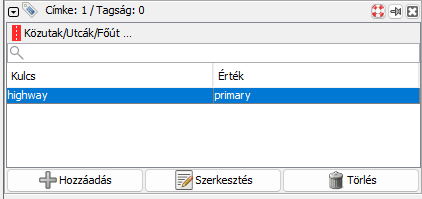
Ez a világ azonban nem érhető el semmilyen nyilvános forrásban, ezért a rekonstrukciója az előző fejezetben ismertetett módszer alapján történt. A JOSM-ben történő rajzolás során különös figyelmet kellett fordítani az arányok helyes megválasztására, mivel a méretek hibás beállítása esetén a létrehozott úthálózat a valóságosnál lényegesen nagyobb, akár több kilométeres, kiterjedésűvé válhat.

A képen művészet látható

Előfordulhat, hogy az AI által létrehozott tartalom helytelen.

8. ábra: az úthálózat rekreációja

Az így megrajzolt útszakaszokat ezt követően címkékkel (kell ellátni, amelyek meg határozzák az utak különböző tulajdonságait a szimuláció során. A címkék segítségével adhatók meg többek között az útburkolati jelek, a sávok száma, az út típusa (például főút, mellékút, gyalogút), valamint a forgalmi irány vagy a kötelező haladási irány.



9. ábra: „főút” címkével megjelölt útszakasz

Ezt követően az elkészült úthálózat .osm fájlként menthető. A mentett állomány ezután az 1.1.2. alfejezetben ismertetett módszerrel alakítható át a Webots szimulációs környezet által értelmezhető világfájllá (.wbt), így a megtervezett pálya közvetlenül betölthető és tesztelhető a programban.



10. ábra: az elkészült úthálózat a szimulációban

Ezzel az úthálózat azon részei, amelyekkel a robot közvetlenül kapcsolatba kerül, elkészültek. A projekt következő fázisaiban a hangsúly a pálya vizuális és funkcionális kiegészítésére helyeződik, amely magában foglalja a környezet részletgazdagabbá tételét, például épületek, fák és egyéb dekorációs elemek elhelyezését. Igény esetén a meglévő úthálózat is tovább bővíthető, hogy a szimuláció még életszerűbb és változatosabb legyen.

# Csomagok és jármű kezelése

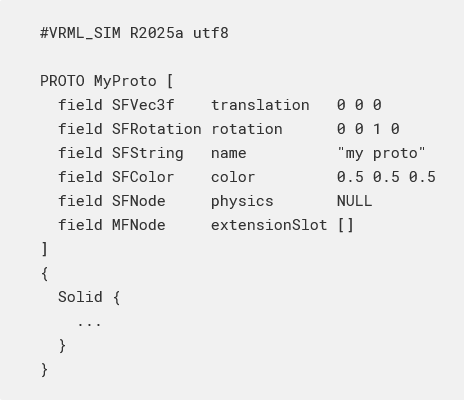
Ez a dokumentáció bemutatja a Webots szimulációs környezet három alapvető fogalmát: a protót, a controllert és a supervisort. Az egyes fejezetek részletesen ismertetik ezen elemek működését, használatát és a projektben betöltött szerepüket, kitérve a konkrét megvalósítások magyarázatára is.

## Protók áttekintése

### Általános áttekintés

A protó (angolul: PROTO) egy újrahasznosítható objektum sablon a Webots-ban. Lényegében minden, a szimulációs térben látható és kezelhető elem – például járművek, utak, épületek vagy a csomagot jelölő doboz – egy protóként van definiálva. Technikailag a protók különböző csomópontok (nodes) rendezett halmazai, amelyek egy komplex objektumot alkotnak.

A Webots egyik legnagyobb előnye a beépített protók széles könyvtára, amely jelentősen megkönnyíti a fejlesztést. Ezeket a meglévő sablonokat szabadon felhasználhatjuk, módosíthatjuk, vagy akár teljesen új, egyedi protókat is létrehozhatunk a semmiből. A program a protókat egyszerű, szöveges .proto fájlokban tárolja, amelyek könnyen szerkeszthetők.



1. ábra: példa protó fájl

Egy protó fájl általános szerkezete a következő: egy fejlécből, a protó nevéből, a paraméterezhető mezőkből (protoFields), valamint a protó testéből (protoBody) áll, amely a tényleges csomópontokat tartalmazza.



2. ábra: egy protó általános szerkezete

### A projektben használt protók

A szimulációhoz felhasználtunk egy beépített és egy egyedi készítésű protót.

### Tesla Model 3

A csomagszállítási feladatot egy, a Webots által biztosított Tesla Model 3 protó segítségével valósítottuk meg. A járművet egyedi controller vezérli, és kiegészítettük egy Display csomóponttal, amely egy kis ablakban jeleníti meg a sebességre és kormányzásra vonatkozó telemetriai adatokat.

A jármű a szimulációs világot leíró .wbt fájlban az alábbiak szerint lett definiálva:

DEF TESLA TeslaModel3 {

translation -10 5 0.5

rotation 0 0 1 -1.5707953071795862

controller "keyboard\_drive"

sensorsSlotCenter [

Display {

width 300

height 80

}

]

}

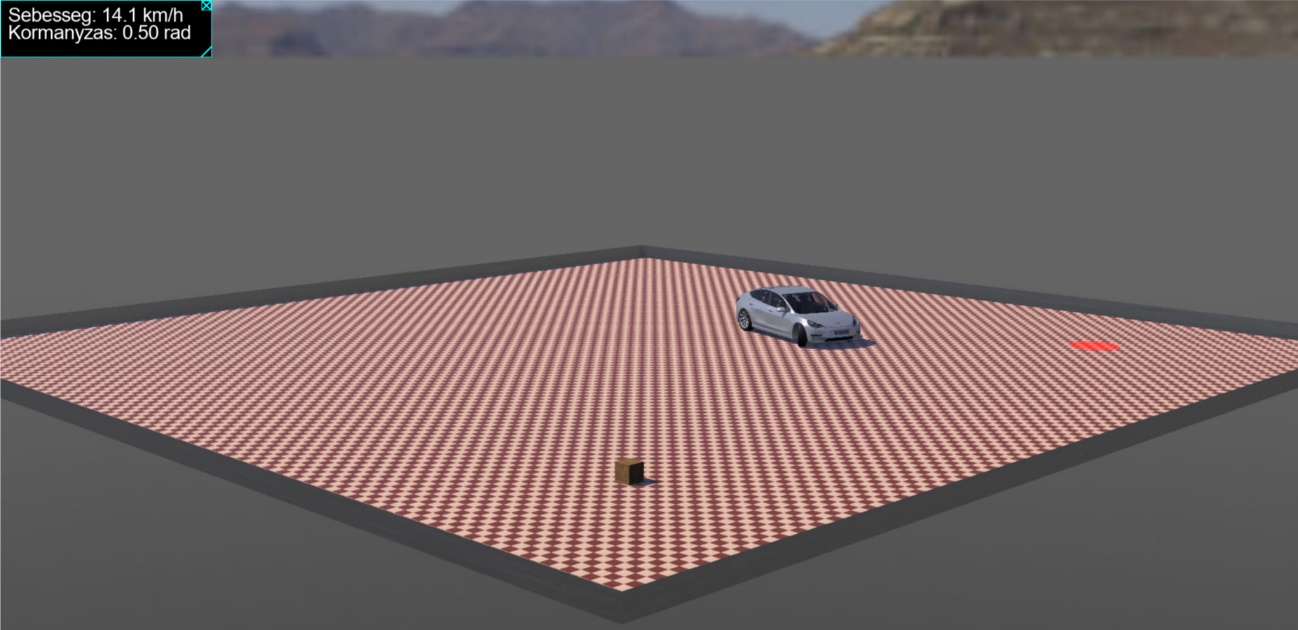
1. kódrészlet: Tesla protó kódja

* DEF TESLA: Egyedi nevet ad a protó példánynak, amellyel később hivatkozni lehet rá.
* translation: Az objektum pozíciója a 3D térben (x, y, z) méterben.
* rotation: Az objektum elforgatása egy tengely körül (x, y, z, szög) radiánban.
* controller: Meghatározza, hogy melyik vezérlő szkript felelős a jármű irányításáért.
* sensorSlotCenter: A jármű központi szenzorhelye, ahova a Display ablakot illesztettük.

A protó használatához szükséges a .wbt fájl elején importálni azt a Webots hivatalos GitHub repositoryjából.

EXTERNPROTO "https://raw.githubusercontent.com/cyberbotics/webots/R2025a/projects/vehicles/protos/tesla/TeslaModel3.proto"

2. kódrészlet: Szükséges import a Tesla működéséhez



3. ábra: Tesla működés közben

### DeliveryMarker (Saját protó)

A projekt tartalmaz egy teljesen egyedi készítésű protót is, amely a csomag kiszállítási helyét jelöli a pályán. Ez egy áttetsző, piros vagy zöld színű henger, amelynek tulajdonságai (pozíció, szín) paraméterezhetők.

#VRML\_SIM R2025a utf8

PROTO DeliveryMarker [

field SFVec3f translation 0 0.01 0

field SFColor color 1 0 0

]

{

Transform {

translation IS translation

children [

Shape {

appearance Appearance {

material Material {

diffuseColor IS color

emissiveColor IS color

transparency 0.3

}

}

geometry Cylinder {

height 0.02

radius 0.7

}

}

]

}

}

3. kódrészlet: Marker-t megvalósító protó

* PROTO DeliveryMarker [...]: Definiálja az új protó nevét és annak mezőit (translation és color).
* Transform: Egy csomópont, amely a gyermekeire (itt a Shape-re) alkalmaz geometriai transzformációkat, mint például az eltolást. Az IS translation összeköti a Transform csomópont translation mezőjét a protó translation paraméterével.
* Shape: Egy látható geometriai alakzatot definiál.
* Appearance és Material: Az objektum vizuális tulajdonságait határozzák meg, mint a szín (diffuseColor), az önálló világítás színe (emissiveColor) és az átlátszóság (transparency).
* Cylinder: A geometry csomópont, amely egy egyszerű hengert hoz létre megadott magassággal és sugárral.

## Controllerek áttekintése

### Általános áttekintés

A controllerek azok a szkriptek, amelyek a szimulációban szereplő robotok és járművek "agyát" képezik. Ezek felelősek a szenzoradatok feldolgozásáért és a végrehajtó szervek (motorok, kerekek) vezérléséért. A Webots támogatja a C, C++, Java, Python és MATLAB programozási nyelveket, így a fejlesztők a számukra leginkább ismert környezetben dolgozhatnak.

A projektben a Python nyelvet választottuk, mivel magas szintű nyelvi eszközei és a mobil robotikában elterjedt csomagjai egyszerűsítik a fejlesztést.

### A Teslát vezérlő controller

A projektben a Tesla manuális irányítását egy keyboard\_drive.py nevű Python controller valósítja meg. Ez a szkript a billentyűzetről beolvasott jelek alapján vezérli a járművet.

# --- Importok ---

from vehicle import Driver

from controller import Keyboard

# --- Konstansok ---

TIME\_STEP = 16

THROTTLE\_STEP = 0.05 # 0 es 1 kozotti ertek

BRAKE\_STEP = 0.1 # 0 es 1 kozotti ertek

STEERING\_STEP = 0.03 # Radian

MAX\_STEERING\_ANGLE = 0.5 # Radian

# --- Inicializalas ---

driver = Driver()

keyboard = Keyboard()

keyboard.enable(TIME\_STEP)

# --- HUD beallitasa ---

try:

display = driver.getDevice("display")

except:

display = None

print("ERROR: 'display' nem talalhato")

font = "Arial"

# --- Alap ertekek beallitasa ---

throttle = 0.0

brake = 0.0

steering = 0.0

gear = 1 # Alapertelmezett fokozat: elore menet

is\_gear\_key\_down = False

driver.setGear(gear)

print("↑↓ - gaz/fek, ←→ - kanyarodas, G - sebessegvalto, Space - kezifek")

# --- Esemenyek kezelese ---

while driver.step() != -1:

key = keyboard.getKey()

current\_key\_is\_G = (key in (ord('G'), ord('g')))

is\_throttle\_active = (key == Keyboard.UP)

is\_brake\_active = (key == Keyboard.DOWN or key == ord(' '))

is\_steering\_active = (key == Keyboard.LEFT or key == Keyboard.RIGHT)

# --- Gazadas ---

if is\_throttle\_active:

throttle = min(1.0, throttle + THROTTLE\_STEP)

brake = 0.0

# --- Fekezes ---

elif is\_brake\_active:

if key == Keyboard.DOWN:

brake = min(1.0, brake + BRAKE\_STEP)

elif key == ord(' '):

brake = 1.0

throttle = 0.0

# --- Kanyarodas ---

if key == Keyboard.LEFT:

steering = max(-MAX\_STEERING\_ANGLE, steering - STEERING\_STEP)

elif key == Keyboard.RIGHT:

steering = min(MAX\_STEERING\_ANGLE, steering + STEERING\_STEP)

# --- Valto ---

if current\_key\_is\_G and not is\_gear\_key\_down:

if throttle == 0.0: # Csak akkor, ha allo helyzetben van

gear = -gear

driver.setGear(gear)

brake = 0.0

print("Fokozat:", "Eloremenet" if gear == 1 else "Hatramenet")

is\_gear\_key\_down = True

elif not current\_key\_is\_G:

is\_gear\_key\_down = False

# --- Kormany visszaallitasa ---

if not is\_steering\_active:

steering \*= 0.90

# --- "Motorfek" szimulacioja ---

if not is\_throttle\_active and not is\_brake\_active:

throttle \*= 0.98

brake \*= 0.9

# --- Ertekek beallitasa ---

driver.setSteeringAngle(steering)

driver.setThrottle(throttle)

driver.setBrakeIntensity(brake)

# --- HUD ---

if display:

display.setColor(0x000000)

display.fillRectangle(0, 0, display.getWidth(), 60)

display.setColor(0xFFFFFF)

display.setFont(font, 20, True)

speed = driver.getCurrentSpeed()

display.drawText(f"Sebesseg: {speed:.1f} km/h", 10, 10)

display.drawText(f"Kormanyzas: {steering:.2f} rad", 10, 35)

4. kódrészlet: Tesla irányítását megvalósító controller

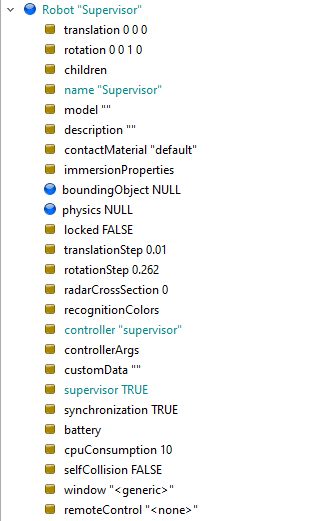
A controller kódjának főbb funkciói:

1. Inicializálás: Létrehozza a Driver és Keyboard objektumokat, amelyek a jármű vezérléséhez és a billentyűzet figyeléséhez szükségesek. Emellett beállítja a Display eszközt a HUD (Head-Up Display) megjelenítéséhez.
2. Fő ciklus: A while driver.step() != -1: ciklus a szimuláció minden egyes időlépésében lefut.
3. Billentyűleütések kezelése: A ciklus minden iterációjában lekérdezi a lenyomott billentyűt (keyboard.getKey()).
   1. Fel/Le nyilak: A gázadást (throttle) vagy a fékezést (brake) szabályozzák.
   2. Jobb/Bal nyilak: A kormányzást (steering) állítják.
   3. 'G' billentyű: Vált a hátra- és előremeneti sebességfokozat között, de csak álló helyzetben.
   4. Space: Vészfékezést (maximális fékintenzitást) aktivál.
4. Szimulált fizika: A kód finomítja a vezérlést:
   1. Ha a kormányzás nem aktív, a kerekek lassan visszaállnak egyenesbe (steering \*= 0.90).
   2. Ha sem a gáz, sem a fék nincs lenyomva, a kód "motorféket" szimulál a sebesség lassú csökkentésével (throttle \*= 0.98).
5. Parancsok kiadása: A kiszámított értékeket a driver objektum megfelelő függvényeivel (setSteeringAngle, setThrottle, setBrakeIntensity) továbbítja a járműnek.
6. HUD frissítése: A Display ablakra kirajzolja az aktuális sebességet és a kormányszöget.

## Supervisorok áttekintése

### Supervisor vs. Controller

A supervisor egy speciális jogosultságokkal rendelkező controller. Míg egy hagyományos controller csak a hozzárendelt robotot vagy járművet képes irányítani, a supervisor a szimuláció teljes egészét manipulálhatja: csomópontokat (objektumokat) hozhat létre, törölhet, vagy módosíthatja azok bármely tulajdonságát (pl. pozíció, szín) futás közben. A használatához egy Robot csomópontot kell létrehozni, és annak supervisor mezőjét TRUE-ra kell állítani.



4. ábra: Supervisor csomópontok

### A csomagokkezelő supervisor

A projektünkben a supervisor felel a csomagszállítási logika megvalósításáért: figyeli a jármű pozícióját, és kezeli a csomag felvételét és leadását.

# --- Importok ---

from controller import Supervisor

import math

# --- Konstansok ---

TIME\_STEP = 16

PICKUP\_DISTANCE = 4.5 # Meter

DROPOFF\_DISTANCE = 4.5 # Meter

TESLA\_DEF = "TESLA"

PACKAGE\_DEF = "PACKAGE"

MARKER\_DEF = "MARKER"

# --- Inicializalas ---

supervisor = Supervisor()

tesla = supervisor.getFromDef(TESLA\_DEF)

package = supervisor.getFromDef(PACKAGE\_DEF)

marker = supervisor.getFromDef(MARKER\_DEF)

if tesla is None or package is None or marker is None:

print("ERROR: Hianyzo NODE vagy DEF nev")

exit(1)

has\_package = False

delivered = False

# --- Jelolo szin reset ---

marker\_color\_field = marker.getField("color")

marker\_color\_field.setSFColor([1, 0, 0]) # Piros

# --- Tavolsag meghatarozasa ---

def distance(a, b):

return math.sqrt((a[0]-b[0])\*\*2 + (a[1]-b[1])\*\*2)

# --- Esemenyek kezelese ---

while supervisor.step(TIME\_STEP) != -1:

# --- Poziciok lekerese ---

tesla\_pos = tesla.getPosition()

package\_pos = package.getPosition()

marker\_pos = marker.getPosition()

if delivered: # Kiszallitas utan

continue

if not has\_package and distance(tesla\_pos, package\_pos) < PICKUP\_DISTANCE: # Csomag felvetele

print("Csomag felveve!")

has\_package = True

package.getField("translation").setSFVec3f([0, 0, -10]) # Kikuldjuk a vilagbol

elif has\_package and distance(tesla\_pos, marker\_pos) < DROPOFF\_DISTANCE: # Csomag leszallitasa

print("Csomag kiszallitva!")

has\_package = False

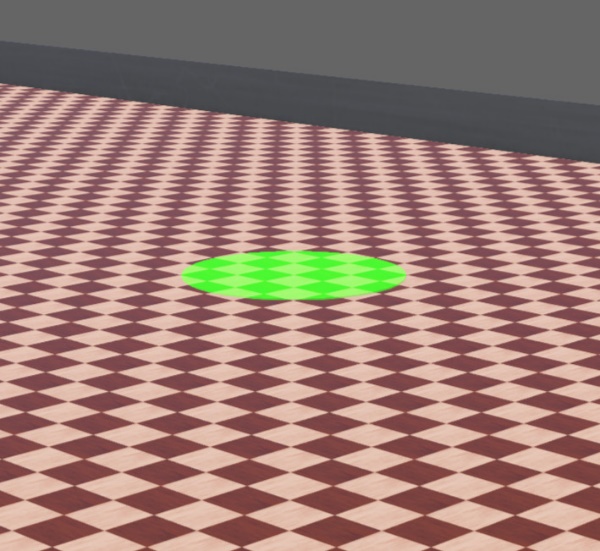
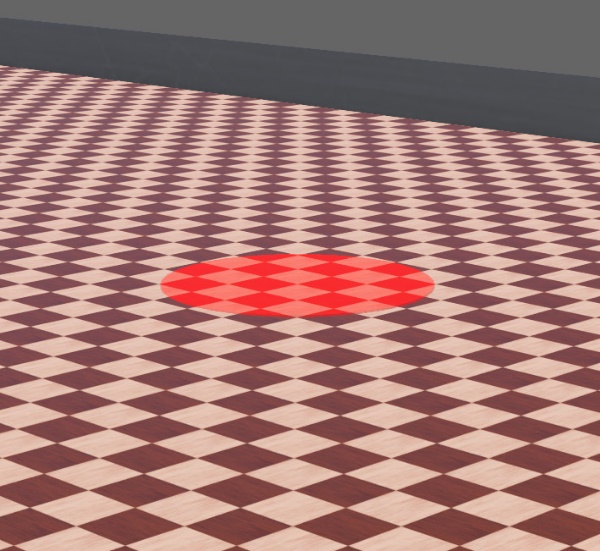
delivered = True

marker\_color\_field.setSFColor([0, 1, 0]) # Zold

5. kódrészlet: Csomagok kezelését végző supervisor

A supervisor.py szkript működése:

1. Inicializálás: A supervisor lekéri a szimulációban szereplő kulcsfontosságú objektumokat (a TESLA járművet, a PACKAGE dobozt és a MARKER jelölőt) a DEF nevük alapján. Emellett alaphelyzetbe állítja a jelölő színét pirosra.
2. Fő ciklus: A szimuláció minden időlépésében lefut.
3. Pozíciók lekérdezése: Folyamatosan lekéri a Tesla, a csomag és a céljelölő aktuális pozícióját.
4. Távolságszámítás: Egy segédfüggvény segítségével kiszámolja a Tesla távolságát a csomagtól és a célterülettől.
5. Csomagfelvétel: Ha a jármű a csomag közelébe ér (a PICKUP\_DISTANCE-nél közelebb) és még nem vette fel azt, a supervisor a csomagot "eltünteti" a pályáról azáltal, hogy egy láthatatlan helyre teleportálja ([0, 0, -10]).
6. Csomag leadása: Ha a jármű már felvette a csomagot és a céljelölő közelébe ér (a DROPOFF\_DISTANCE-nél közelebb), a supervisor a kézbesítést sikeresnek minősíti, és a jelölő színét zöldre változtatja.



5. ábra: Piros és zöld jelölő

## Összefoglalás

Ez a rész bemutatta a Webots szimulációs szoftver három kulcsfontosságú elemét: a protókat, mint újrahasznosítható objektumokat, a controllereket, mint a robotok viselkedését meghatározó szkripteket, és a supervisorokat, amelyek a szimulációs környezet egészét képesek dinamikusan manipulálni.

A bemutatott projektben ezen eszközök segítségével egy manuálisan, billentyűzettel irányítható járművel megvalósított csomagszállítási feladatot hoztunk létre. Ez a rendszer stabil alapot biztosít a további fejlesztésekhez, amelyek végső célja egy teljesen önvezető, szenzorokkal felszerelt csomagkihordó jármű létrehozása.

Mellékletek

GitHub repository: <https://github.com/Karacsony-Mate/delivery_simulation>

Mappaszerkezet:

delivery\_simulation/

│

├── algorithms/

├── webots/

│ ├── worlds/

│ └── controllers/

├── integration/

├── docs/

│

├── .github/

│ ├── workflows/

│ ├── ISSUE\_TEMPLATE/

│ └── PULL\_REQUEST\_TEMPLATE.md

│

├── .gitignore

└── README.md