A szakdolgozat második felében továbbfejlesztettem a már meglévő webots világot és a vezérlőt is. Az autót újra kellett csinálni, mert nem volt teljesen rendben a fizikája, nem interaktált jól a webotsba beépített út elemekkel. Kinézetre nagyjából ugyanolyan lett, mint az előző, csak mostmár tényleg hasonlít is autóra.

Ezúttal a beépített AckermannVehicle-ből indultam ki. Az AckermannVehicle PROTO segítségével a könnyedén létre lehet hozni bármilyen járművet, amely megfelel az Ackermann modellnek. Ez a modell azt az autó, vagy más jármű kormányzásánál fellépő geometriai problémát oldja meg, amikor egy kanyarban a belső és külső oldalon lévő kerekeknek különböző sugarú köröket kell leírniuk. A PROTO egy olyan webots mechanizmus, amellyel a felhasználók hozzáadhatnak saját nodeokat a már meglévőkhöz, így nagyon komplex objektumokat hozhatnak létre, és ezeket könnyen újból fel is tudják használni. Ebben a PROTO-ban kerekek pozíciója és tájolása automatikusan kiszámításra kerül a PROTO mezői alapján. Ezek a kerekek automatikusan kapcsolódnak a megfelelő csuklókhoz, hogy a megfelelő tengely mentén forogjanak. Az aktuátorok az első kerekekhez vannak csatlakoztatva, hogy lehessen kormányozni a járművet. Az AckermannVehicle PROTO egy Robot típusú node, ami azt jelenti, hogy tudom kontrollerrel irányítani. Az extensionSlot nevű mezőjében pedig hozzá tudok adni minden szükséges plusz dolgot (az autó alakja, érzékelők, egyéb aktuátorok). Felhasználtam még a beépített Car PROTO-t is, amely az AckermannVehicle-ből származódik és továbbfejleszti azt. A Driver könyvtárral együtt használva hozzáad az autóhoz egy motor modellt, sebességváltót, lámpákat és fék modellt.

A jármű fizikáját ismét egy box shape képviseli, a váz és a többi alkatrész pedig mind egy-egy Transform node. Ennek a nodenak nincs fizikája, így nem ütik egymást az alkatrészek a lépések számításakor. Ezúttal 7 távolságszenzort használtam, hatot az akadályok és falak kerülésére, egyet pedig, hogy ne menjen neki az úton haladó többi autónak. Minden szenzornak beállítottam a LookUpTable-jét, amely megmondja, hogy milyen távolságról érzékeljen, és hogy ha x távol érzékel, arra mekkora értéket adjon vissza.

Ezután létrehoztam egy kezdetleges pályát a webots-os utak segítségével. Először csak a StraightRoadSegment-et és a RoadIntersection-t használtam, ami az egyenes út és a kereszteződés.

A kontroller nyelvet átváltottam python-ra. Sokkal egyszerűbb a használata, és kell majd függvényeket definiálnom, ami C -ben nem triviális. Szóval először a meglévő kontrolleremet írtam át python-ra és azt kezdtem el továbbfejleszteni. A felépítése ugyanaz, mint C-ben, először a külső könyvtárak/header-ök/fájlok behúzása, globális függvénydefiníciók és -változók létrehozása, szükséges előszámolások elvégzése, szenzorok és aktuátorok definiálása, majd a while-on belül maga az irányítás.

A bearing (irány) és a távolság számítás voltak azok dolgok, amiket változtatás nélkül megtartottam az előző kontrollerből. A többi dolgon vagy a nyelvváltás, vagy a feladat bővülése miatt változtatni kellett. A szenzorok definiálása és az értékeik lekérése hasonlóképpen történik, csak nem használok enumot.

Először amikor összeraktam a teszt úthálózatot, ugyanúgy, mint a szakdolgozat első felében, kézzel adtam meg az autónak a koordinátákat, ahová mennie kellett. Ez most az miatt sem lenne megfelelő, mert nagyon nagy lesz a végleges utam, nagyon időigényes és rosszul is néz ki. Tehát ezen okok miatt váltottam át arra, hogy kihasználom a beépített utak beállítható értékeit. Mindegyiknek van ugye helyzete, rotációja, valamint beállítható nekik, hogy melyik ID-jú úthoz csatlakoznak. Az egyenesnél és a kanyarnál ez egy kezdő és egy végpont, a kereszteződésnek pedig egy lista. A helyzetük az egyeneseknél az út eleje, kanyarnál a kör középpontja, kereszteződésnél pedig a közepe, ezért ennél nem is kellett további számolás, hogy meghatározzam a koordinátákat az autónak. Egyenesnél a helyzethéhez házzáadtam a hossza felét, így megkaptam a közepét, kanyarnál pedig, a rotációjától függően a görbületi faktorából számoltam ki a koordinátákat.

Ezeket a számolásokat nem a kontrolleremben végeztem el, hanem egy Supervisor-ben. Ha a robotomnak a supervisor mezője igazra van állítva, továbbra is rendesen robotként, de emellett használhatók vele a supervisor függvények, melyekkel lehett vezérelni a szimulációt, vagy módosítani a világ elemeit. Ez helyettesíti általában az emberi tevékenységeket, mint a megtett távolság mérése vagy a robot visszahelyezése kiindulási helyzetbe és még nagyon sok minden másra. A lényege az, hogy általában olyan funkciókat lát el, amelyek általában nem találhatók meg valódi robotokon, inkább egy emberi beavatkozásnak felelnek meg és a valóságban nehéz is átültetni őket.

Amikor kiszámolom az utak koordinátáit, a python pickle metódusa segítségével kiírom őket egy fájlba, majd ezt a fájlt olvasom be a kontrolleremmel, így már egy kész dictionary (python tároló, amely kulcs:érték párokból tevődik össze, az én esetemben ezek „út id : x és y koordináta”) áll rendelkezésemre.

Ezt a dictionaryt felhasználva határozom meg mely útvonalon kell az autómnak mennie, egy BFS (szélességi keresés) algoritmus segítségével, amelynek megadom a gráfomat (a dictionary-m), a kiindulási valamint a végpontomat, az pedig visszaadja a legrövidebb utak közül azt, amelyiket először megtalálta, amin végigmenve el tud jutni az autóm a kezdőpontból a végpontba.

A kontrolleremben minden lépés elején lekérem a távolságszenzoraim, a GPS és az iránytű értékeit, a jelenlegi időt, a távolságot és az irányt a célponthoz. Az első lépésben kiírom, hogy kezdjük azzal, hogy megkeressük a legrövidebb utat a célpontba, majd az említett utat, amit a BFS algoritmusom megtalált. Ezután, hacsak nem vagyunk már végpontban, elindul az autó a második koordináta felé (az első a kiinduló pont). Első sorban azt figyelem, hogy elértük-e a következő koordinátát, ha igen, és ez a végpont, megáll az autó, ha nem az, akkor megy tovább a kijelölt útvonalon.

Figyelni kell arra, hogy az autó a célpont felé haladjon, ezt a bearing (irány) mutatja meg, ettől függően valamekkora angle-ben (szög) kanyarodni kell. Ezt az értéket adom oda a kerekeknek, és ezt számolja ki az AckermannVehicle, hogy a kerekeknek mekkora szögben kell elforulni, hogy az autóm akkora szögben forduljon, amekkorát én átadok neki értéknek. Ez -0,75 –től (nagyon balra), 0 –án át (egyenes) 0,75 –ig (nagyon jobbra) terjedhet.

Ezután figyelem azt, hogy az autó milyen útszakaszon van éppen. Ha egyenesen vagy kanyaron van, akkor mindenképp tartsa a jobb sávot (ha már irányban van), ha pedig kereszteződésben van, akkor előbb forduljon el a megfelelő irányba, majd amikor már elhagyta a kereszteződést, csak akkor tartson jobbra. A sebességem, addig amíg nincs irányban 1-10 (attól függ mennyire kell kanyarodni) között változik, majd ha már irányban van felmehet 50-ig, mintha városon belül vezetnénk.

Legvégül nézem a távolságszenzoraim értékét. Ezeknek az értékei mindenképpen felülírnak minden más eddigi értéket (sebesség és fordulás), annak érdekében, hogy neki ne menjen valaminek.

A kontroller végén átadom a kiszámolt sebesség és fordulás értéket az autónak.