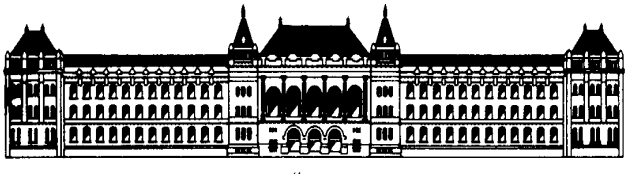


Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésmérnöki és Járműmérnöki Kar
Közlekedésautomatikai Tanszék

Swarco változtatható jelzésképű táblák vezérlőprogramjának fejlesztése

Biró Sándor
ACB8B4

2011.

Tartalomjegyzék

Ábrajegyzék

1. Absztrakt

A dolgozat egyik fő célja a változtatható jelzésképű táblák forgalmi- és gazdasági hatásainak vizsgálata volt. A munka másik célja a BME Közlekedésautomatikai Tanszék közúti laborjában található SwarcoFuturit táblák vezérlő programjának fejlesztése, és ennek segítségével a táblák automatizált, dinamikus rendszerként történő felhasználásának előkészítése. Az eredményeket Vissim szimuláció segítségével demonstráljuk, amelyet COM felületen keresztüli programozással készítettem fel a program működésének vizsgálatához.

Kulcsszavak:

közlekedés, VJT, vezérlőprogram, Vissim

Konzulens:

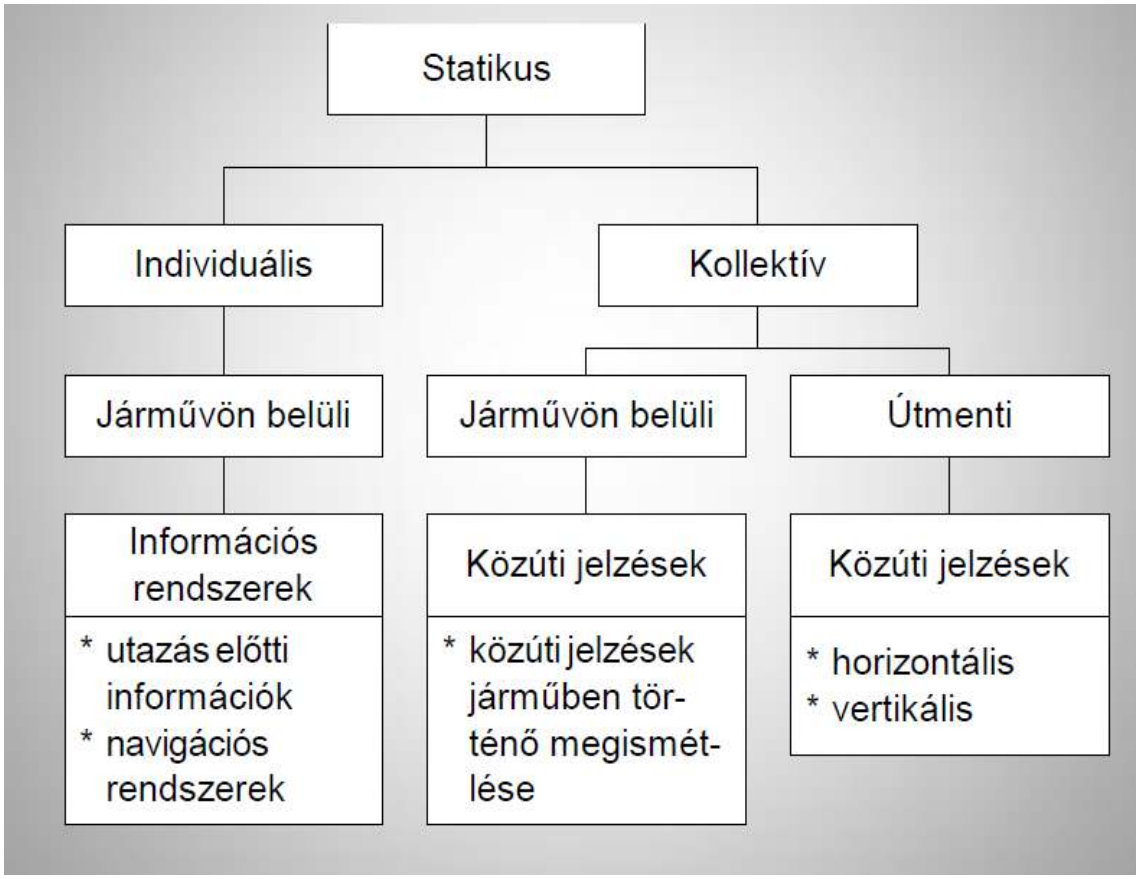
Tettamanti Tamás

2. Bevezetés

A közlekedés a jelenlegi társadalmi és gazdasági rendszer egyik alapköve, amely nélkül az egész mostani életünk elképzelhetetlen lenne. Sőt, a közlekedéssel kapcsolatos igények napról-napra növekednek. Az igények kielégítése nem megoldható pusztán az úthálózat bővítésével, hiszen ez a legdrágább módszer, és a rendelkezésre álló hely is egyre korlátozottabb. Napjaink tendenciája, hogy inkább a forgalom szabályozásával (direkt és indirekt eszközök segítségével) igyekeznek a meglévő kapacitások maximális kihasználását biztosítani. A forgalomszabályozás fontos része az utazók tájékoztatása, információkkal, ajánlásokkal és szükség esetén utasításokkal való ellátása. A legjobb megoldás természetesen az lenne, ha minden közlekedő számára a neki legmegfelelőbb – gyakorlatilag személyre szabott - információkat nyújthatnánk, ennek azonban jelenleg még komoly technikai akadályai vannak. Ezen probléma megoldásáig kollektív információs rendszerek segítségével kell a lehető legjobb eredményt elérnünk. A közlekedők dinamikus jellegű, kollektív informálására jelenleg a változtatható jelzésképű táblákat (VJT) alkalmazzák. Ezek a táblák lényegben hatalmas kijelzők, amelyeken tetszőleges üzenet (szöveg és ábra) megjeleníthető. A VJT-k tehát nem önmagukban használatos eszközök. Más rendszerekkel összekapcsolva használják őket, az adott rendszer által javasolt információk közlésére. A szakdolgozat készítése során rendelkezésre álló SwarcoFuturitVJT-k vezérlő programjának fejlesztésével foglalkoztam. Célom az volt, hogy a táblákat alkalmassá tegyem bármilyen, a forgalomra reagáló és azt dinamikusan szabályozó rendszerekkel való együttműködésre. A kialakított kommunikációs- és vezérlőprogram egy intelligens rendszert valósít meg, amely segítségével emberi jelenlét nélküli, automatizált vezérlés alakítható ki a VJT-k számára. A munka eredményének gyakorlati használhatóságát is szeretném hangsúlyozni, hiszen a magyar autópályákon üzemelő VJT-k nem vagy csak részben működnek intelligens, automatizált rendszerben.

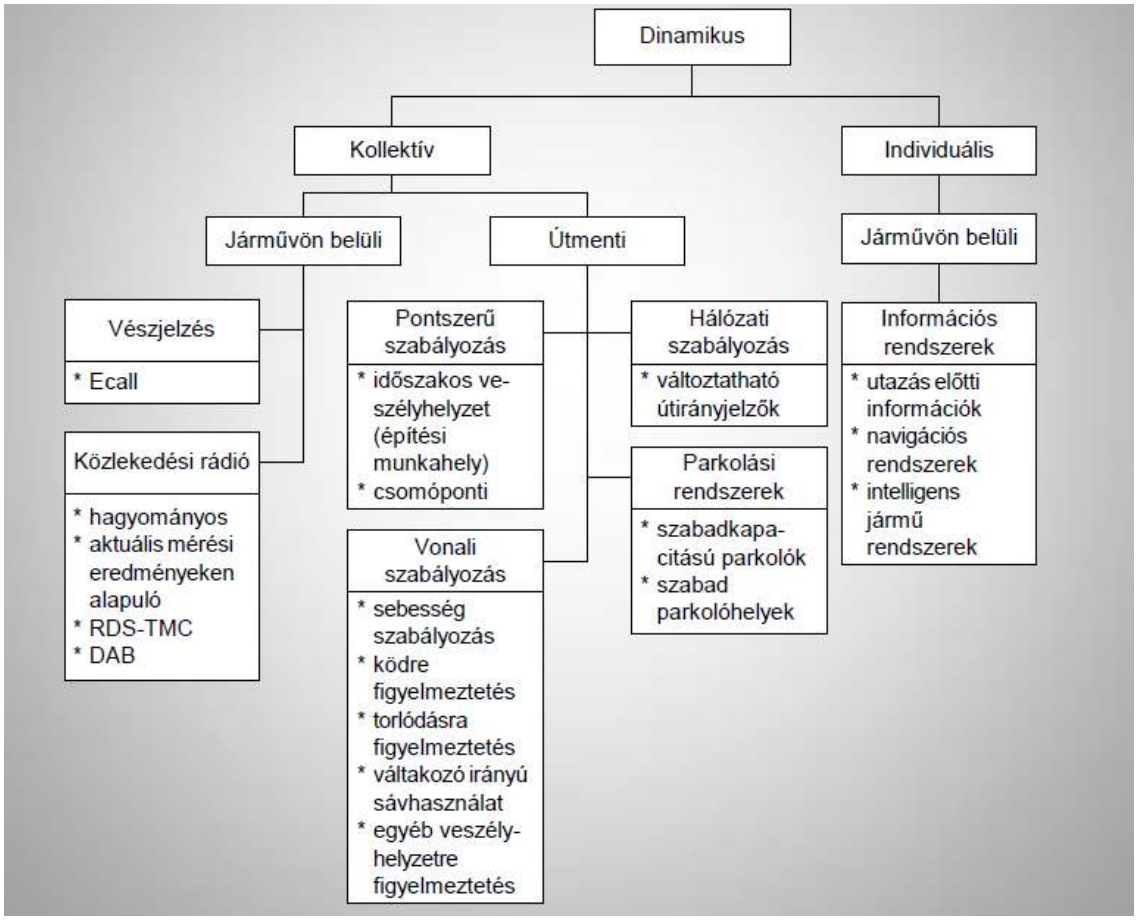
3. Változtatható Jelzésképű Táblák (VariableMessageSign[1.])

A közúti közlekedés nagyfokú fejlődése szükségessé teszi annak irányítását, szabályozását. Ennek érdekében különböző rendszereket fejlesztettek ki a forgalom kezelésére és a közlekedők tájékoztatására. A közlekedők tájékoztatására a közúti információs rendszerek szolgálnak, amelyeket, a következőképpen lehet csoportosítani:



1. ábra: Statikus információs rendszerek (forrás: Közúti informatika jegyzet[1.])

Individuális út menti rendszer kiépítésére nincs lehetőség, és nem is lenne célszerű. Ha már úgy is csak egy közlekedőnek szól, akkor sokkal jobb megoldás bejuttatni az információt a járműbe, főleg annak figyelembevételével, hogy manapság még az egyszerű személyautókban is mennyire elterjedtek a fedélzeti számítógépek és navigációs eszközök.



2. ábra: Dinamikus információs rendszerek (forrás: Közúti informatika jegyzet[1.])

Az információs rendszerek segítségével a következő feladatokat láthatjuk el:

- Kapacitáskihasználás növelése
- Munkamegosztás javítása
- Forgalomlefolyás javítása (torlódások elkerülése, felesleges útkeresések kiküszöbölése, utazási idő csökkentése)
- Környezeti károk csökkentése
- Közlekedési igény befolyásolása
- Biztonság növelése (balesetek számának és súlyosságának csökkentése)

A VJT-ket a dinamikus, kollektív, út menti információs rendszerek eszközeiként használják, melyek egyaránt alkalmasak pontszerű, vonali, hálózati és parkolás szabályozási feladatokra. A közlekedés során képesek az emberek aktuális információkkal és ajánlásokkal való ellátására.

Ezek az információk a következők lehetnek:

- Sebességkorlátozás
- Alternatív útvonalak ajánlása
- Hátralévő út megtételéhez szükséges becsült idő (főirányon és az alternatív lehetőségek esetén is, megkönnyítve ezzel a választást)
- Balesetekre való figyelmeztetés
- Forgalmi rend változása
- Közlekedésbiztonságot befolyásoló környezeti hatások
- Általános biztonsági információk (biztonsági öv, fényszórók,stb)
- Parkolási információk (szabad helyek száma, elhelyezkedése)
- Nem közlekedéssel kapcsolatos üzenetek (üdvözlő üzenetek)

Az egyre szigorúbb ellenőrzéseknek és egyre komolyabb mértékű büntetéseknek köszönhetően tovább erősödik a táblák által nyújtott ajánlások elfogadása, hiszen most már nem csak a vezető biztonságérzetén múlik, hogy beköti-e a biztonsági övet vagy betartja-e a sebesség korlátozást. Nem csak az jelent visszatartó erőt, hogy fél-e egy esetleges balesettől és szeretné-e elkerülni annak következményeit, hanem az is, hogy komoly pénzbírságra, vagy akár a vezetői engedélyének bevonására is számíthat, ha nem tartja be a szabályokat.

A változtatható jelzésképű táblák típusai[1.]:

- Mechanikus
  - Prizmás
  - Forgó lapos
  - Rolós
  - Forgó lamellás
- Fénytechnikán alapuló
  - Izzós
  - LED-es

Ma már a közlekedésben szinte kizárólag LED-esVJT-ket használnak, mivel ezek teljesítik legjobban a követelményeket. Tömegközlekedési járműveken megtalálhatóak még a forgó lamellás táblák, amelyekkel

- a járat számát
- az időt
- a következő megálló nevét
- az átszállási lehetőségeket

közlik az utasokkal. A forgó lamellás táblák esetén a tábla képpontjai egy-egy lapkából állnak, melyeknek egyik oldala jellemzően fekete, másik oldala fehér (vagy tetszőleges, a feketétől kellő mértékben eltérő színű). Az üzenet kirajzolásához használt képpontokat

a színes oldalukkal, míg a többit a fekete oldalukkal fordítják az informálandó közönség felé. A többi mechanikus VJT ezzel szemben csak néhány előre elkészített fix üzenetet tud váltogatni. A korszerűbb járműveken már szintén LED-es kijelezőket használnak erre a célra, de általában csak egyszínűeket.

A közlekedésben a VJT-knek, az elvárt élettartamon, környezeti hatásoknak való ellenálláson kívül, három alapvető követelményt kell teljesíteniük:

- Láthatóság
- Felismerhetőség
- Érthetőség

A láthatóságot fénytechnikai úton lehet megoldani, az üzenet kellő erejű megvilágításával, vagy az üzenet eleve fényt kibocsátó egységekből történő kirajzolásával. Forgalomszabályozásban leginkábbLED-esVJT-ket alkalmaznak, amelyeknél a megfelelő fényerejű diódák gondoskodnak a láthatóságról. A LED-ek sajátossága azonban, hogy viszonylag szűk sávban szórják a fényt, így a táblák nem megfelelő pozicionálása sokat ronthat a fényerőn.

A VJT-k helyét úgy kell megválasztani, hogy ne csak funkcionális szempontokat vegyünk figyelembe, hanem azt is, hogy a közlekedők számára, a VJT mennyire lesz észrevehető. Ezt elsősorban az út mellől belógó és a táblát eltakaró növényzet, valamint a háttér befolyásolja, mivel a rossz szín és alak megválasztás esetén a táblák könnyen beleolvadhatnak a háttérbe és a közlekedők nem, vagy csak későn veszik észre őket.

A VJT-ken megjelenített információ lehet szöveges, kép, vagy ezek kombinációja. Szöveg esetén figyelni kell, hogy lehetőleg ne tartalmazzon rövidítéseket, amennyire lehetséges több nyelven is jelenítsük meg az információt (de csak amíg ez nem megy az olvashatóság rovására), igyekezzünk minél tömörebben, minél rövidebben, de mindig egyértelmű információt szolgáltatni. Képek esetén célszerű egyszerű, nemzetközileg egységes, mindenki számára könnyen értelmezhető ábrák, piktogramok használata. Ha egyszerre jelenítünk meg képet és szöveget is, akkor pedig a két információ tartalma mindig legyen fedésben egymással. KRESZ táblák megjelenítésére is alkalmasak, ezeknél azonban figyelni kell, hogy a kivetített kép pontosan megegyezzen az eredetivel és amennyiben az út mellett hagyományos KRESZ tábla is található, akkor annak ne mondjon ellent, vagy ha ideiglenes forgalmi rend változásra hívja fel a figyelmet és emiatt jelez ki eltérő táblát, akkor lehetőség szerint adjon indoklást a változásra.

A VJT-k jellemzően rendelkeznek saját számítógéppel, ami valamilyen kisebb teljesítményű alaplapból (és processzorból) valamint grafikus meghajtóból áll, de mind egy-egy, az adott területért, létesítményért, útvonalért felelős központra vannak csatlakoztatva, és ezekből a központokból hálózaton keresztül történik a vezérlésük. Legjellemzőbb felhasználási területük a közlekedésen belül autópályákon, forgalmi és balesetvédelmi (dugó, baleset, nem megfelelő környezeti körülmények, biztonsági öv) információk szolgáltatása, városi forgalomban sebességszabályozás, ritkább esetben idő, hőmérséklet, esetleg levegőszennyezettség mértékének kijelzése. Ezeken kívül jól használhatóak parkolókban is, szabad parkolóhelyek számának, esetleg helyének megadására. Magyarországon a gyorsforgalmi útvonalakat 2006-ban kezdték el felszerelni VJT-kkel, ekkor az Állami Autópálya Kezelő Zrt. egy 2,5 milliárd Ft értékű projekt keretében 40 db mátrixos táblát telepítetett az igazgatása alá tartozó gyorsforgalmi úthálózat vonalán[6.].

4. Gazdasági vonatkozás

A KTI és a Levegő Munkacsoport – A közúti és vasúti közlekedés társadalmi mérlege[2.] című tanulmánya alapján,Magyarországon az államháztartás közlekedési mérlege 2004 és 2008 között – a bevételeket és kiadásokat tekintve – 730 és 1400 milliárd forint közötti tételeket tartalmaz.Tehát évente mind a bevételi oldalon, mind a kiadási oldalon átlagosan 1000 milliárd forintos pénzmozgásról lehet beszélni. Ezek az összegek több területet érintve jönnek össze. A bevételi oldalon jellemzően a gépjárművek birtoklásához és üzemeltetéséhez kapcsolódó adók állnak, melyeken forgalomtechnikai eszközökkel nem áll módunkban változtatni, vannak azonban olyan tételek is, melyek átgondolt és jól szervezett forgalomirányítással és szabályozással jelentősen javíthatók.

4.1 Forgalomtechnikai eszközökkel növelhető bevételek

- Autópályadíj
- Európai emisszió-kereskedelmi rendszer
- Osztalékbevételek
- EU támogatások
- Munkát terhelő adók és járulékok

A, Autópályadíjak

Az éves összbevételhez viszonyítva az autópályadíjakból származó bevétel viszonylag kisebb tételt jelent, 2006-ban összesen 28,5 milliárd forint volt és még a pótdíjakkal sem érte el a 30 milliárd forintot, ettől függetlenül nem szabad elfeledkezni erről sem. A VJT-ket túlnyomórészt autópályákon alkalmazzák a kapacitások jobb kihasználására és a forgalom gyorsabb, zavartalanabb levezénylésére. Az ezen területeken elért eredmények mindenképp növelni fogják az autópályákat igénybevevők számát, ami az autópályák használatáért fizetendő díjakból származó bevétel növekedését fogja maga után vonni.

B, Európai emisszió-kereskedelmi rendszer

A kiotói szerződés értelmében Magyarország vállalta, hogy csökkenti az üvegházhatású gázok kibocsátását. Ennek egyik módja, hogy a közúti közlekedésben csökkentjük a fölöslegesen a hálózatokon töltött időt (cirkálások parkolóhelyeket keresve, dugóban állás). A VJT-k segítségével az utazási idők lényegesen csökkenthetők elsősorban kevésbé terhelt alternatív útvonalak ajánlásával. Ezek a csökkenések önmagukban kismértékűek, azonban ha figyelembe vesszük, hogy Magyarországon az üvegházhatású gázok 97%-ának kibocsátásáért a közúti közlekedés a felelős, akkor összességében jelentős javulás érhető el. A szerződés értelmében pedig azok az országok, amelyek képesek voltak ezt az értéket a szerződésben meghatározott határérték alá csökkenteni, a

fellépő különbséget (határérték – teljesített érték) eladhatják olyan országoknak, akiknek még nem sikerült teljesíteni e kötelezettségeiket. Így tehát ezen a téren az állam plusz bevételre tehet szert.

C, Osztalékbevételek

Az államnak részesedése van több, közlekedéshez kapcsolódó tevékenységű vállalatban. A közlekedés gyorsabb és gazdaságosabb lefolyása csökkenti ezen vállalatok gépjármű üzemeltetésre fordított kiadásait, aminek következtében profitjuk növekszik, így a tulajdonosok magasabb osztalékhoz juthatnak.

D, EU támogatások

Az EU támogatásokat a legtöbb tanulmány óvatosan kezeli, hiszen ezeket a pályázati pénzeket abból a keretből fizetik ki, amiket a tagországok együttesen befizettek. A pályázatokon elnyerhető támogatások mértéke azonban nem függ, az adott tagország által befizetett összegtől, tehát egy rosszul pályázó ország lehet, hogy semmit nem lát viszont a befizetett pénzéből, míg egy jó pályázatokat benyújtó ország az általa befizetett összegnek akár többszörösét is elnyerheti ilyen támogatások formájában. A közlekedés elengedhetetlen az emberi társadalom jelenlegi berendezkedésének fenntartásához, tehát ezen a területen mindig jó eséllyel lehet támogatásokat nyerni a fejlesztésekre és bővítésekre. Ennek értelmében az úthálózat bővítése mellett a forgalomtechnikai infrastruktúra fejlesztésére is törekedni kell, mert jól kidolgozott projektekkel egyre több EU pályázaton keresztül nyerhető el támogatás.

E, Munkát terhelő adók és járulékok

Közvetlenül számtalan munkahely kapcsolódik a közlekedéshez (pl.: hivatásos sofőr), közvetett módon pedig lényegében az összes munkahely függ a közlekedéstől. Minél gyorsabb és probléma-mentesebb a közlekedés, a cégek annál gyorsabban jutnak alapanyagokhoz, felszerelésekhez és annál gyorsabban tudják értékesíteni a saját termékeiket, miközben a munkavállalók is hatékonyabban tudnak dolgozni (kevesebb szabadidejük megy el a közlekedéssel, kevesebb stressz éri őket a forgalmi zavarok miatt stb). Így tehát a forgalom fejlődése és romlása kihatással van a cégek teljesítőképességére. Minél jobban fejlődik a közlekedés, a cégek annál könnyebben tudnak működni és annál nagyobb bevételre számíthatnak. A bevételek növekedésével pedig növekednek az államnak fizetendő adók és járulékok, amelyek ez által növelik az állam bevételeit.

1. Táblázat: Munkát terhelő adók és járulékok

2006-ban a közvetlenül közúthoz kapcsolódó cégek által befizetett adók	
Adófajta	összeg (milliárd Ft)
Személyi jövedelemadó	165,6
Nyugdíjbiztosítási járulék	159
Egészségbiztosítási járulék	111,6
Egészségügyi hozzájárulás	12,2
Szakképzési hozzájárulás	3,8
Munkaadói járulék	20,2
Munkavállalói járulék	7,1
Összes	479,5

A táblázatból látható, hogy az állam összesen 479,5 milliárd Ft adóbevételre tett szert. Ez még államháztartási szinten is jelentős összegnek tekintendő, ami bizonyítja, hogy ezen a területen a legkisebb javulás is érezhető bevétel-növekedést eredményez.

4.2 Forgalomtechnikai eszközökkel csökkenthető kiadások

- Környezetszennyező és –károsító hatások
- Baleseti veszteségek

A, Környezetre gyakorolt hatások

Ezek a hatások közvetlenül nem jelennek meg a költségvetésben, azonban az élet egyéb területein jelentős változásokat okoznak, ami bevételcsökkenéseket és kiadásnövekedéseket eredményez. Számszerűsítésük különböző mérések, vizsgálatok és becslések alapján történik.

A közlekedés az elhasznált fosszilis üzemanyagok következtében jelentős mennyiségű CO2-t bocsát ki, ami az üvegházhatást növelve éghajlati változásokat eredményez. Becslések szerint ezzel a személyautók évi átlagosan 33.8, a teherautók 26.7, a buszok pedig 2.2 milliárd forint kárt okoznak.

A levegőbe kerülő szennyező anyagok nem csak az éghajlatra vannak hatással, de az emberek és más élőlényeg egészségére is. Rövidül az élettartam, romlik az egészség, ezáltal csökken a munkaképesség és növekednek az egészségügyi költségek. Ezen a

területen a személyautók körülbelül 58.6, a közúti teherszállítás 108.6, míg a buszok 9.1 milliárd Ft költséget okoznak évente.

Az üzemanyag elégetése során kibocsátott anyagok nem csak a levegőt szennyezik, de a vizeket és a talajt is, ami a vízi élővilág és a növények pusztulását idézi elő. A személyautók évente átlagosan 6.4, a teherszállító járművek 8.4, a buszok pedig 1.7 milliárd Ft költséget jelentenek.

A káros anyagokon kívül a járművek jelentős zajt is kibocsátanak, amelynek szintén jelentős egészségkárosító és életminőség csökkentő hatása van. A zajszennyezésből eredő költségek személyautók esetén évente átlagosan 15.6, tehergépkocsiknál 25.6, buszoknál 0.7 milliárd Ft-ot tesznek ki.

A negatív externális hatások közé tartozik még a természetes élőhelyek feldarabolódása és pusztulása. Ezt a hatást azonban forgalomtechnikai eszközök segítségével nem lehet csökkenteni, hiszen ezek az eszközök a meglévő úthálózat jobb kihasználására irányulnak, nem pedig a meglévő forgalom minél kisebb úthálózaton történő lebonyolítására. A több hatás viszont jelentősen csökkenthető azáltal, hogy csökkentjük a járművek fölöslegesen a hálózaton töltött idejét, erre pedig tökéletesen alkalmasak a VJT-k.

B, Baleseti veszteségek

A baleseti költségek több darabból tevődnek össze:

- Személyi sérülések költségei
- Tárgyi sérülések (jármű, rakomány sérülése)
- A baleset által előidézett kapacitáscsökkenésből, torlódásokból származó járulékos anyagi károk

A balesetek jelentősmértékű aluljelentettségéből fakadóan nem állnak rendelkezése pontos értékek, így a felmerülő költségeket is csak becsülni lehet. A fenti három elemre a balesetek költsége 2006 a következőképpen alakult:

2. Táblázat: Balesetekből származó költségek

Balesetek költsége 2006-ban	
Baleset típusa	Költség (milliárd Ft)
Baleseti (személyi sérüléssel kapcsolatos)	600
Anyagi	130
Externális	470
Összes	1200

Látható, hogy a balesetek jelentős tételt képeznek a költségvetésben, összesen körülbelül 1200 milliárd Ft-ot, így a balesetek számának és súlyosságának csökkentésével komoly megtakarításokat lehet elérni. Ennek egyik mondja a közlekedés szabályrendszerének és annak betartására vonatkozó ellenőrzések szigorítása, ez viszont állami feladat. Másik hatékony módszer azonban, amit VJT-k segítségével eredményesen lehet kivitelezni, ha felhívjuk a közlekedők figyelmét a nem megfelelő haladási sebességre, vagy ha tájékoztatjuk őket a közlekedést hátrányosan befolyásoló környezeti hatásokról, illetve a forgalom alkalmi változásairól. Az esős időben rosszul felmért vezetési képességek vagy magának az időjárás súlyosságának hibás felmérése, illetve a nem várt forgalmi változások (baleset, torlódás, sávlezárás) a balesetek jelentős százalékáért felelősek. Ezeket a VJT-ken keresztüli megfelelő tájékoztatással és figyelmeztetéssel nagymértékben csökkenthetjük.

5. VJT-k hatása a forgalomra

5.1 Külföldi irodalom

Mark G. M. Brocken és Martie J. M. Van der Vlist a Ring Road-on hajtottak végre vizsgálatokat, melynek eredményét a TrafficControlwithVariableMessageSigns[3.]című cikkükben publikálták. Az autópálya Amsterdam köré épült és 1990 szeptemberébenkészült el. Kiválóan alkalmas kipróbálni a táblákat, mivel elég sok lehetőséget kínál más utakon eljutni a célhoz, illetve található rajta két alagút:Coentunnel (2x2 sávos) és Zeeburgertunnel (2x3 sávos).

A, Szimulációs tanulmány témája:

- A táblák hatása az útvonalválasztásra és a vezetési magatartásra
- Közlekedési folyamatok szimulálása, a közlekedők számára, VJT-kkel biztosított dinamikus információk esetén

A körgyűrű néhány részén jelentkező nagyon magas forgalom és a nagyszámú alternatív útvonal miatt, ideális lehetőséget biztosított a dinamikus útajánlásokra.

Szükséges információ a táblák kihelyezése előtt:

- hogyan: hogyan lehet olyan útvonalajánlásokat tenni, hogy a közlekedők kövessék azokat?
- hol: a táblákat olyan helyen kell elhelyezni, ahol a közlekedőknek 2 hasonló lehetőségeket nyújtó útvonal közül kell választaniuk
- mikor: a közlekedők és rendszer operátorok szempontjából milyen forgalom állapotok esetén kell vagy célszerű ajánlásokat tenni?

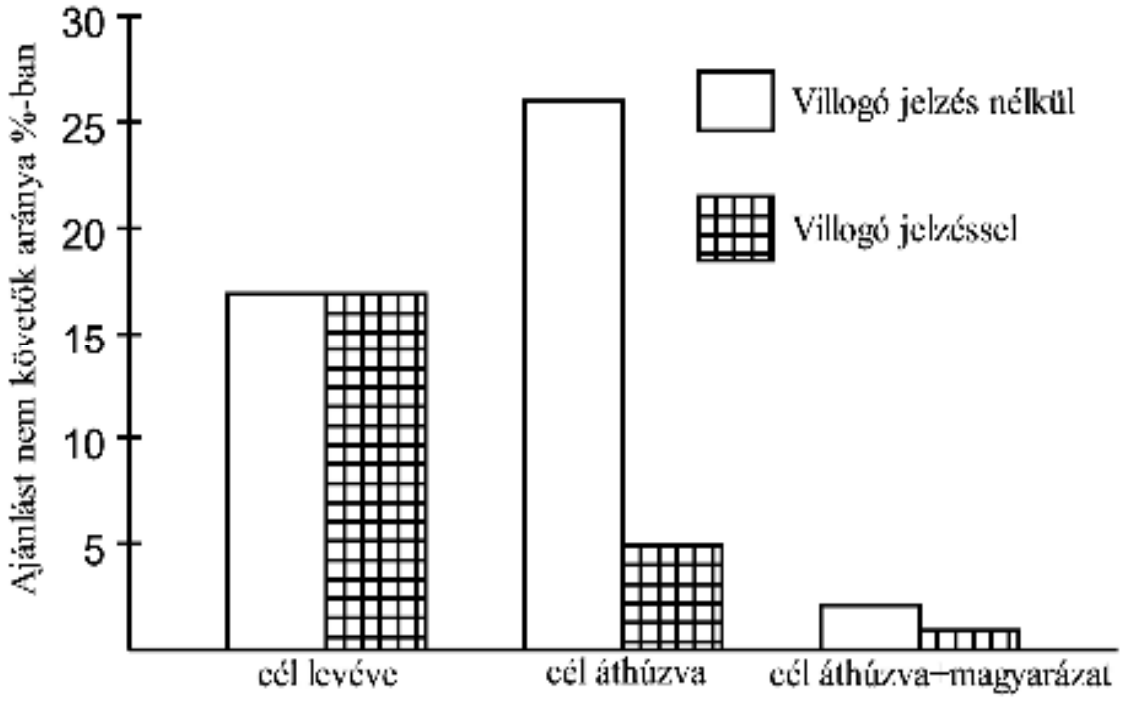
B, Hogyan

A „hogyan” kérdésre 3 alternatívát dolgoztak ki:

- az egyenesen tovább haladó iránynál eltüntették és a lehajtó sávot jelző táblán tüntették fel a kritikus állomást
- az egyenesen haladó sáv felett áthúzták, a lehajtónál pedig kiírták a kritikus állomást
- a második módszer mellett még magyarázatot is adtak, hogy miért ez az ajánlás

Ezeken kívül használtak egy sárga villogójelzést is, a kihajtónál. Így összesen 6 féle variációt próbáltak ki.

A vizsgálatok a következő eredményt hozták:



3. ábra: Útvonalajánlás elfogadottsága (forrás: Mark G.M.Brocken, Martie J.M.van der Vlist–TrafficControlwithVariableMessageSigns[3.])

A diagramon jól látható, hogy amikor a tovább haladó sávból csak eltüntették a kritikus célállomást, akkor a közlekedők kb 17%-a hagyta figyelmen kívül az ajánlást és ezen nem változtatott a kijárati táblán elhelyezett sárga villogó jelzés sem. A legrosszabb eredménnyel az áthúzásos módszer zárt, ennél 27% körül volt az ajánlással nem foglalkozó közlekedők száma, azonban miután a kijárati táblán elhelyezték a villogó jelzést, ez az érték az előző variáció alá javul és alig 5%-ra csökkentette ezt a számot. A legjobb hatást az utolsó variációval érték el, amikor nem csak áthúzták az adott célállomást a továbbhaladási sávot jelző VJT-n, de információt is adtak, hogy miért tették ezt. Ezzel a módszerrel 3% alá sikerült szorítani a nem engedelmeskedők számát. A sárga villogó ebben az esetben is javított az adatokon, de csak alig 1%-kal.

C, Hol

A „hol” kérdésre válaszolván az egyértelmű, hogy ott, ahol nem egyértelmű, melyik választás a jobb. Ehhez azonban el kell dönteni, hogy mitől lesz jobb az egyik, vagy a másik választás. A forgalomtechnikai szempontok helyett inkább a közlekedők szempontjait vették figyelembe. A legfontosabb paraméterek az útvonalválasztásnál az eljutási idő, távolság és a szűk keresztmetszet. Az egymástól való viszonylagos függetlenségük miatt azonban ezeket nem egyszerű együttesen kezelni. Az eljutási időt kihagyták a vizsgálatból, mivel sokkal relevánsabbnak találták a szűk keresztmetszet okozta késést, mint magát az elméleti utazási időt. Ennek alapján véleménykutatásokat végeztek és arra az eredményre jutottak, hogy az otthonuk és munkahelyük között ingázók 1,27km-t plusz utat hajlandóak megtenni azért, hogy 1 percet lefaragjanak a

késési időből. Autópályáról lévén szó, ez kevesebb, mint amennyit a megengedett sebességgel 1 perc alatt meg lehet tenni. Feltételezésük szerint ez a különbség a felmerülő plusz költségek miatt jelentkezik. Ez azt jelenti, hogy a felkínált alternatív útvonalon az utazási időnek, késési időnek jelentősen kisebbnek kell lennie, mint az eredeti, rövidebb úton haladva.

D, Mikor

A „mikor" kérdés vizsgálatakor több különböző paraméterű szimuláció lefuttatásával vizsgálódtak. Nézték az utazási időt, a megtett távolságot, az átlagsebességet és az elhasznált energiát.

Összesen 9esetet vizsgáltak:

1. alapszituációnak a reggeli 2 órás csúcsidőszakot választották a Coentunnel-en keresztül, VJT nélkül (BASE)
2. még mindig nem használtak VJT-t, de a forgalmat lecsökkentették az alap 70%-ra (1200jm/ó), viszont létrehoztak egy 20 perces fennakadást (CT0)
3. ugyan az, mint a második eset, de üzembe helyeztek egy VJT-t, aminek az ajánlásait a közlekedők 30%-a fogadta el (CT30)
4. az elfogadást 60%-ra növelték (CT60)
5. az elfogadást 100%-ra növelték (CT100)

6-9. (ZT0/30/60/100) megegyezik a 2-5. pontokban vázolt variációkkal, azzal a különbséggel, hogy a Coentunnel helyett a Zeeburgertunnel-nél vizsgálták a forgalmat.

3. Táblázat: Szimulációk eredményei (forrás: Mark G.M.Brocken, Martie J.M.van der Vlist – TrafficControllwithVariableMessageSigns[3.])

	Órák (*1000)	Kilométerek (ezer kilométer)	Átlagsebesség (km/h)	Energia felhasználás (ezer liter)
Alap	8,4	781	93	71,8
CT0	10,6	781	74	74,8
CT30	10	787	78	74
CT60	9,8	790	81	73,4
CT100	9,8	791	81	73,1
ZT0	8,9	781	87	71,9
ZT30	9	781	87	72,2
ZT60	9,1	782	86	72,8
ZT100	9,3	783	85	72,4

Összességében arra a következtetésre jutottak, hogy a közlekedőknek minél nagyobb része követte a VJT-ken jelzett ajánlásokat, annál inkább javultak ezek az értékek.

5.2 Belföldi irodalom

Tettamanti Tamás diplomamunkájában[4.] egy másik lehetőséget vizsgált meg. A VJT-s rendszerek nem csak önmagukban képesek hatékonyan szabályozni a forgalmat,hanem más rendszereket kiegészítve is hasznosnak bizonyulnak. Jó példa erre a fejlett nyugati országokban már régóta használatos felhajtókorlátozás (ramp-metering). Felhajtókorlátozás alkalmazásakor az alapelv az, hogy az autópályán haladó forgalmat védik. A pályán haladó forgalom zavartalan lefolyása az elsődleges cél és ennek érdekében szabályozzák, hogy az adott felhajtón mikor és hány jármű léphet be a forgalomba.

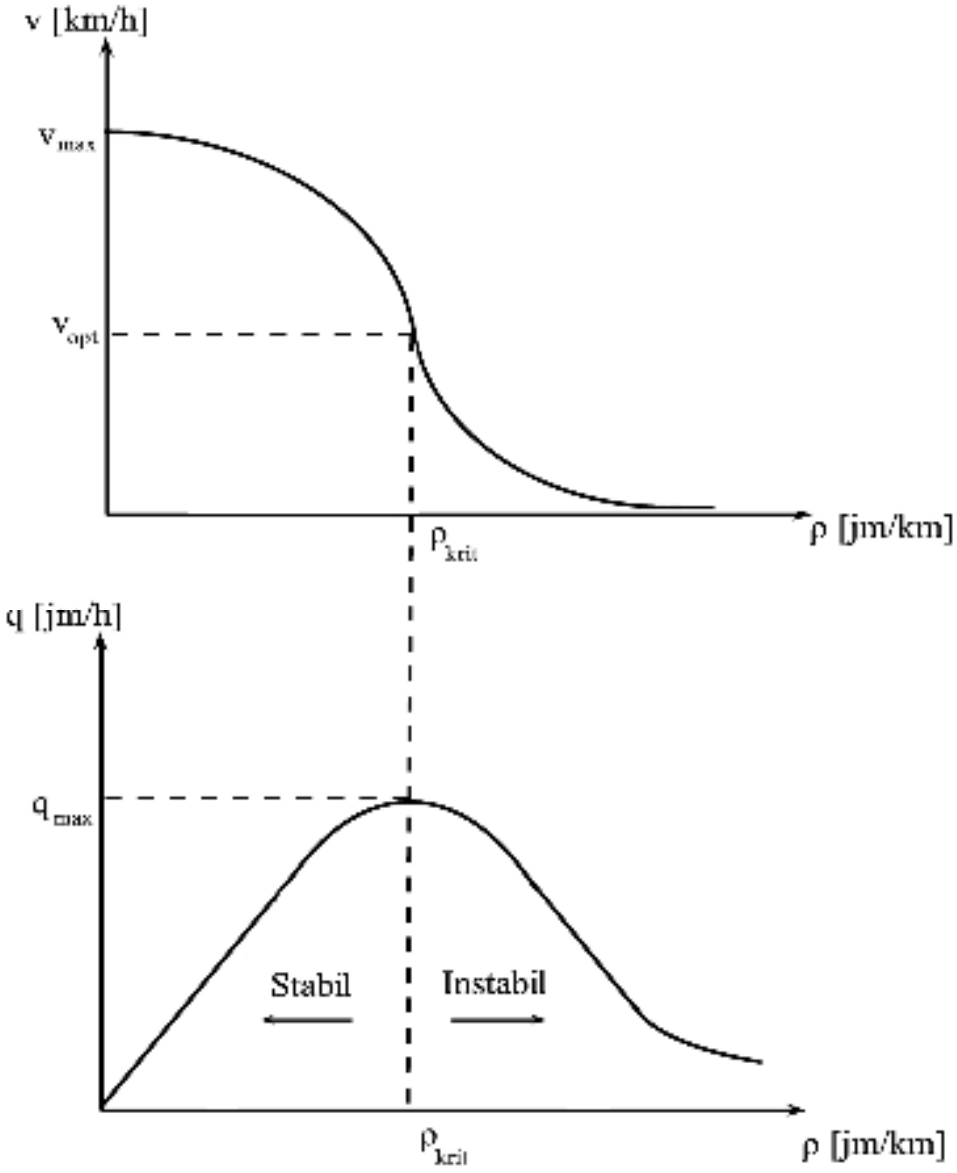
Erre három megoldás létezik:

- Fix idejű
- Lokális szabályozás
- Összehangolt szabályozás

Ez a módszer az autópálya telítettsége esetén a felhajtókon sorfelépülést okoz, viszont teljes lezárás nem megengedett, percenként minimum egy járművet fel kell engedni, így az autópályán haladóknak biztosított zavartalan haladás kompenzáljaa felhajtani szándékozók torlódását. A VJT-ket kétféleképpen lehet bevonni ebbe a rendszerbe:

- Változtatható sebességkorlátozás
- Alternatív útvonalak ajánlása

A változtatható sebességkorlátozást a fundamentális diagram felhasználásával építik fel.



4. ábra: Fundamentális diagram

Ennek értelmében a megengedett sebesség változtatásával igyekeznek a forgalom sűrűségét a kritikus értéken tartani. Ez a forgalomsűrűség biztosítja a legnagyobb átbocsátóképességet. Ennél kisebb sűrűség esetén a forgalom stabilabb lesz, biztosan nem alakul ki torlódás, viszont csökken a kapacitása, hiszen kevesebb autó fér fel a hálózatra. Nagyobb sűrűség esetén ugyan több jármű lesz a hálózaton, viszont a forgalom instabillá válik és torlódás alakul ki, ami miatt szintén csökken az átbocsátóképesség.

Ez a módszer önmagában nem okoz jelentős kapacitásnövekedést, a felhajtószabályozással összehangolva azonban jelentősen jobb eredmények érhetőek el, mint külön-külön használva őket.

Az alternatív útvonal ajánlása ebben az esetben a felhajtani szándékozó közlekedők számára lehet segítséget nyújtó megoldás. Afelhajtókorlátozást vezérlő logikát úgy oldják meg, hogy ne tiltsa meg teljesen a felhajtás lehetőségét és minél kisebb torlódást eredményezzen a felhajtón.Csúcsidőben azonban, amikor az autópálya sűrűsége a kritikus sűrűséget elérte vagy már meg is haladva instabillá válik a fogalom, akkor hatékony megoldást jelenthet, ha az autósokat megpróbáljuk más útvonalra terelni.Ezzel elérhetjük, hogy ne a felhajtón várakozzanak, amíg beengedik őket, hanem, ha van rá lehetőség, akkor a hálózat kevésbé használt vonalait próbálják meg használni. Amíg ezeken elérik a célállomásukat, vagy legalább a következő felhajtót,addig van rá esély, hogy az autópályán javul a forgalom stabilitása.

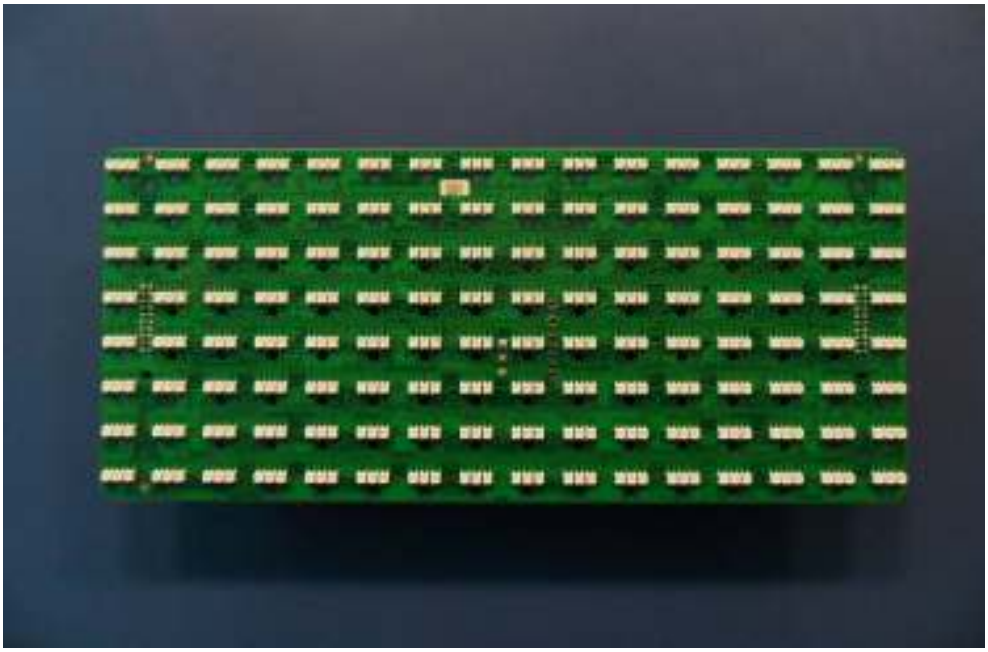
6. Swarco Futurit táblák

Az általam használt VJT-ket a Közlekedésautomatikai Tanszék vásárolta a SwarcoTrafficHungariatól[5.]. Mindkét tábla LED-es és színes, mátrix felépítésű. A LED-eklehetővé teszik a jó láthatóságot és hosszú élettartamot. Az egyszerűbb VJT-knél mindösszesen néhány előre meghatározott és nem módosítható jelzési kép érhető el. Ennek lényege az olcsóság, hiszenelegendő csak annyi képpontot használni, amennyivel ez a néhány jelzéskép megjeleníthető. A mátrixos felépítés ezzel szemben egy lényegesen bonyolultabb, drágább, viszont sokkal nagyobb szabadságot biztosító rendszer, hiszen a képpontok számától függően tetszőleges jelzéskép állítható elő. A nagyobbik VJT96x64, a kisebbik 64x64 képpont felbontású.

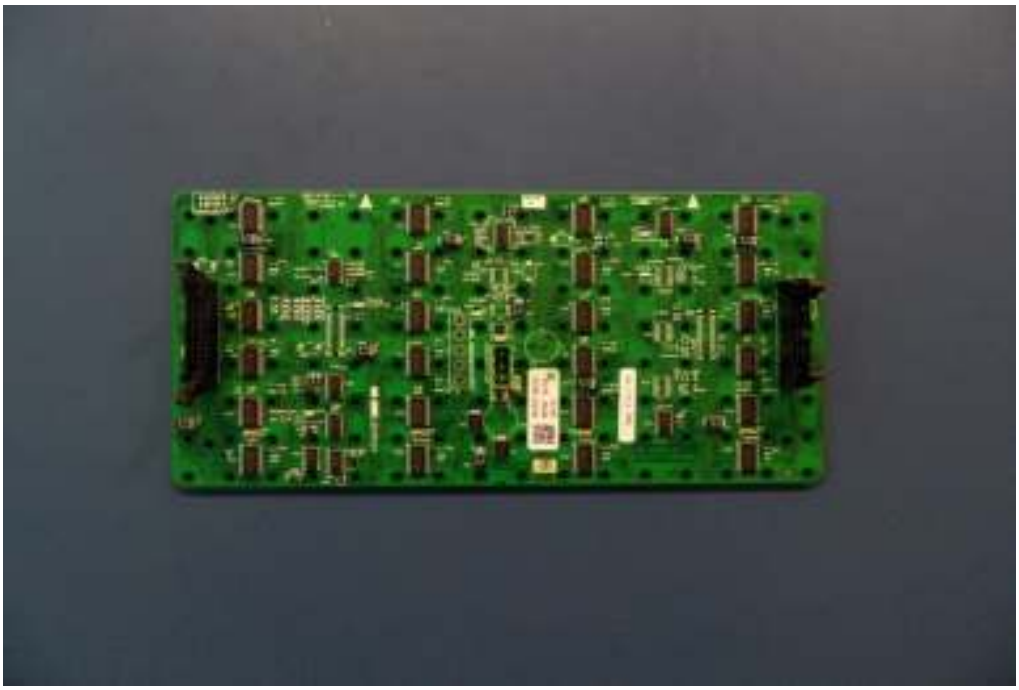
5. ábra: LED-ek előtti lencsék a kijelzőn

Minden képpont 1-1 egymás mellett, egy közös, speciális lencse alatt elhelyezkedő piros, zöld és kék fényt kibocsátó diódából áll, melyek 256 féle színárnyalat megjelenítésére képesek. A Swarco nagy hangsúlyt fektet rá, hogy az általuk forgalmazott VJT-k a lehető legmagasabb minőségi besorolást érjék el, különös tekintettel a fényerősségre és fényeloszlásra. Ennek érdekében a legjobb minőségű LED-eket használják és fényerősség szenzorokat is beszereltek a táblák házának elő- és hátlapjába, hogy a környezethez folyamatosan alkalmazkodva a legmegfelelőbb beállításokkal működjenek a táblák. További érdekesség, hogy kutatások eredményeire alapozva és a KRESZ táblákból kiindulva, VJT-ken többnyire piros, vagy részben piros árnyalatú színeket használnak az információk megjelenítésére, ami a piros LED-ek fokozott igénybevételét eredményezni. Ehhez alkalmazkodván a piros LED-eknél egy tokba két fénykibocsátó diódát alakítottak ki, ezzel biztosítva a közel azonos élettartamot, a nagyobb igénybevétel ellenére. Felépítésüket tekintve az egész elektronika egy zárt, a környezeti hatásoknak ellenálló alumínium házban kapott helyett, melyek hátulján található 2-2 speciális kulccsal nyitható ajtó, amiken keresztül hozzáférhető az elektronika.

6.1 LED-eket tartalmazó panel



6. ábra: LED-eket tartalmazó panel előlapi oldala a felületszerelt LED-ekkel



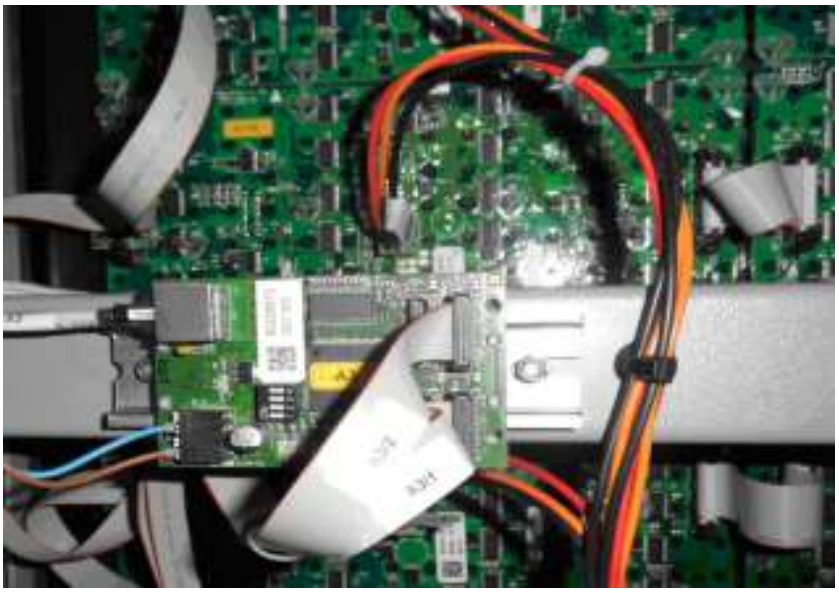
7. ábra: LED-es panel hátsó oldala a csatlakozókkal

A kisebbik LED fal 4x8, a nagyobbik 6x8 db, egyenként 16x8 képpontot tartalmazó panelből épül fel, ami jelentősen meggyorsítja a szerelhetőséget. Minden panel hátlapján (a tábla háta felé néző oldal) találhatóak állapotjelző LED-ek, amelyek jelzik, hogy a panel áram alatt van, illetve azt is, ha megszakadt a kapcsolat a táp, vagy a vezérlő egység között és azt is, ha az adott lapon valamilyen meghibásodás történt. Ezek pusztán a karbantartó személyzet számára jelentenek információt, de emellett a táblákban lévő számítógépnek folyamatosan ellenőriznie kell, hogy minden LED

megfelelően működik-e és azonnal hibajelentést kell küldenie, ha valamelyiket ki kell cserélni. Minden panelen három csatlakozó található. Középen van a tápcsatlakozó, a két végén pedig egy-egy szalagkábeles aljzat segítségével lehet biztosítani a kapcsolatot a vezérlő és a panelek között. Minden sorban az első lap bal szélső csatlakozóján keresztül kapcsolódik a lentebb látható vezérlőhöz, a jobb oldalon keresztül pedig a sorban mögötte található panelhez, így ezek soronként egymás után sorosan, oszloponként pedig párhuzamosan vannak kötve.

6.2 Paneleket kiszolgáló vezérlő

A panelek elé elhelyezett vezérlő biztosítja, hogy a PC-től érkező jelek a megfelelő sorba és ott a megfelelő panelra menjenek. A PC-től UTP kábelen keresztül kapja az információkat, majd azokat szalagkábeleken keresztül juttatja el a megfelelő panelre. A vezérlőből 2 db szalagkábel indul ki, ezen történik a panelek címzése és az adatátvitel is. A szalagkábel lábkiosztásáról nincs információnk, így nem tudjuk maximálisan hány panel lehet egy sorban, de mind a 64x64-es, mind a 96x64 tábla esetén elegendő 1 db vezérlő és mind a két esetben ugyan úgy van az egyik kábelre a felső 4, a másikra az alsó 4 sor panel kötve.



8. ábra: Panel vezérlő

6.3 Fedélzeti számítógép

9. ábra: A táblák házának sarkában kapott helyett a beépített számítógép

A VJT-kben található egy fedélzeti számítógép, ami a LED-ekkel együtt a beépített tápegységektől kapja a működése számára szükséges tápfeszültséget. Ez a gép hálózaton keresztül képes kapcsolódni egy központhoz, hogy onnan vezérelhessék, de ennek hiányában, akár önmaga is képes működtetni a táblát. Erre a PC-re beágyazott Windows XP operációs rendszer került, megtisztítva minden fölösleges tartalomtól, hogy elegendően kis helyen elférjen és a gép háttértáraként szolgáló Flash kártyán maradjon hely a szükséges programoknak és a feltöltött tartalmaknak is.

A gép három részből áll:

- Alaplap
- Grafikus kártya
- PC

A, Alaplap és grafikus meghajtó

10. ábra: Alaplap és grafikus kártya

A képen az alaplap és közvetlen fölötte a grafikus kártya látható. A grafikus kártya a PC-től kapott adatok alapján megtervezi a kijelzendő képet, meghatározza, hogy mely LED-eknek kell égniük és milyen intenzitással, hogy a kívánt információ a kívánt színben jelenjen meg a táblán. Miután végzett a számításokkal, az adatokat átadja az alaplapi kártyának, amiről UTP kábel segítségével jutnak el a jelek a LED-eket tartalmazó panelekig. Ezen a kártyántalálható egy B típusú USB csatlakozó, valamint egy UTP csatlakozó. Ezeken keresztül biztosított az összekötés a PC-vel.

B, PC

11. ábra: Wafer LX3 R20 pc kártya

A VJT-k egy-egy Wafer(Embedded PC) LX3 R20 típusú integrált számítógépet kaptak. A szerelt hűtőbordákkal a kártya -40 és +85ºCközötti hőmérséklet tartományban képes működni. A kártya egy AMD Geode LX800 típusú 500MHz-es processzorral és 512 MB-nyi integrált memóriával rendelkezik. Ez végzi el a kommunikációs feladatokat és az utasítások végrehajtásához szükséges számításokat, majd elküldi a megfelelő információkat a grafikus kártyának. A kártyán található két UTP port, ebből az egyik a központi számítógéppel való kapcsolatot biztosítja. A kártyán két A típusú USB port van, egyik a fennmaradó UTP-vel együtt a grafikus kártyával való kapcsolatot szolgálja, a másik pedig jellemzően közvetlen csatlakozás esetén valamilyen beviteli periféria (billentyűzet, egér) csatlakoztatását teszi lehetővé. A kártya rendelkezik olyan tüskesorral, ami lehetőséget ad további USB port csatlakoztatására, azonban ezek a szokásos 2,54 mm-es helyett, csak 2 mm-es lábtávolságúak, így csak speciális csatlakozóval lehet őket használni. A két UTP aljzat mellett a kártyán további hálózati kommunikációra alkalmas csatlakozó is található (2 db RS-232, valamint 1 RS-422 és 1 RS-485) és itt található a tápcsatlakozó is.

A PC kártya rendelkezik ATA típusú IDE csatlakozó fogadására alkalmas tüskesorral is, amire merevlemezes, vagy optikai meghajtó köthető, a VJT-kben azonban egy CF kártyát használnak háttértárként, aminek a processzor kártya hátoldalán található a foglalata. Ezek a kártyák kimondottan alkalmasak HDD-k helyette-sítésére, szilárd félvezető technológián alapulnak, vagyis nem tartalmaznak mozgó alkatrészeket, aminek köszönhetően lényegesen vékonyabbak a hagyományos merevlemezes meghajtóknál, sokkal kisebb az energiafogyasztásuk és a hőtermelésük, valamint sokkal

12. ábra: CF kártya

ellenállóbbak a fizikai hatásokkal szemben, emiatt kimondottan alkalmasak az ilyen jellegű felhasználásra. A VJT-k használata során nem jelentkezik igény különösebben nagyméretű adatok tárolására vagy nagy sebességű írási, olvasási műveletre, így nem okoz problémát, hogy ezekben a jellemzőkben messze elmaradnak a merevlemezes háttértáraktól.

A táblákhoz kétféleképpen lehet csatlakozni. Egyrészt közvetlenül a helyszínen, másrészt Ethernetes hálózaton keresztül.

A helyi kapcsolódáshoz rendelkezésre áll egy D-SUB port monitor csatlakoztatásához, valamint egy szabadon hagyott USB port egy input periféria számára. Ez utóbbi okozott némi problémát, mivel nem állt rendelkezésemre trackball-al (fordított egér) egybeépített billentyűzet, és bár az XP operációs rendszerben való navigálás nem okoz problémát az ismert billentyű kombinációkkal, a táblákhoz kapott SSC-ben (Sign Service Center) és a tesztprogramban, már gondot jelentett az egér hiánya. A bővítésre lehetőséget adó tüskesor lábkiosztásának meghatározása nem okozott problémát, azonban nem rendelkeztem kisebb lábkiosztáshoz használható csatlakozókkal, így ezt a lehetőséget nem tudtam kihasználni.

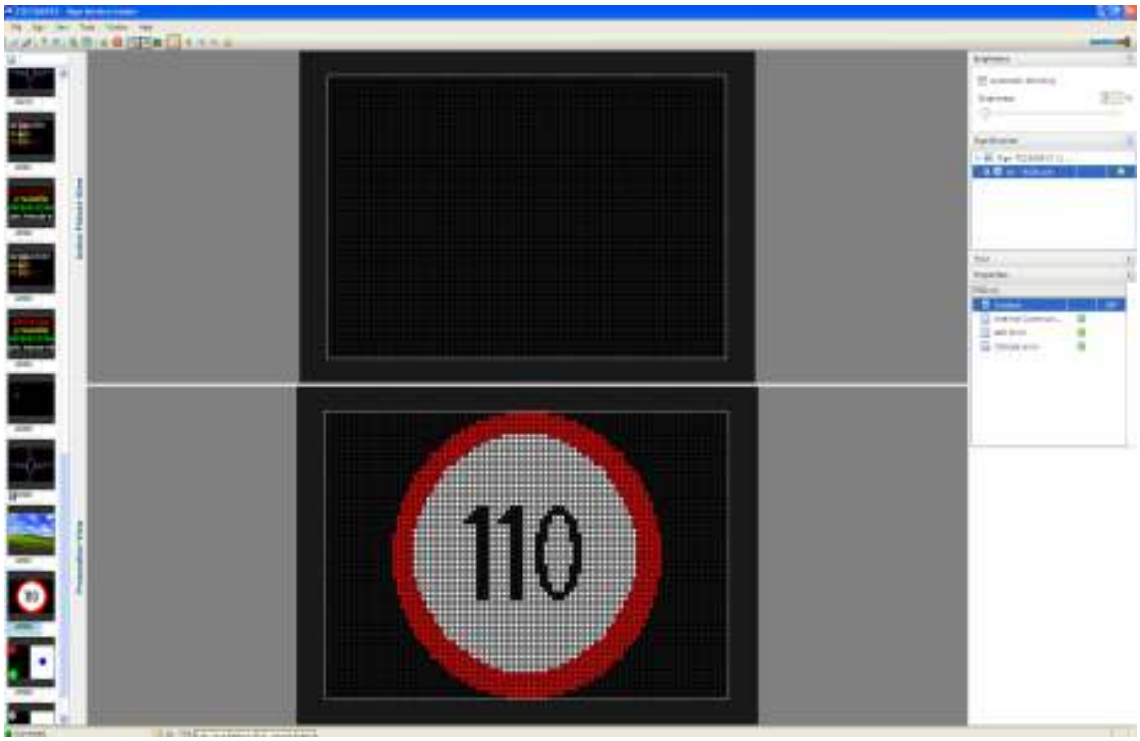
Hálózaton keresztüli kapcsolat esetén további két lehetőség van. Az egyik, hogy távoli asztal szolgáltatás segítségével csatlakozik a felhasználó a táblákra, mintha közvetlenül a VJT-kre kapcsolódna. Ezzel a módszerrel lehetőség van távolról is módosítani a VJT operációs rendszerének beállításait, telepíthet és eltávolíthat programokat, valamint tartalmakat. A másik módszer a Swarcotólkapott SSC szoftver használata.

7. Sign Service Center (SSC, [12.])

Az SSC-t a Swarcotól kapta a tanszék a táblákhoz, hogy ezzel lehessen vezérelni őket.Azonban elég körülményesen használható és a táblák kommunikációs protokollja által nyújtott lehetőségeknek csak egy részét fedi le, míg néhány hasznos funkció elérhetetlen vagy nem elég jól kihasználható a segítségével.

7.1 Felhasználói felület

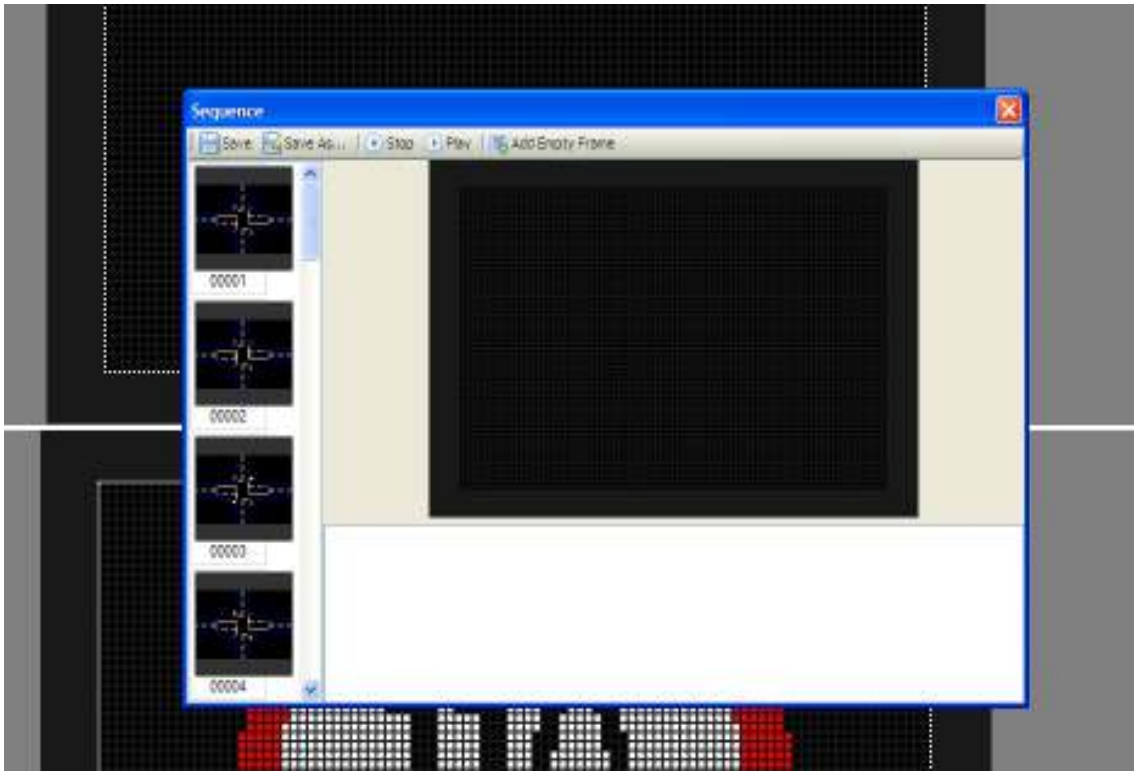
Az SSC elindításakor először egy project fájlt kell megnyitni. Ezeket a gyártó előre elkészítette, külön a nagy és külön a kis táblához. Ezekben vannak eltárolva a tábla fizikai adatai (pl. felbontás) és az IP címe, valamint a kommunikációhoz használt port is, de ezeket a programon belül lehetséges módosítani. Fontos viszont, hogy itt csak azt lehet átállítani, hogy a program milyen címen és portonpróbáljon csatlakozni, a tábla IP címét ez nem állítja át, tehát ha olyan címet ad meg a felhasználó, amilyet egyik hálózatra kötött tábla sem birtokol, akkor a kapcsolódás sikertelen lesz. Ezután be kell jelentkezni, ezt három féle, különböző szintű felhasználóval lehet megtenni. Az Operatoruser, az egyszerű felhasználó, meg tudja nézni, hogy milyen tartalom van kitéve a táblára, állíthatja a fényerőt, de nem módosíthatja a megjelenített információt. A következő szint a Poweruser, ezzel a szinttel már módosítható a tartalom és szekvenciát is létre lehet hozni. A legmagasabb szint az Administrator, ehhez nincs hozzáférésem, feltehetőleg ezen a szinten lehet új project-et létrehozni és egyéb funkciók is elérhetővé válnak. Ezek után, a szoftver a kiválasztott tábla IP címének és a használt port számának megadása után képes csatlakozni a táblákhoz és ezen a felhasználói felületen keresztül befolyásolható, hogy milyen információk jelenjennek meg a kijelzőkön.



13. ábra: SSC kezelőfelület

A képen az SSC szoftver grafikus felületének felépítése látható. Az ablak tetején hagyományosan egy menüsor található, alatta pedig egy ikon sor, a választott felhasználói szintnek megfelelő ikonokkal. A képernyő bal szélén, a táblán megtalálható tartalom listája látható, amiből kiválasztható, hogy melyiket szeretné aktiválni a felhasználó. Jobb oldalon a különböző beállítások és kapcsolati információk találhatóak, itt lehet beállítani automatikusra, vagy fix értékűre a fényerősséget, a felhasználó választhat tartalomlistát (én csak egy listát használtam, minden tartalom ebben található), ez alatt pedig a VJT-vel való kapcsolatra vonatkozó információk jelennek meg, itt ad a program tájékoztatást a kapcsolódás során esetlegesen felmerült problémáról is. Középen látható a VJT virtuális mása. A projekt fájlban tárolt információk alapján tudja a program, hogy mekkora méretű táblát kell, megjelenítsen. A felső képen az aktuálisan kijelzett tartalom látható, míg az alsón a tervezett. Tartalom kiküldésekor ebbe az alsó részbe kell egérrel behúzni az aktiválni kívánt tartalmat, a listából, ha az már fel van töltve a VJT-re, vagy a programon kívülről fájlként, ha új tartalmat szeretne felölteni a felhasználó. Ha beillesztette a megfelelő tartalmat, akkor az activate nevezetű ikonnal tudja végrehajtatni az utasítást. A tartalom kiválasztására és aktiválására nem áll rendelkezésre gyors gomb, a művelet, csak az ikon vagy menüsorból végezhető.

A tartalom lehet szöveges, grafikus vagy vegyes, ezen kívül be lehet állíthatni fix képet, de akár szekvencia is létrehozható.



14. ábra: Szekvencia készítés SSC-ben

A szekvencia készítés során ez az ablak ugrik fel. Bal szélen itt a feltöltött tartalmak láthatóak, felül szintén megjelenik egy aktuális menüsor. Az alsó világos sávba kell behúzni a szekvencia által tartalmazott képeket a megjelenítésük sorrendjének megfelelően. A behúzott képre jobb gombbal kattintva előugrik egy menü, ahol beállítható, hogy mennyi ideig szeretné a felhasználó aktívan tartani az adott tartalmat.

A beállított képek felett pedig a tábla virtuális mását lehet láthatni, ahol látható, hogyan fog kinézni a szekvencia a táblán, ha a play gomb segítségével lejátszásra kerül.

A minimális idő, amíg egy képnek kinn kell lennie: 10ms, ennél gyorsabban ugyanis a VJT beépített számítógépe nem képes frissíteni a LED-ek állapotát. Maximális idő is meg van ugyan adva, de ez csak egy még ésszerűnek ítélt, egyébként teljesen indokolatlan érték. Részben, mert ugyan azon kép egymás után többszöri beillesztésével ez a korlát könnyedén megkerülhető, másrészt mert maga a kommunikációs protokoll is ennek sokszorosát képes továbbítani. Az elkészült szekvencia beállítása és aktiválása ezután már ugyanúgy történik, mint egy statikus tartalomé, a listában megjelenik a sorozat, a kezdő képpel és a megfelelő azonosítóval, ezt beillesztve és aktiválva a VJT-n megjelenik a kép és automatikusan elindul a lejátszás, amikor a végére ért, kezdi elölről.

A tartalmat lehet választani a tábla memóriakártyáján tároltakból, de feltölthető új tartalom is. Ekkor a felhasználónak kell az általa előállított képet beilleszteni a szoftverbe, ami – ha erre a kép előállításánál nem figyelt – átkonvertálja a megfelelő felbontásra és színmélységre, majd az aktiválást követően feltölti azt a VJT-re, majd pedig utasítást ad a megjelenítésére. Ezen művelet lassúsága és körülményessége tette szükségessé egy saját vezérlő program megírását. Mivel nem lehet minden lehetőségre előre felkészülni a megfelelő információval, ezért szükség volt egy olyan programra, ami a környezet változásához dinamikusan és főleg gyorsan alkalmazkodva képes előállítani a megfelelő információt és automatikusan feltölteni azt a VJT-re és meg is jeleníteni. Ennek ellenére természetesen a VJT-ken megtalálható lesz a legáltalánosabb képek listája, amiket ez által minimális számítási kapacitás igénybevétele mellett lehet majd megjeleníteni, de a forgalom folyamatos követése és szabályozása sokkal nagyobb számban követel majd speciális, egyéni jelzésképeket. Az ezek előállításához szükséges forgalmi paramétereket a táblák megkaphatják az őket vezérlő központból, vagy akár közvetlenül egy a közelben telepített forgalomirányító berendezéstől, a hozzá tartozó detektorok adatai alapján. Az adatokból a megfelelő információ vagy forgalomtechnikai utasítás előállítása történhet magán a VJT-nvagy a forgalomirányító berendezésen. Emiatt viszont ügyelni kell, hogy a használt forgalomirányítási modell elfogadhatóan hatékony, mégis elegendően egyszerű legyen ahhoz, hogy ne haladja meg az eszközök beépített számítógépeinek kapacitását.

7.2 Kommunikáció [11.]

A táblákkal UDP-n keresztül a FuturitCom 2.0 protokoll szerint lehet kommunikálni, amennyiben szeretnénk elérni a táblák szolgáltatásait. A protokoll leírását a Swarco a rendelkezésünkre bocsátotta.

A FuturitCom 2.0 egy Master-Slave típusú protokoll, ami azt jelenti, hogy van egy Master, ez a táblákat felügyelő és vezérlő központi számítógép és van(nak) egy vagy több Slave eszközök, ezek a táblák. A parancsokat a master küldi ki tetszőlegesen egy konkrét slave-nek, egy csoportnak, vagy pedig az összes slave eszköznek. A megszólított slave eszköz minden esetben reagál az üzenetre, kivéve, ha annak struktúrája eltér az előírttól (átvitel közben megsérült üzenet, vagy külső illetéktelen személy üzenete). Ha az üzenet felépítése megfelelő, de olyan adatokat tartalmaz, amiket nem tud értelmezni (nem létező tartalomazonosító), akkor úgynevezett NACK üzenet segítségével jelzi a hibát. Ha minden rendben van, akkor csak egy nyugtázó választ küld.

Az üzeneteket két csoportba lehet sorolni: a master által küldött üzenetek és a slave által küldött válasz üzenetek.Felépítésükegymástól eltér, de logikailag hasonló, a csoportokon belül pedig funkciótól függetlenül azonos. Amaster által küldött üzenetek több részre bonthatóak, van köztük, ami az OSI modell hetedik.,és van, ami a második szintjére vonatkozik. Az OSI modellt nyílt rendszerek összekapcsolására fejlesztették ki, lehetővé teszi a számítógépek egymással történő kommunikálását. Az első réteg a fizikai réteg, ez az egész alapja, innen indulva épülnek egymásra. Minden réteg az alatta lévők által nyújtott lehetőségeket, funkciókat használja fel és az általa kínáltakat a felette lévőknek szolgáltatja. A második réteg az adatkapcsolati réteg. Ez biztosítja, hogy adatokat tudjunk átvinni két hálózat között, valamint figyeli a fizikai rétegben felmerülő hibákat és lehetőség szerint javítja azokat. A 7. az alkalmazási réteg, ami a szoftverek közötti kommunikációt szolgálja. Az üzenetben vannak szinkronizáló byte-ok, vannak ellenőrző összegek, van, ami a parancs típusát jelzi és van természetesen a lényegi adat tartalom. A válasz üzenet felépítése szinte ugyan ez, de beékelődik plusz 3 tag, amik szintén az 7. OSI szinten fognak érvényesülni, ezek pedig a következők:

Status: a kapott parancsra vonatkozó állapotjelentés, pl: kép aktív.

Error1: hardveres hiba jelentését tartalmazza, pl: LED hiba.

Error2: szoftveres hiba jelentése, pl: érvénytelen tartalomazonosító.

A slave eszköz a válasz üzenetet mindig ugyan arra az IP címre és – alap esetben – ugyan azon a portonkeresztül küldi vissza, mint amiről a mastertől érkező parancsot kapták.

7.3 Parancsok [11.]

A parancsokat kétféle csoportba lehet sorolni, vannak az egyszerű parancsok, ezeknél egyetlen üzenetben elküldhető az utasítás, ami megadja, hogy pontosan mit kell csinálni és az adat, ami tartalmazza, hogy az utasítás által meghivatkozott paramétert milyen értékre kell beállítani. Ebbe a csoportba tartozik az utasítások túlnyomó többsége, mint például az egyszerű állapot lekérdezések, a cím beállítások, megjelenítendő (már feltöltött) tartalom azonosítója vagy akár a fényerősség stb. A másik csoport az extended-nek nevezett parancsok. Ezek olyan utasítások, melyek adattartalmának mérete meghaladja a protokoll által egy üzenetben küldhető maximális értéket, így azt több üzenetben kell továbbítani. Ilyenek pl a tartalom feltöltését vagy szekvencia elkészítését végző utasítások. Emiatt az üzenetek struktúrája is változik egy kicsit. Bekerül egy újabb rész, amit azt mutatja meg, hogy hány üzenet fog még érkezni, mire a teljes adattartalom átvitelre kerül. A feltöltendő adatot 127Byte-os méretű csomagokra bontva kell továbbítani, ennél többet ugyanis a protokoll nem engedélyez.

8. Vissim

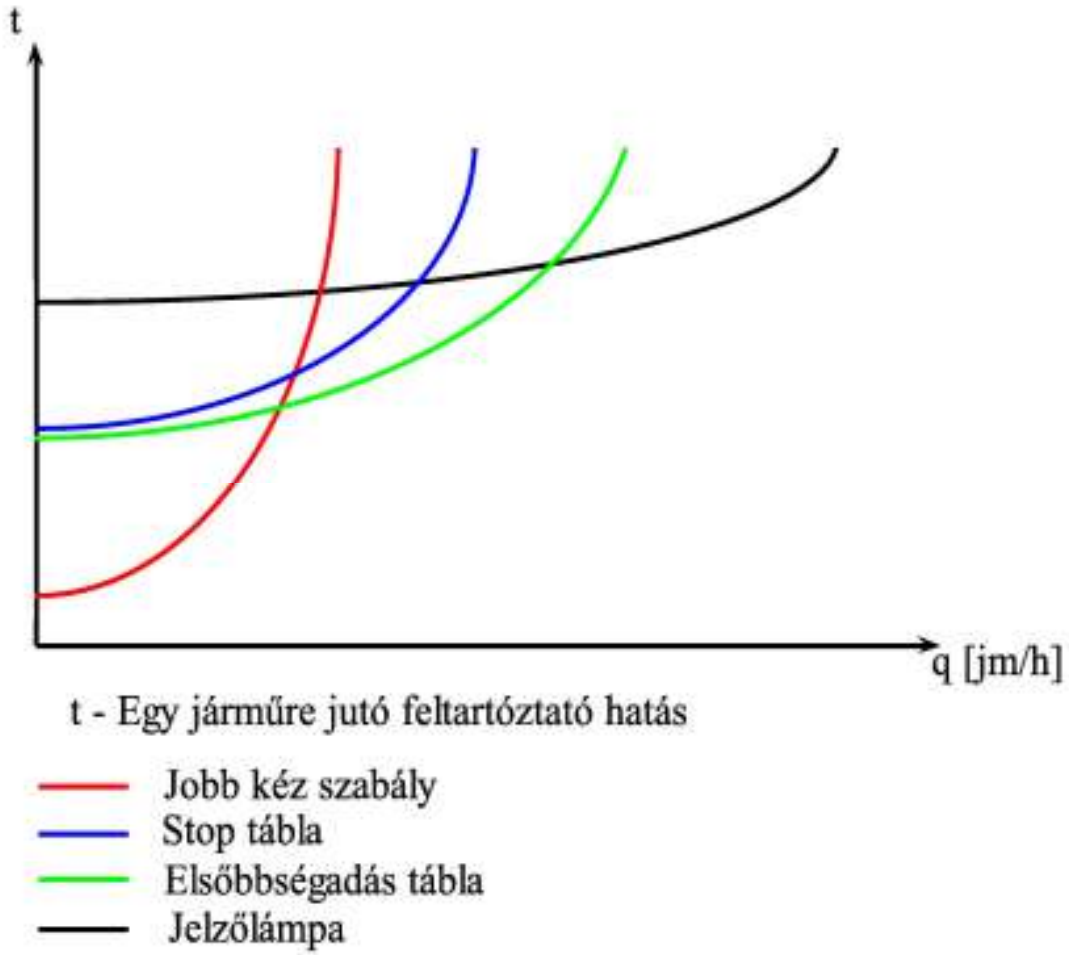
A Vissim a PTV AG által fejlesztett szoftver, amely makroszinten alkalmas multimodális forgalom modellezésére.

Képes kezelni gyalogosokat, bicikliseket és bármilyen közúti, vízi vagy légi közlekedésben megtalálható járművet. Ennek köszönhetően mára, ezen a téren a legelfogadottabb szimulációs programmá nőtte ki magát, így minden közlekedéssel foglalkozó intézmény elfogadja, az általa produkált eredményeket. Egyaránt alkalmas már meglévő rendszerek elemzésére, felülvizsgálatára, valamint új forgalomtechnikai módszerek kipróbálására, legyen szó akár autópálya, illetve stratégiai főútvonalak kezeléséről, belvárosi közlekedés irányításáról, közösségi közlekedés szervezéséről, vagy akár a forgalom környezeti hatásairól. Ezek a képességei mind időben, mind költségekben jelentős megtakarításokat eredményeznek, a forgalom fejlesztésére, korszerűbb szabályozására irányuló projektek megvalósítása során.

8.1 Szimuláció készítése [7.], [8.]

A Vissim által nyújtott, szinte végeláthatatlan lehetőségek miatt egy új felhasználó számára a szoftver használatának megkezdése meglehetősen bonyolult lehet. Még a legegyszerűbb szimulációk megvalósításához is rengeteg elem létrehozását és beállítását kell elvégezni.

A szimulációk készítésére egy grafikus kezelőfelületet szolgáltat a program. Egy új munka elkezdésekor először a kívánt közlekedési pályát kell megrajzolni. Ehhez lehetőség van térkép betöltésére (pl. GoogleMaps), hogy ennek mentén haladva létrehozható legyen egy már létező, vagy legalábbis megtervezett hálózat pontos másolata, de tervezhet teljesen saját elképzeléseken alapuló hálózatot is a felhasználó. Ezt követően meg kell határozni, hogy az egyes csomópontokban melyik sávból melyik sávba lehessen tovább haladni, majd a megfelelő be- és kilépési pontok megadásával be kell állítani az egyes útvonalakra terhelendő forgalmat. Mind a forgalom nagysága, mind összetétele módosítható és lehetnek akár fix, akár időszakosan változó értékek. A csomópontokban találkozó útvonalak konfliktus pontokat, területeket eredményeznek, ahol fennáll az egy időben megérkező járművek összeütközésének veszélye. Ennek kezelésére a program lehetővé teszi e területek beállítását, hogy melyik legyen a védett- és melyik az alárendelt irány, ezzel biztosítva az elsőbbségi szabályokat és elkerülve a baleseteket. Kisebb csomópontok esetén ennyi is elegendő lehet, nagyobb forgalmú csomópontok esetén azonban részben a biztonság, részben pedig a minél jobb kapacitás kihasználás érdekében szükséges jelzőlámpák használata.



15. ábra: Csomópontok irányítási módjai

A fenti ábrát forgalomtechnikai összefüggésekkel és tapasztalati mérésekkel határozták meg. Ez megmutatja, hogy az adott kereszteződésben milyen irányítási mód a legmegfelelőbb.A jelzőlámpák elhelyezésekor meg kell adni, hogy melyik vezérlő alá tartozzanak (a valóságban ez a vezérlő felel meg a forgalomirányító berendezésnek), az csomópontokat célszerű külön-külön kezelni és az egy csomóponthoz tartozó lámpákat ugyan azon vezérlő alá besorolni.Ezt követően meg kell adni az egyes vezérlők fázis tervét. Itt több lehetőség áll a tervező előtt, mint pl:

- Fix fázisterv
- VAP
- External

Minden esetben ki kell alakítani a fázisokat, vagyis, hogy melyek azok az irányok, amelyek egyszerre kaphatnak szabadjelzést. Ennek meghatározására szintén vannak megfelelő forgalomtechnikai számítások, ökölszabályként megemlíthető, hogy az egy fázisba tartozó irányok ne, vagy csak kis mértékben (pl.tele zöld) akadályozzák, veszélyeztessék egymást. Meg kell határozni a közbensőidő mátrixot. Ez tartalmazza, hogy az egyes fázisok minimálisan mennyi idővel követhetik egymást, hogy biztosítva legyen a véget érő fázisból utolsóként belépő jármű biztonságos kihaladása, mielőtt a soron következő fázis első járműve behaladna a csomópontba. A ciklusidő megadása

után be kell állítani a korábban már megtervezett fázisidő tervet, ennél a lépésnél a program figyelembe veszi a korábban megadott közbensőidő mátrixot és nem enged olyan fázisidőket beállítani, amik ezt megsértik.

Fix fázisterv választása esetén a program által biztosított grafikus felületen lehet megcsinálni ezeket a beállításokat, ez a legegyszerűbb mód, a pontos forgalomirányításra és szabályozásra azonban nem alkalmas. A VAP opcióval, egy a Vissimhez tartozó VisVap programban elkészített logikát lehet betölteni és a Vissim ezen logika alapján vezérli a lámpákat.

Az External megnevezésű fázisterv kiválasztása csak akkor engedélyezett, ha a Vissimhez megvan a megfelelő licensz és a SignalController API, amivel fejlesztői környezetben lehet programozni a lámpákat. Ez a módszer igényli a legnagyobb hozzáértés,a Vissim mellett, valamilyen programnyelvhez (C++, Visual Basic, Java) is érteni kell, viszont ez biztosítja a legnagyobb szabadságot. Itt válik ténylegesen lehetségessé külső forrásból származó adatok felhasználása, valamint összetett irányításelméleti módszerek alkalmazása a forgalom szabályozására. Ezek mellett a Vissim még számtalan lehetőséget kínál, hogy a szimuláció minél realisztikusabb legyen.

Az eredmények könnyebb prezentálása érdekében pedig a grafikus felület tartalmaz egy olyan opciót is, melynek segítségével az elkészült szimulációt egy videó segítségével lehet bemutatni. A szimuláció során a felhasználó átválthat 3D-s nézetbe, amelyhez a vizuális alapokat az adott helyszínről készült felvételek, valamint a Vissim adatbázisban található közlekedőkről és berendezésekről készült 3D-s modelljei szolgáltatják, az így készült mozgókamerás felvétellel, pedig könnyedén bemutatható a megvalósított hálózat és a használt modell működése.

8.2 COM [9.]

A COM segítségével a felhasználó egy programozói környezetben képes kezelni a szimulációt. Az így létrehozott futtatható állomány elindításával elindul maga a szimuláció is (emiatt fontos, hogy csak olyan gépen futtatható, ahol a Vissim megfelelő verziója is megtalálható és az elérési utak megegyeznek a programozás során beállítottakkal), a szimuláció futása közben lehetséges manipulálni a szimuláció paramétereit, illetve a forgalommal kapcsolatos információkat lekérdezni. Ehhez azonban a szimuláció létrehozásakor engedélyezni kell az adatokhoz való hozzáférést. A Vissim két eszközt is felkínál:

- Detector
- Data CollectionPoint

A detektorok a szimuláció során használhatóak, hogy információt küldjenek a jelzőlámpák vezérlőjének, ami ez által befolyásolhatja a fázistervet. A COM-on keresztüli adatgyűjtésre a Data CollectionPoint a megfelelő. Ezeket kell elhelyezni a vizsgálni kívánt útszakaszok megfelelő pontjain és meg kell adni, hogy milyen információkat akarunk lekérdezni. Ezek nélkül a COM-ból nem lehetséges elérni a forgalmi paramétereket.

9. Saját program [10.]

Az SSC említett hiányosságai miatt szükségesnek éreztem egy saját program megalkotását, ami a VJT-k dinamikus használatát lehetővé tevő módon használja ki a Futurit protokoll által nyújtott lehetőségeket. A program megírására a Java nyelvet választottam, mivel ez egy igen elterjedt, jól használható programnyelv. Ezen kívül a Tanszék laborjában található VTC 3000 ACTROS típusú forgalomirányító berendezés szintén Java nyelven programozható, így lehetőség nyílik a két berendezés összekapcsolására a munka későbbi szakaszaiban.

9.1 Lib

Egy az SSC-hez hasonló önálló, grafikus felületű program helyett egy univerzálisan használható megoldás mellett döntöttem. Ezáltal a program sokkal mobilisabb lesz, mivel bármilyen másik Java programmal könnyedén képes lesz majd együtt működni, ráadásul a cél, a forgalom változását minél gyorsabban lereagáló és automatikus VJT-vezérlés, emiatt nem okoz majd problémát a grafikus felület hiánya, hiszen az csupán az emberi felügyelet és beavatkozás számára lenne hasznos. Emberi vezérléssel azonban a dinamikus, gyorsan reagáló rendszer nem lenne kihasználható, ha mégis ilyen megoldásra lenne szükség, például valamilyen meghibásodás miatt, akkor erre az esetre ott lesz az SSC. A Java objektum orientált felépítése lehetővé teszi, hogy a vezérlő, egyszerűen beépíthető legyen más programokba. Az univerzitás mellett az adatcsere folyamatát is sokkal egyszerűbbé teszi ez a megoldás, mivel nem két külön program között kell biztosítani az adatok átadását, hanem az integrálás után ez egy programon belül fog megtörténni, tehát előre lefoglalt memória területek vagy külön adattároló fájlok helyett programon belüli változókkal megoldható a probléma. Az eredeti szoftver a vizsgált, mért és kiszámolt adatokat saját magán belül adhatja át az elkészült programnak, ami azután levezényli a kommunikációt a VJT-kkel.

A programot ennek értelmében Libként készítettem el, ami könnyedén beépíthető bármilyen más Java programba.

Első lépésben létre kellett hozni egy külön osztályt, ami tartalmazza a VJT-k adatait:

- IP cím
- Kommunikációs port
- Felbontás

Az elkészült Lib-et egyszerűen csak hozzá kell adni, egy már meglévő Java projekthez és már használható is. A Lib ugyan rendelkezik saját main függvénnyel, viszont ez csak a tesztelést szolgálja, más projektbe integrálva ez a része már nem fog lefutni, csak a főprogram main függvényében meghívott függvényei kerülnek végrehajtásra. A main az a része egy programnak, ami valójában elindul, minden más függvényt, amit a program tartalmaz, a main-en belül hívunk meg, ha szükség van rájuk, ott és akkor, amikor szükséges. De ha a main függvényből nem hívjuk meg őket, akkor nem fognak lefutni.

Az

```
x = newVMS("192.168.0.44",12346);
```

paranccsal lehet új VJT-t felvenni. X az új tábla neve, VMS az osztály, amit létrehoztam, zárójelben pedig paraméterként lehet megadni az új tábla IP címét, és a port számot, amin keresztül kommunikálni lehet vele. Ez a következő konstruktorok valamelyikét fogja meghívni:

```
VMS(Stringaddress, int port) throwsSocketException {
            this(newInetSocketAddress(address,port));
      }
VMS(InetAddressaddress, int port) throwsSocketException {
            this(newInetSocketAddress(address,port));
```

Az elsőt akkor, ha a VJT felvételekor adjuk meg a paraméter listában az IP címet és a portot (ahogy a mintakódban is látható). A másodikat akkor, ha a cím már korábban tárolásra került az InetAddressben és paraméterként már ezt írjuk be. Az első kontsruktorban látható, hogy míg a port szám típusként van definiálva, addig az IP cím szövegként. Erre azért van szükség, mert a hagyományos IP cím 4 részre tagolódik és ezeket '.'-tal választják el egymástól, ez viszont olyan karakter, ami nem fér bele a szám típusba, ezért kell string-ként definiálni.

A fentiek egy újabb konstruktort hívnak meg:

```
VMS(InetSocketAddresssocket) throwsSocketException {
            vms = socket;
            mySocket = newDatagramSocket(vms.getPort());
            mySocket.setSoTimeout(TIMEOUT);
```

ami az operációs rendszertől elkéri, és saját részre lefoglalja a megadott portot. A művelet során, több helyen is felmerülhet hiba. Ezekre a hibákra vonatkoznak a függvények `SocketException` részei. Amennyiben a megadott paraméter nem értelmezhető, vagy az operációs rendszer nem engedélyezni számunkra a használatát, akkor ezzel kapcsolatosan hibaüzenetet generál a felhasználó számára.

Miután a programba bevittünk egy VJT-t már lehet vele kommunikálni. Erre a következő függvény szolgál:

```
privatevoidsendMessage(bytecmd, byte[] data) throwsException
      {
            String.format("%x", Byte.valueOf(Sync1));
            if(data.length>127)
                  thrownewIOException("Hosszu a data!");
            byte[] buffer = newbyte[7+data.length+1];
            buffer[0] = Sync1;
            buffer[1] = Sync2;
            buffer[2] = Addr;
            buffer[3] = SubAddr;
            buffer[4] = (byte) (1+data.length);
```

```
            buffer[5] = 2;
            buffer[6] = checksum(buffer,0,6);
            buffer[7] = cmd;
            System.arraycopy(data, 0, buffer, 8,data.length);
            buffer[buffer.length-1]=checksum(buffer,7,7+data.length);
            DatagramPacketmsg =
newDatagramPacket(buffer,buffer.length,vms.getAddress(),vms.getPort())
;
            mySocket.send(msg);
      }
```

Ez egy visszatérési érték nélkül függvény, ami annyit jelent, hogy ha meghívjuk, akkor végrehajtja a benne leírt feladatot, amit a külvilág érzékel, de a programon belül nem fog eredményt produkálni, vagyis nem adhatjuk meg értékként egy változónak. Meghíváskor paraméterként meg kell adni, hogy az üzenetben milyen parancsot és milyen adathalmazt kívánunk elküldeni. A cmd, vagyis maga az utasítás egy bájtnyi érték, az elküldendő adat viszont már egy byte típusú tömb, mivel egy üzeneten belül akár 127 bájtnyi adat is elküldhető. Ez azonban nem írható be közvetlenül a függvény paraméterében, sokkal könnyebben kezelhető, ha előre letároljuk egy tömbben és paraméterként már csak ezt a tömböt vesszük elő. Kezdetben még csak egyszerű üzenet küldése a cél, ami maximálisan 127 bájtnyi adatot tartalmazhat, ezért először meg kell vizsgálni, hogy a küldeni kívánt információ megfelel-e ennek a kritériumnak. Folyamatban van a megoldása, hogy nagyobb méretű üzenet esetén a program automatikusan megfelelő méretű részekre darabolja az adatot és az ExtendedTelegrams-nak megfelelő üzenetstruktúrával továbbítsa azt. Az üzenet összeállítása automatikusan történik. A program először létrehoz egy buffer nevezetű tömböt, amiben már nem csak maga az adat fog szerepelni, hanem a protokollnak megfelelő struktúrában az egész üzenet. Az üzenet első hét eleméről biztosan tudjuk, hogy 1 byte hosszúságú, az adat azonban 1 és 127 byte között bármekkora méretű lehet. A kiterjesztett üzeneteket leszámítva az adathalmaz csak nagyon ritkán töltené ki a rendelkezésére álló 127 bájtnyi területet, ezért fölösleges lenne minden üzenetnél fenntartani neki. Emiatt az üzenet tárolására szánt tömb méretének megadásakor nem fix értéket adunk meg, hanem lekérdezzük az adatot tartalmazó data tömb hosszát és ennek megfelelően az adott üzenethez igazodva dinamikusan kerül megadásra a szükséges tömb mérete. Ezután következik az üzenet összeállítása. Az első két byte egy szinkronizációs érték, ami minden üzenet esetén megegyezik. Címeket, a tanszék két táblája esetén egyelőre fixen adok meg, további vizsgálatok után azonban ez is dinamikusan változtatható érték lesz. A negyedik byte az adat rész hosszára utal. Az ötödik byte egy úgynevezett Tag, ami lényegében üzenetszámlálóként működik. Minden újabb üzenetnél növekszik az értéke, a tábláktól visszaérkező üzeneteket pedig e Tag alapján lehet azonosítani, hogy melyik kiküldött üzenet válaszául érkeznek. Ez segít abban, hogy egy hibás üzenet esetén azonosítható legyen, hogy melyik üzenetet kell újra elküldeni, szükség esetén pedig előtte akár javítani is. A hatodik egy ellenőrző összeg, ami lehetőséget ad az üzenet esetleges sérülésének és a hiba helyének azonosítására. Ennek meghatározását a program végzi, a protokollban leírt metódusnak megfelelően. Ehhez egy újabb

függvény kerül meghívásra, az alábbi paranccsal:

```
buffer[6] = checksum(buffer,0,6);

privatestaticbytechecksum(byte[] data,intfrom, intto)
      {
            int i=0;
            for (int j = from; j<=to;j++)
                  i-=data[j];
            return (newInteger(i)).byteValue();
      }
```

Ez a függvény már rendelkezik visszatérési értékkel, ez pedig a kiszámolt ellenőrző összeg lesz, amit a fenti paranccsal a buffer tömb hatodik elemének fogok értékként megadni. A függvény meghívásakor paraméterként meg kell adni, hogy mely adatokból szeretném az ellenőrző összeget kiszámolni. Az értékek konkrét megadása helyett inkább azt adom át, hogy hol találhatóak ezek az értékek (jelen esetben a buffer nevű tömbben), illetve, hogy a forrás hányadik elemétől, hányadik eleméig kerüljön figyelembe vételre. Az első ellenőrző összeget az őt megelőző, tehát első 6bájt értékéből kell kiszámolni. Java-ban a tömbök elemeinek sorszámozása 0-tól indul, ezért kell paraméterként a 0-6 intervallumot megadni.

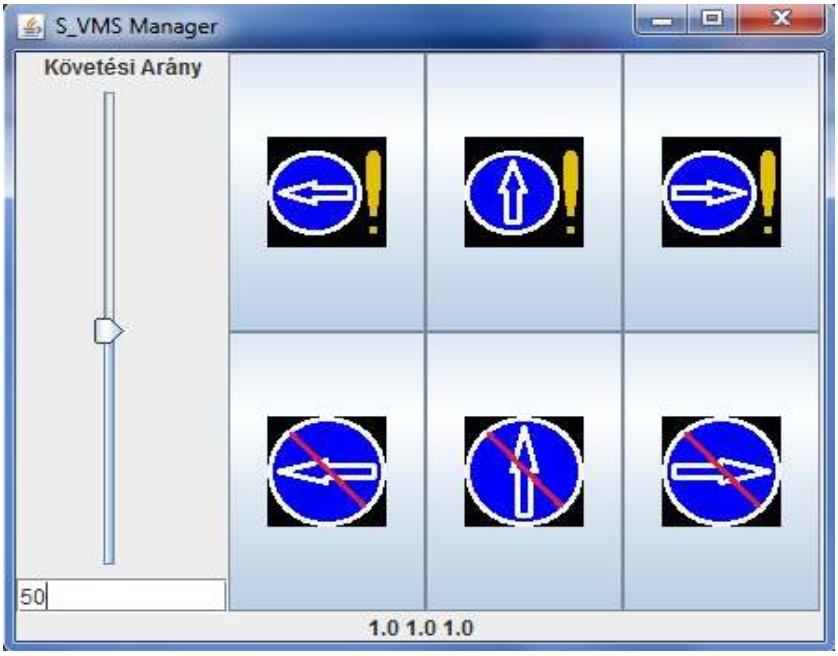
A checksum függvény a paraméterekként kapott adatoknak létrehoz egy saját tömböt, ez a data tömb, lokális változó, tehát csak ezen a függvényen belül érvényes, semmi köze a sendMessage függvényen belül használt data tömbhöz. Ebben fogja tárolni a buffer tömbből átvett értékeket. A from és a to változókban pedig az fog szerepelni, hogy hányadik elemtől hányadik elemig kell elvégezni az ellenőrző összeg számítását. Az összeg kiszámítása a túlcsordulást kihasználó összeadáson alapszik. Ennek köszönhetően egy egyszerű for ciklussal megvalósítható a feladat. A for ciklus a programozási nyelvekben gyakran alkalmazott eszköz. Lényege, hogy egy változót valamilyen kezdeti értékről indítva folyamatosan változtatjuk (növeljük, vagy csökkentjük) a for paramétereként, relációval (kisebb /</, kisebb-egyenlő /<=/, egyenlő /=/, nagyobb-egyenlő />=/, nagyobb />/) megadott határéig. A for egy elöl tesztelős ciklus, ami azt jelenti, hogy először megvizsgálja, hogy a változó aktuális értéke teljesíti-e a feltételt. Ha még nem, akkor végrehajtja a rajta belül megadott utasításokat, ha teljesíti, akkor pedig tovább lép és nem fut le többször. A ciklusváltozó értékét többféleképpen is változtathatjuk. Az i++, a ciklus lefutása után egyel növeli, az i— pedig egyel csökkenti az értékét, míg a ++i és --i, a ciklus lefutása előtt növeli, illetve csökkenti az értékét. A for ciklusnak nagy hátránya azonban, hogy a feltétel vagy a ciklusváltozó változtatásának rossz megválasztása esetén könnyen végtelen ciklusba kerülhetünk, amiből a program, csak emberi beavatkozás segítségével tud kilépni. Az általam használt for ciklus mindössze egyetlen összeadási műveletet tartalmaz, majd a ciklus végeztével az eredményt visszaadja a sandMessage függvénynek. Az ezután tovább lép az üzenet következő részére, ami a cmd, amit paraméterként kapott, a meghívásakor. Ez fogja megmondani a tábláknak, hogy a kapott adatokkal milyen műveletet hajtsanak végre (aktiválják, töröljék, mentsék stb). Ezt követi egy System.arraycopy nevezetű parancs, aminek az a feladata, hogy egy adott tömb elemeit átmásolja egy másik tömbbe. Jelen esetben a data tömbből, a 0. elemmel kezdődően másolja át a buffer tömbbe, annak 8. elemétől kezdve –vagyis az előzőekben megadott 7 byte-nyi üzenetrész után— egészen a data tömb utolsó eleméig (data.lenght). Ezután már csak az üzenet utolsó része marad, ami egy újabb ellenőrző összeg, ez a buffertömb

utolsó elemének helyére (buffer.lenght-1) kerül. Ennek az elemnek a helye azért így kerül megadásra, mert nem tudom, hogy az adott üzenetnél milyen hosszú a data rész, ezért a tömb hátuljától indulva visszafelé haladva azonosítom, hogy az utolsó. Ehhez pedig azért kell a ’-1’, mert a tömb elemeinek számozása 0-tól indul, vagyis egy 10 elemű tömb első eleme a 0-ás, utolsó eleme pedig a 9-es sorszámot kapja. Ezzel az üzenet elkészült. A program ezután létrehoz egy msg nevű, DatagramPacket, vagyis adatcsomag típusú változót, amiben elhelyezi az üzenetet, annak hosszát, a kiválasztott tábla címét, a kommunikációra használt portot és ezután kiküldi a csomagot.

További feladat az egy üzenet által megengedett méretet meghaladó adatok feldarabolása és több üzenetben történő továbbítása, valamint a tábláktól érkező válaszüzenetek kezelése. Tag automatikus megadása.

9.2 Grafikus kezelőfelület (GUI), szerver

A valóságban a közlekedés résztvevői látják a VJT-ken kijelzett üzeneteket és értelmezik azokat. A Vissim azonban egy program, így alternatív megoldást kellett találni, hogy a táblák jelzésképét visszacsatolva megvizsgálható legyen annak forgalomra gyakorolt hatása. A megoldás, hogy a táblákat vezérlő program nem csak a táblákon megjelenítendő információt küldi el a megfelelő táblának, hanem egy annak megfelelő adattartalmú üzenetet küld a Vissimhez kapcsolódó programnak, amely ennek megfelelően befolyásolja a szimuláció futását. Az eredeti elképzelés a COM felületen C++ nyelven megírt program Java nyelvre fordítása és a táblákat vezérlő programmal történő egybeépítése volt, de ez indokolatlanul sok időt és energiát emésztett volna fel, így hatékonyabb megoldásnak bizonyult, ha megmaradnak önálló programoknak és inkább megvalósítom a kettő közötti kommunikációt. A táblavezérlő ugyan már képes üzenetet küldeni a VJT-knek, azonban még nincs leprogramozva semmilyen forgalomszabályozó algoritmus, ami önállóan lenne képes kiválasztani a megjelenítendő információt, így megírásra került egy grafikus kezelő felületű program, amelyen keresztül a felhasználó tetszőlegesen választhat valamilyen útvonalajánlást.



16. ábra GUI (Grafikus teszt program)

Az ablak bal oldalán látható egy csúszka, ami a felette elhelyezett szövegnek megfelelően annak beállítására szolgál, hogy a járművek hány százaléka vegye figyelembe a megjelenített ajánlást. Az alatta lévő rubrikában a csúszka mozgatásakor a program automatikusan kijelzi az éppen aktuális értéket, illetve lehetőség van a csúszka nélkül közvetlen ide beírni a megfelelő értéket megkönnyítve ezzel a kívánt követési arány beállítását.

A csúszkától jobbra található 6 db útvonalajánlás:

- balra ajánlott
- egyenesen ajánlott
- jobbra ajánlott
- balra nem ajánlott
- egyenesen nem ajánlott
- jobbra nem ajánlott

Az ablak alján pedig egy status bar (állapotjelző) található, ami kijelzi a program által a választott ajánlásnak megfelelő fordulási ráta értékeket.

Ebbe a programba került beépítésre a socketes kommunikáció szerver oldala:

```
	privateclassServerThreadimplementsRunnable {
		ServerSocketserverSocket;
	publicServerThread(int port) {
	try{
				serverSocket = newServerSocket(port);
			}catch (IOException e) {
				e.printStackTrace();
			}
		}
	publicvoidrun()
	{
		while (serverSocket!=null)
		{
			try
			{

	SocketclientSocket = serverSocket.accept();
	BufferedReaderin =
newBufferedReader(newInputStreamReader(clientSocket.getInputStream()));
	vissim = newPrintWriter (clientSocket.getOutputStream());

	String s;
	while ((s = in.readLine()) != null)
			{
		if (!s.equals(""))
		{
			String[] splits = s.split(" ");
			EredetiBal = Double.valueOf(splits[0]);
			EredetiEgyenes = Double.valueOf(splits[1]);
			EredetiJobb = Double.valueOf(splits[2]);
		}
			}

		  }
			catch (Exception e)
			{
```

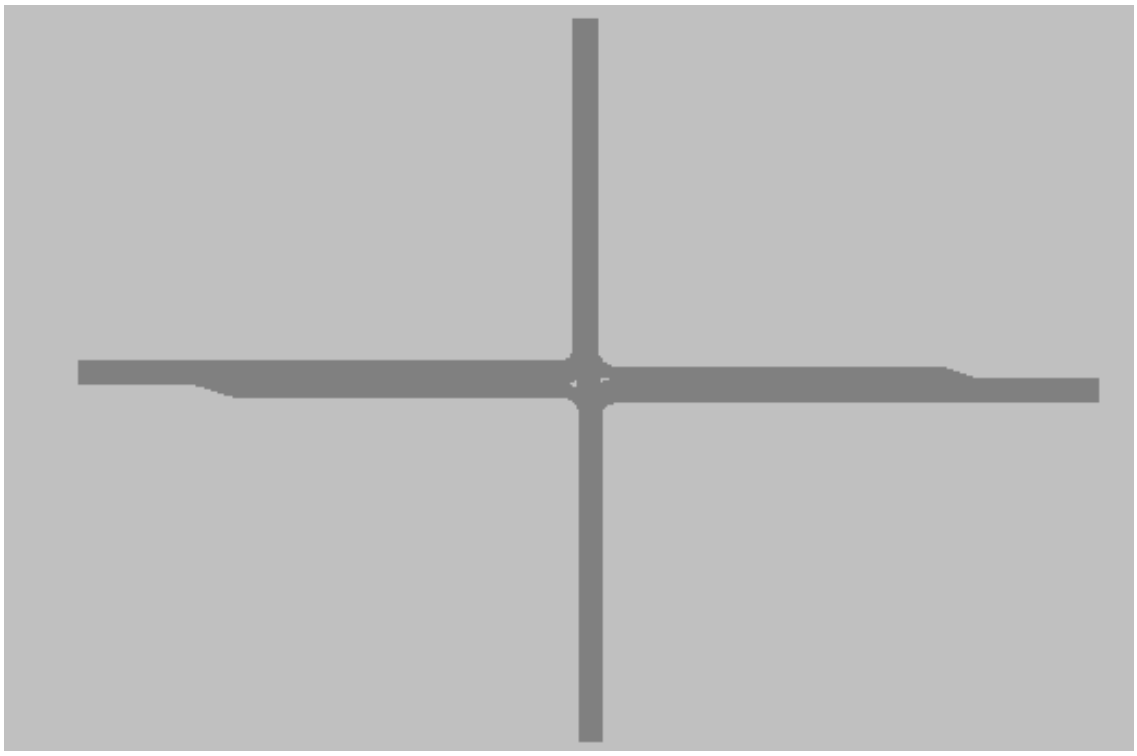
```
                        StatusBar.setToolTipText(e.getMessage());
                        e.printStackTrace();
                    }
                }

            }
        }
```

A szerver külön szálra került, tehát az alapprogram mellett folyamatosan fut. Megpróbálja lefoglalni és megnyitni az előre megadott (jelen esetben 7750-es) portot, majd ha sikerül, akkor várja, hogy valaki csatlakozzon hozzá. Amikor egy kliens, a – teszt során a Vissimhez tartozó program – bejelentkezik csatlakozásra, akkor elfogadja azt és átveszik a tőle érkező üzenetet, amely a szimuláció eredeti fordulási rátáit tartalmazza. Ezek után a felhasználó döntéseinek megfelelően eseményről eseményre elküldi a kliensnek a módosított értékeket.

Eredeti céljának megfelelően a Lib beépítésre került ebbe a programba, lehetővé téve így, hogy amikor a felhasználó választ egy ajánlást, akkor a program ne csak a kliensnek küldje el a megváltozott értékeket, de a Lib-en keresztül a VJT-knek is üzenetet küldjön, amelynek hatására a megfelelő táblán megjelenik az ajánláshoz tartozó ábra.

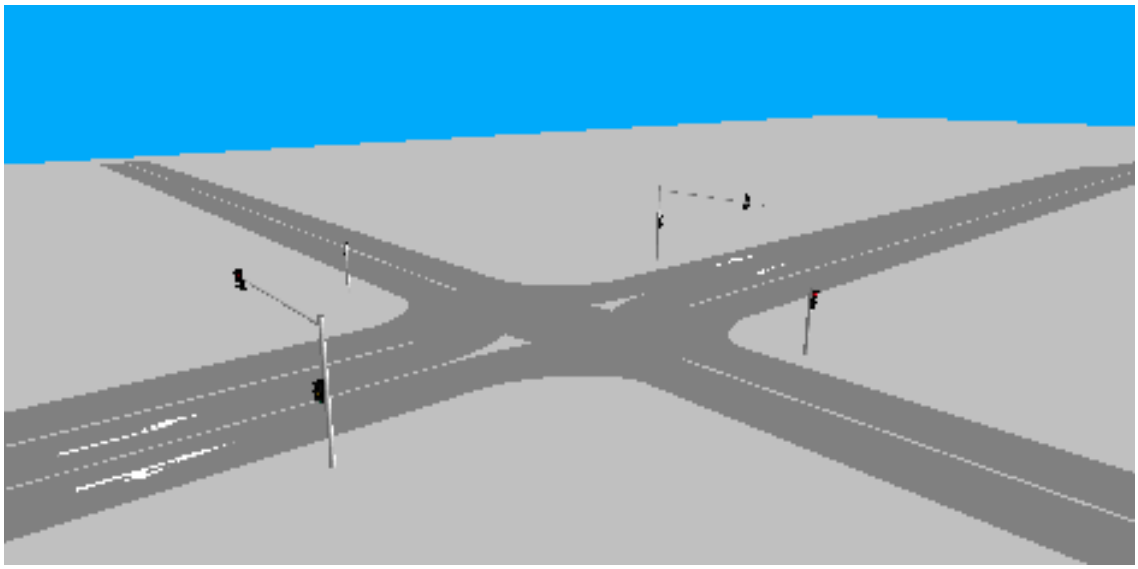
10. Vissim szimuláció [8.]

Próba szimulációra egy egyszerű kereszteződést választottam. A kereszteződés egy nagy forgalmi terhelésű és egy kisebb alárendelt út kereszteződését valósítja meg. A forgalmat jelzőlámpák irányítják. A lámpák fázisideje egyelőre fix, de további vizsgálatok során a SignalControl API segítségével forgalomfüggő fázisterv megvalósítása is szerepel a tervek között. A munkaelőrehaladtával ehhez a VTC 3000 berendezés szolgáltatná az adatokat, a Vissimben futó szimuláció paraméterei alapján. Ehhez szükség lesz majd egy kétirányú kommunikáció megvalósítására a forgalomirányító berendezés és a szimulációs programot futtató számítógép között. Ez még csak távlati tervek között szerepel, jelen dolgozat elkészítéséhez csak fix fázisidőket alkalmaztam.

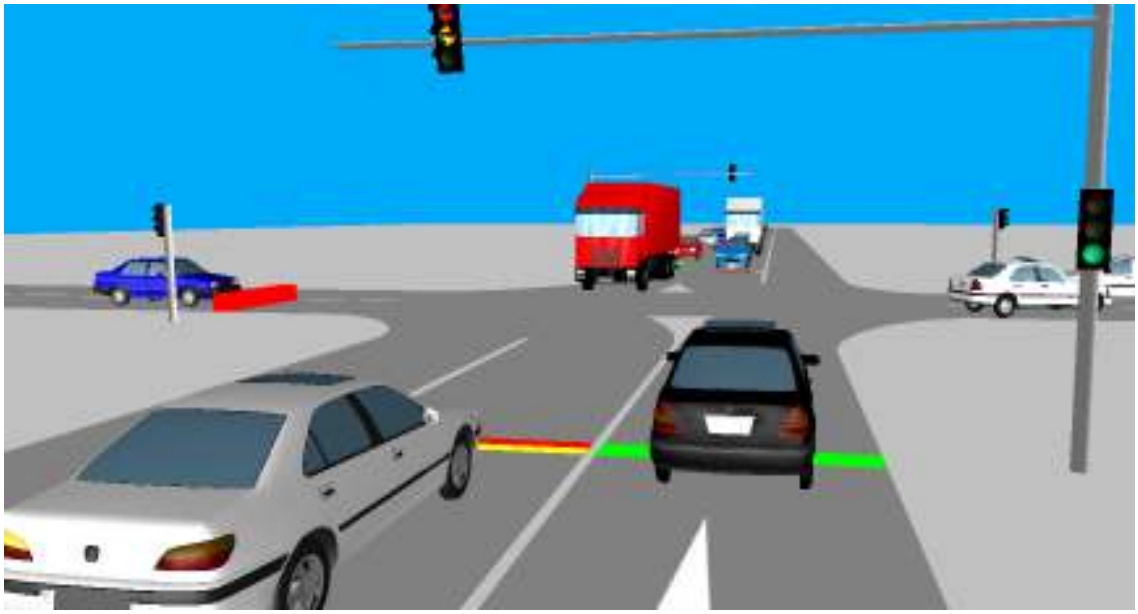


17. ábra: Szimulált csomópont

A nagyobb terheltségű úton létrehoztam egy kanyarodó sávot, hogy mérsékeljem a sorképződést és növeljem a kapacitást. A középső sávból balra lehet kanyarodni, míg a szélsőből egyenesen és jobbra lehet tovább haladni. A lámpák programjában 3 fázist alakítottam ki, egyet a balra kanyarodóknak, egyet az egyenesen illetve jobbra haladóknak, egyet pedig a keresztirányú alárendelt útnak. A szimuláció felépítése ugyan 2D-s nézetben lehetséges, azonban prezentációk céljára a Vissim 3D-s nézetet is támogat, melyből akár az egész hálózatot jól bemutató videó is készíthető.



18. ábra: Szimulált csomópont 3D-s nézetben

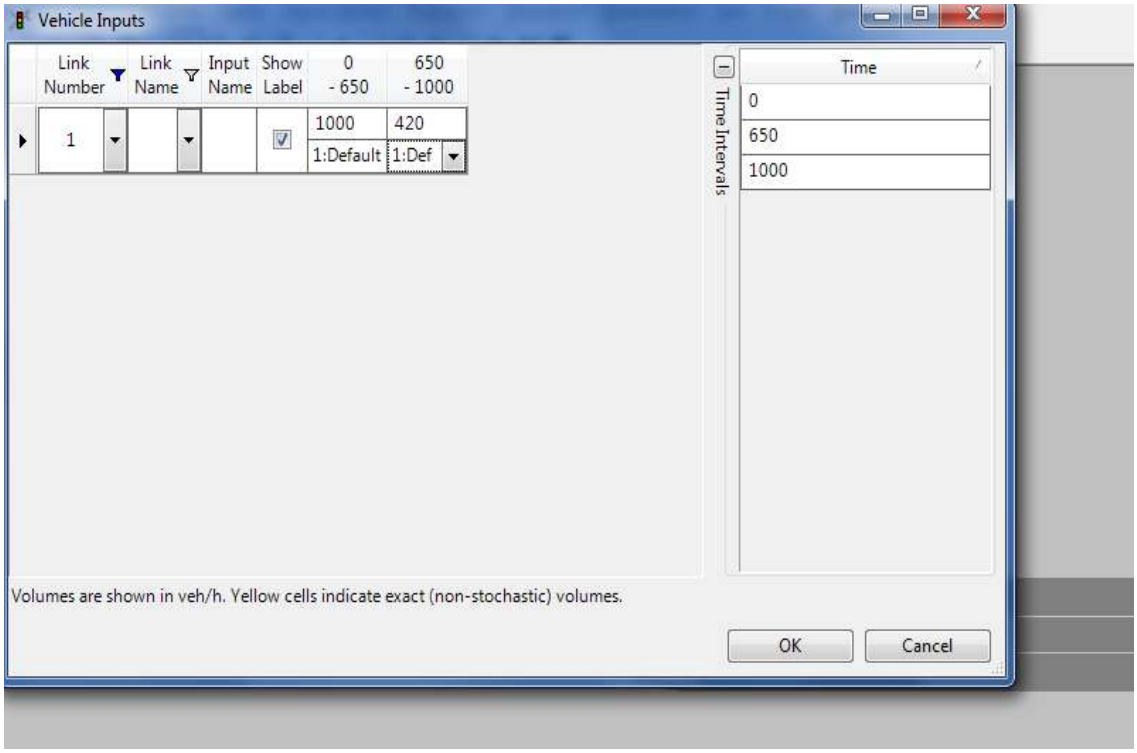


19. ábra: Szimulált forgalom 3D-s nézetben

A Vissim rendkívül részletes 3D-s modelleket tartalmazó adatbázissal rendelkezik, mind a járműveket, mind a környezetet illetően. Az adatbázis bővíthető is és a környezet felépítésénél teljesen szabad kezet kap a készítő. Még egy egyszerű jelzőlámpánál is számtalan lehetőségünk van, mind a lámpaoszlopra mind magára a lámpára és annak elhelyezésére vonatkozóan és ugyanígy gazdagíthatjuk a szimulációt táblák használatával is.

10.1 Szimulációs paraméterek beállítása

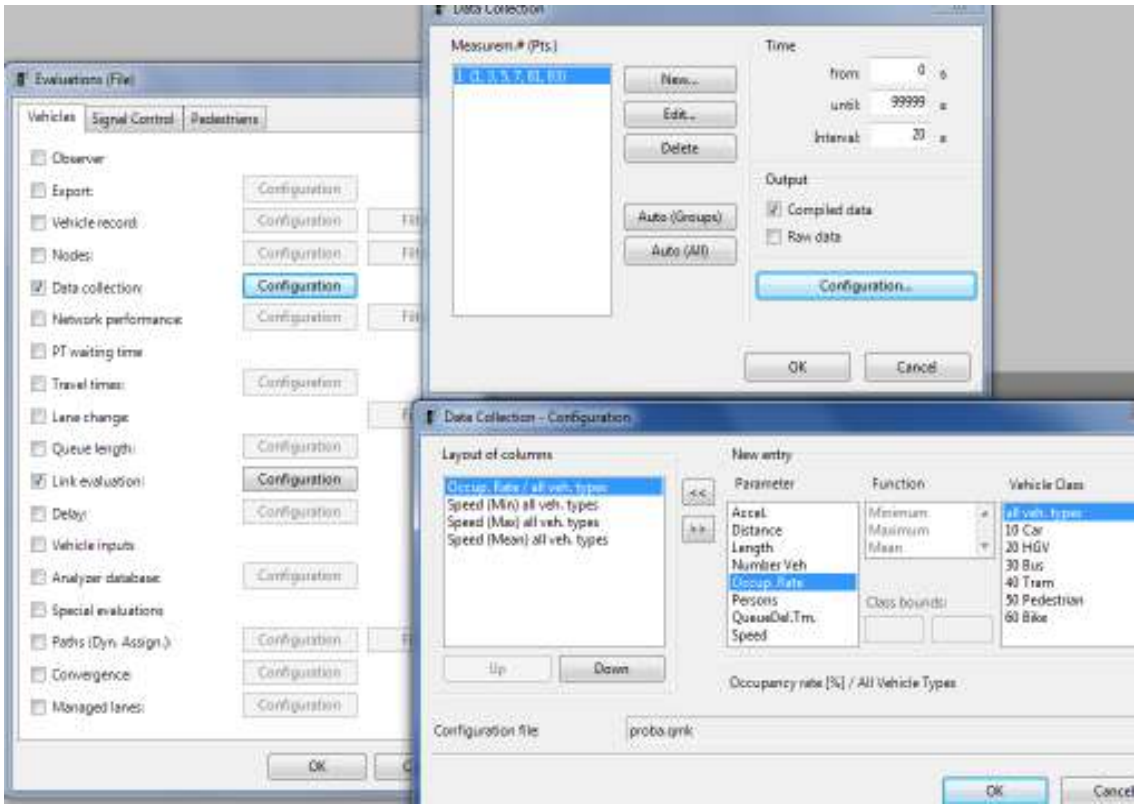
A vizuális beállítások mellett a forgalom is számos állítható paraméterrel rendelkezik. Meg kell adni az engedélyezett irányokat, az elsőbbségi viszonyokat, a forgalom nagyságát és lehetséges irányonkénti megoszlását.



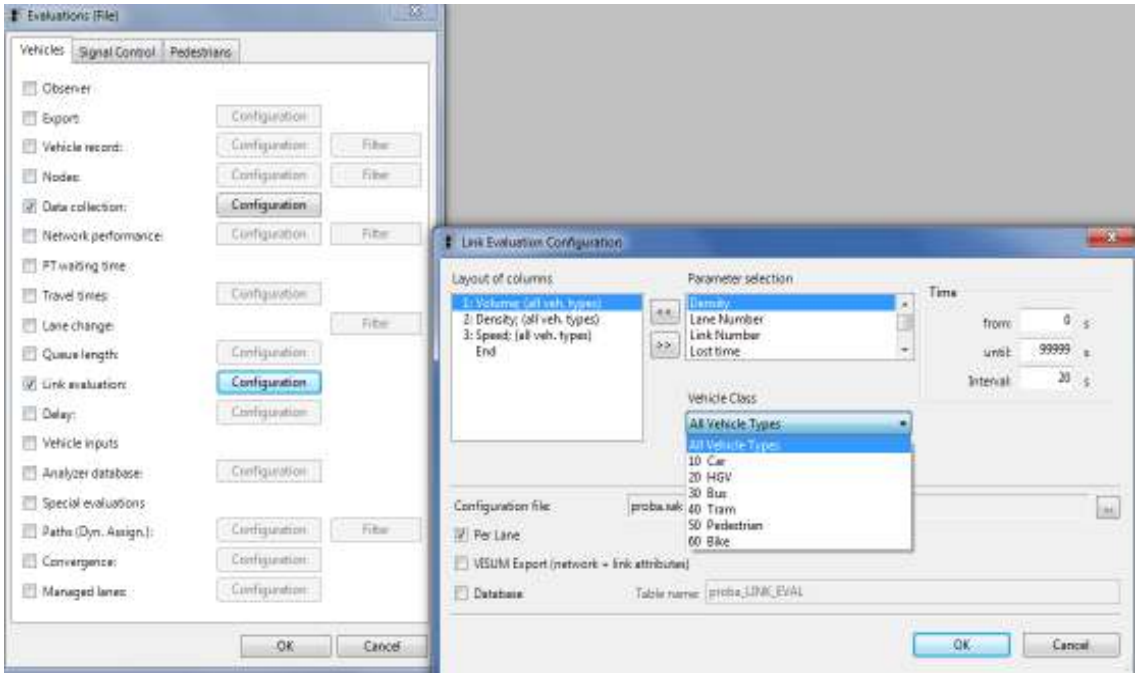
20. ábra: A hálózatba belépő járműszám beállítása

Forgalomnagyság megadásakor lehetőség van időszakokra lebontva más-más forgalmat generálni az adott útszakaszokra. Ekkor meg kell adni, hogy az egyes időszakok a szimuláció hányadik másodpercétől, hányadik másodpercéig tartsanak. Ezután megadtam a forgalom nagyságot. A forgalom összetételét is lehet módosíthatni. A Vissim tartalmaz egy alapbeállítást, de lehetőség van sajátot is létrehozni, ahol szabadon választható meg az egyes járműtípusok aránya. Ezáltal könnyéden figyelembe vehetőek a jellemző csúcsidőszakok illetve az is, hogy az egyes utakon a személyes, fuvarozói vagy éppen a tömegközlekedés a jellemzőbb és ennek megfelelő lehet forgalom-összetételt választani.

Ahhoz, hogy lekérhessem forgalmi adatokat először DataCollection pontokat kellett elhelyeznem a vizsgálni kívánt útvonalakon. Jelen szimulációban minden úton a kereszteződés előtt helyeztem el ezeket a pontokat. Az Evaluation->File menüpont alatt tudom engedélyezni a hozzáférést az adott utakhoz és adat gyűjtő pontokhoz.



21. ábra: Adatgyűjtők beállítása



22. ábra: Útszakaszokon történő mérések beállítása

A DataCollection engedélyezésével az adatgyűjtőkhöz, míg a Link Evaluation bepipálásával az útvonalakhoz való hozzáférést lehet engedélyezni. Mindkettő esetén meg kell adni, hogy milyen paraméterekre vagyok kíváncsi és, hogy milyen járműtípusokat vegyen figyelembe. A szimulációban, én a minimális, átlagos és

maximális sebességet, járműszámot, foglaltságot, forgalomnagyságot, valamint sűrűséget vizsgáltam. Az átlagsebességet pontszerűen az adatgyűjtőn és vonalszerűen a teljes útszakaszra is megvizsgáltam. A megfigyelt járművek esetén minden járművet belevettem a mérésekbe.

A valóságban jelenleg egyedül a hurokdetektorok alkalmasak megbízható pontossággal különbséget tenni az egyes járműtípusok között. Azonban még ezeket sem használják ki, maximum kísérleti vizsgálatok és kutatások esetén. A hurok detektor lényege, hogy az útburkolatban elhelyeznek egy tekercset, amit egy elektronika folyamatos váltakozó feszültséggel táplál. A tekercs körül ennek hatására mágneses tér generálódik, ami a tekercsben egy a vele létrehozóval ellentétes feszültséget indukál. Az elektronikaösszeveti az indukált feszültséget az általa kiküldött meghajtó feszültséggel és a kettő eltérését vizsgálja. Ha nem jön jármű, akkor a kettő megegyezik, ha jön, akkor viszont vasmagként viselkedik és eltorzítja a tekercs körül létrejött mágneses teret, ami ez által az eredetitől eltérő feszültséget fog indukálni. Minél nagyobb a vasmag, vagyis a jármű, annál jobban elhangolja a mágneses teret és annál nagyobb lesz a kiküldött és a visszakapott feszültség közötti különbség. Az elektronika ebből tud következtetni a jármű típusára.

A szimuláció elkészülte után, még érdemes beállítani a szimuláció sebességét, de ha nem tesszük meg, a futtatás közben a billentyűzet numerikus területén található ’+’ és ’-’ gombokkal gyorsíthatjuk, illetve lassíthatjuk a szimuláció futását.

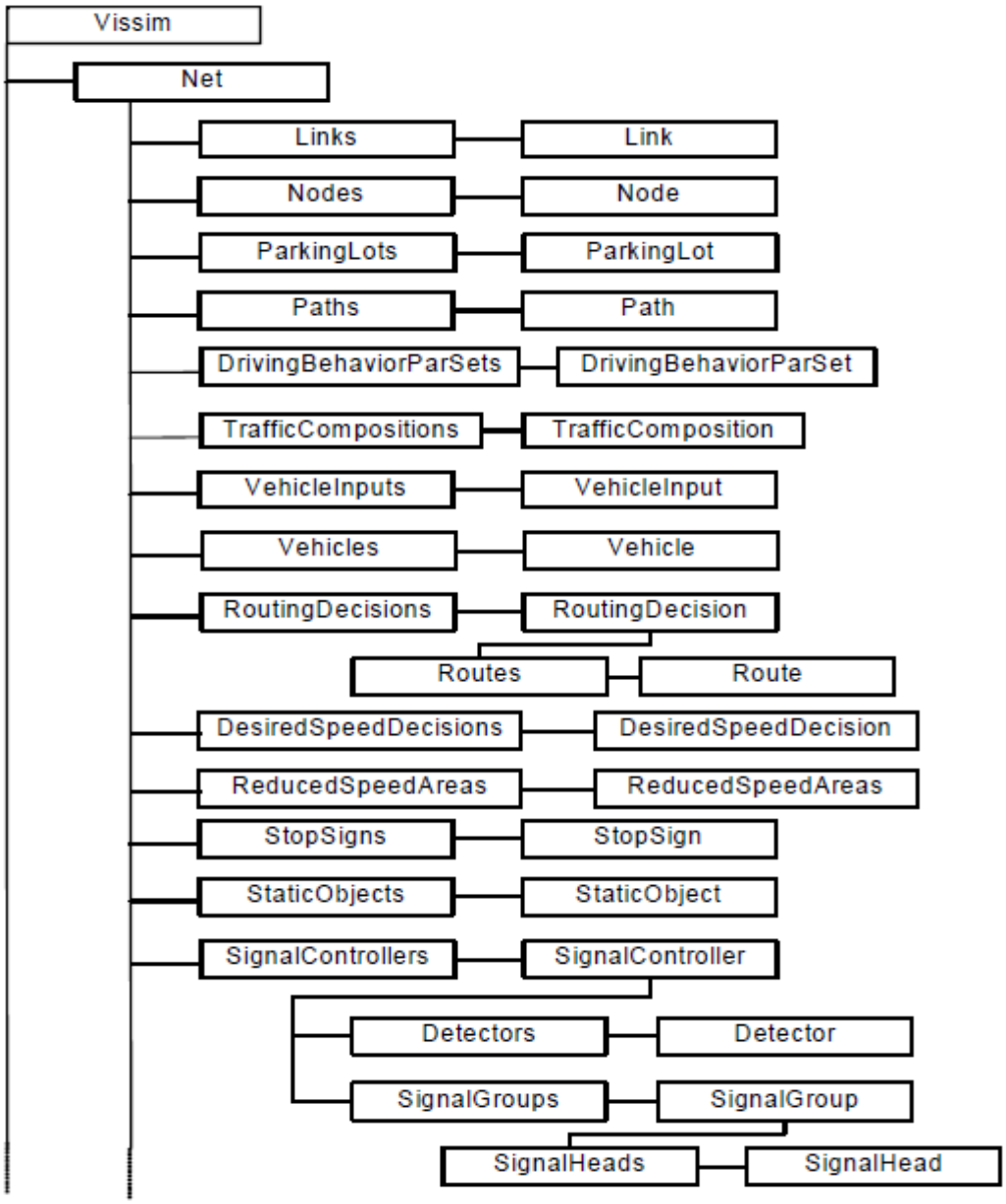
10.2 COM programozás [9.]

Miután elkészítettem a szimulációt és minden paramétert beállítottam, következett a COM-on keresztüli programozás.

A COM használatához a PTV AG biztosít egy help dokumentumot, ami tartalmazza a használható parancsokat, de emellett szükség van VisualBasic, C++ vagy Java nyelv ismeretére is. Én egyelőre C++ nyelven megírt programot használtam. A kód írásához VisualStudio-t használtam. Fordítás előtt szükséges a project ->properties menüpont alatt

néhány beállítást megcsinálni, melyek a következők:

- Debugging: a CommandArguments után be kell írni a *.inp és *.ini-t, az inp kiterjesztés maga a szimuláció, a * helyére mindig az aktuális szimuláció nevét kell írni. Az ini pedig az adott szimuláció vizuális beállításait tartalmazó fájlt, szintén az aktuális szimulációhoz tartozó fájl nevét kell beírni. Ezek arra kellenek, hogy a COM-ban elkészített programmal elindíthassam a szimulációt, amiből lekérem az adatokat.
- A Linker ->General-on belül meg kell adni, hogy hova és milyen névre fordítsa be a VisualStudio a programot. Azén esetemben egy test2.exe állományt hoz létre. Ez egy könyvtárba kell, kerüljön a fentebb megadott *.inp és *.ini fájlokkal.



23. ábra: Interfészek fastruktúrája (forrás: PTV AG – VISSIM_COM Manual [9.])

A Vissimben a szimuláció minden része objektumként van kezelve, amik interfészekbe vannak csoportosítva és a fenti képen látható fastruktúrába vannak szervezve. A legalsó szinten találhatóak az egyes elemek, felettük pedig az interfészek, amik magukba foglalják őket. Ez a modellnek természetesen csak egy részlete, a teljes szerkezet ennél lényegesen kiterjedtebb. Látható például, hogy minden Link külön objektum. A Vissimben Link-nek nevezzük az útszakaszokat, amiket összekötünk egymással, minden linknek külön azonosító száma van, ahogy minden más elemnek is, ezen azonosító segítségével lehet hivatkozni rájuk COM-on keresztül. Ezek egy Links nevű gyűjtő csoportba tartoznak, ami a Net-en belül, az pedig a Vissimen belül található. Ennek azért van jelentősége, mert a COM használatakor ezekre pointerek (mutatók, ezek olyan eszközök a programozásban, melyek nem magát a keresett értéket tartalmazzák, hanem a keresett érték memóriabeli címét) segítségével hivatkozunk és ezek használata előtt végig kell vezetni az adott elem elérési útján ebben a struktúrában.

A feladathoz a következő pointereket definiáltam:

```
static ISimulationPtr spSim(0);
static ISignalControllersPtr spControllers(0);
ISignalControllerPtr spController_1;
static ILinksPtr spLinks(0);
static ILinkPtr spLk_1(0);
static IEvaluationPtr spEval(0);
static IDataCollectionsPtr spDatas(0);
static ISignalGroupsPtr spSignalGroups(0);
static IDataCollectionPtr spData_3(0);
static IRoutingDecisionsPtr spRDs(0);
static IRoutingDecisionPtr spRD(0);
static IRoutesPtr spRoutes(0);
static IRoutePtr spRoute(0);
```

Az I*Ptr elnevezésűek lényegében egy pointer típusok, ezek az interfészeket takarják, az sp* nevűek pedig az adott interfészbe tartozó elemekre mutató pointerek. Ezek közül egyelőre nem használta, mindet, mint pl. az `ISignalControllerPtrspController`-t, amik a jelzőlámpákat vezérlő kontrollerekre mutatnak, de később forgalomfüggő fázisterv kialakításakor szükség lesz rájuk.

```
IVissimPtr spVissim(__uuidof(Vissim));
      if (spVissim == 0)
      {
            cout <<"Cannot start Vissim -> exiting..."<< endl;
            return -1;
      }
            cout <<"VISSIM started"<< endl;

      cout <<"Loading net: path="<< input.c_str() << endl;
      spVissim->LoadNet (input.c_str(), 0);
      INetPtr spNet = spVissim->GetNet();
      if (spNet == 0)
      {
            cout <<"Cannot load net -> exiting..."<< endl;
            throw"Error";
      }

      spControllers = spNet->GetSignalControllers(); //controllerek
      if (spControllers == 0 || spControllers->GetCount()==0)
```

```
{
      cout << endl <<"Error: No junctions defined in "<<
      input.c_str() << endl;
      throw"Error";
}

spLinks = spNet->GetLinks();              //Links
spDatas = spNet->GetDataCollections();    //DataCollection
spEval = spVissim->GetEvaluation();       //evaulation
                                          beallitasahoz
spSim = spVissim->GetSimulation();        //Simulation
if (spSim == 0)
{
      cout <<"Cannot access simulation object -> exiting..."<<
      endl;
      throw"Error";
}
```

Ezek a sorok a main függvénybe kerültek és ezek határozzák meg az interfészekre és azok elemeire mutató pointereket. Ez után már szabadon használhatom őket.

10.3 Paraméterek lekérése

A COM minden Vissimből érkező és Vissimbe továbbítandó értéket egy _variant_t típusú változóban tárol. Ezt a szoftver gyárilag ismeri, így nem igényel külön definiálást. Ez azonban nem konkrét típust takar, mind szöveges, mind szám típus kerülhet bele, és lehet sima változó, vagy akár tömb is, így bármilyen értéket tárolhatunk benne. Ha a benne tárolt értékeket fel szeretnénk használni a programon belül számítási műveletekre, vagy akár csak kiírni a képernyőre vagy fájlba a felhasználók számára, akkor célszerű egy külön definiált változóba átírni az értékeket. Az átírás szerencsére nem jelent különösebb gondot, bármiféle köztes lépés nélkül egyenlővé tehetjük string, byte, int, stb. típusú változókkal.Ugyanakkor azt a programozónak kell észben tartania, hogy az adott esetben milyen típusú értékre számít, és ennek megfelelően kell típust választania, különben ha például a _varinat_t-t egy int típusú változóval teszi egyenlővé, miközben az értékbe szöveges karakterek is kerülnek, akkor a program hibát fog jelezni és nem lesz képes működni.

```
int ciklusido=10; // ciklusido=10 sec
            _variant_t cikl=0;

            for (int i=0; i<30; i++) { // pl. i=30 -> 30 darab
                                       ciklusunk lesz most
            cikl=ciklusido; // atrakjuk _variant_t tipusba

            spSim->PutAttValue("BREAKAT", cikl);
            spSim->RunContinuous();

            ciklusido=10*(i+1);
            cout<<" NEXT BreakAt: "<< ciklusido<<endl;
```

A következő programész egy megszakítást generál. Ez annyit jelent, hogy létrehoz a for ciklussal 30db 10 másodperces ciklust. A 10 mp alatt fut a szimuláció és minden ciklus

végén egy pillanatra megállítja a program. Ez az emberi szem számára nem észlelhető akadást jelent, a számítógépnek viszont elegendő időt ad, hogy a processzor túlzott leterhelése nélkül, amíg a szimuláció szünetel, addig könnyedén lekérdezhesse és kiszámolhassa a keresett paramétereket, értékeket. A továbbiakban ez a rész is átalakításra kerül. Nem fogom egy tetszőleges értéknek megadni a ciklusok hosszát. Ehelyett a COM-on keresztül lekérem a Vissimtől, hogy milyen hosszúságúra lett beállítva a szimuláció és ezt az időtartamot fogomfelbontani annyi részre, ahány mérést szeretnék. Ezzel biztosítható, hogy a szimuláció hosszának bármilyen változtatása esetén is végig kapjak adatokat a szimulált forgalomról, ne csak az első x másodpercéről.

A main-ből ezután meghívok egy általam test-nek nevezett függvényt.

```
void test(){
	fout.open("adatok.txt");
	for(int link_szam=1;link_szam<10;link_szam++)
		{
		linkspeed(link_szam);
		density(link_szam);
		volume(link_szam);
		}
	for(int dc_szam_1=1;dc_szam_1<10;dc_szam_1++)
		{
		int dc_szam = 0;
		if(dc_szam_1==8)
			{
			for(int dc_szam_2=1;dc_szam_2<3;dc_szam_2++)
				{
				dc_szam = dc_szam_1*10+dc_szam_2;
				dcspeed(dc_szam);
				occupancy(dc_szam);
				}
			}
			else
				{
				dc_szam = dc_szam_1;
				dcspeed(dc_szam);
				occupancy(dc_szam);
				}
		}
	fout.close();
	cout<<"COUNTER 1 "<< endl;
	Sleep(500);
	cout<<"COUNTER 2 "<< endl;
	Sleep(500);
```

Ez függvény arra szolgál, hogy ebben bővítsem az elvégzendő műveleteket, újabb függvényhívásokkal, vagy korábbiak inaktiválásával. Ez azért jó, mert így a `main`függvényemben egy letisztult, könnyen átlátható, csak az alap dolgokat tartalmazó rész marad és minden további bővítést, módosítást ebben a részben tudok megvalósítani. Ezen kívül az áttekinthetőséget is nagyban javítja, mert így a különböző műveletek, paraméter lekérdezések biztosan nem fognak összekeveredni a pointer definíciókkal vagy a szimuláció beállításaira vonatkozó metódusokkal, hanem azoktól jól elkülönített helyen fognak elhelyezkedni.

A lekérdezett értékeket átmenetileg egy `adatok.txt` fájlba írom. Ennek megnyitására szolgál az `fout.open` parancs, a függvény végén pedig az `fout.close` zárja le. Minden

paramétert külön függvény segítségével kérek le. A lekérdezett paramétereket két csoportba lehet sorolni, egyik, amiket a linkekről kérdezek le, ezekre az adott link számával lehet hivatkozni, a másik csoport pedig, amiket az utakon elhelyezett adatgyűjtő (datacollections) pontokról kérdezek le, ezekre az érzékelők számával (`dc_szam`) lehet hivatkozni. Mindkettőre 1-1 for ciklust használtam, ami egymás utána az összes meglévőre meghívja a szükséges függvényeket. A dc-kről lekérdezett adatoknál kellett egy feltételes for ciklus. Erre azért volt szükség, mert a Vissimben több sávú utak esetén lehetőségünk van az út létrehozásakor beállítani, hogy hány sávos legyen, de az is megoldható, hogy a sávok számának megfelelő számú egy sávos utat húzunk egymás mellé. Általában véve az egy db, több sávos út készítése tűnik célszerűbbnek, hiszen direkt erre a célra lett kitalálva ez az opció. Ebben az esetben azonban mindenképp legalább 2 utat kellett volna létrehozni, mert az 1 sávos út 2 sávossá bővülése csak úgy oldható meg, ha létrehozok egy 1 sávos, majd a bővülés keresztmetszete után egy kétsávosat és így kötöm össze, vagy másik megoldás, hogy az út teljes hosszában készítek egy egysávos utat és a kibővülés után mellé helyezek egy másik egy sávosat és így kötöm össze őket. Én mind a két lehetőséget kipróbáltam. Ahol 2 sávos utat hoztam létre, ott a két sávra kerülő detektort célszerűen dupla indexeléssel jelöltem, melynek első számjegye az út számát, második számjegye pedig a sáv számát jelenti. Ez azén esetemben egy 81-es és egy 82-es számú érzékelőt eredményezett, miközben a többi dc egy számjegyű jelzést kapott. Emiatt nem lehetett volna megoldani, hogy egy sima for ciklus minden érzékelőt magába foglaljon, hiszen a 9-es után a 81-es a következő, így a 10-hez érve a program megszakadt volna érvénytelen dc azonosító miatt. Ennekérdekében a 8-as számú dc-hez érve elindítottam egy újabb for ciklust és így állítottam elő a 81 és 82-es azonosítókat. A Vissimben az utakra vonatkozóan az Evaluation menüpontban lehetőségvan aktiválni egy ’perlane’ kapcsolót, aminek segítségével több sávos utaknál sávonként végzi el a program a méréseket. Ennek köszönhetően a linkekre vonatkozó adatok mérésekor nem kellett külön figyelmet fordítani arra, hogy az egyik 2 sávos irány 2 db 1 sávos útból, míg a másik 1 db 2 sávosból áll.

10.4 Lekérő függvények

```
void volume(int linkszam)
{
spLk_1 = spLinks->GetLinkByNumber(linkszam);
_variant_t vol = spLk_1->GetSegmentResult("VOLUME", 0, 0, 1, false);
int volume = vol;
cout<<linkszam;
cout<<". ut forgalom nagysag: ";
cout<<volume;
cout<<" [jm/h]"<< endl;
fout<<volume<<endl;
}
```

A függvény nem rendelkezik visszatérési értékkel, a lekérdezett adatot jelen esetben egyből kiírja a képernyőre és egy fájlba is. Később vagy függvényen belül átadja az értéket egy globális változónak, vagy a függvényt átírom, hogy visszatérési értékkel rendelkezzen, amit egyből egy változó értékeként fogok megadni. A függvény meghívásakor egyetlen paramétert adok át, a vizsgálandó link számát. Ezután az

spLk_1pointernek értétek adva a GetSegmentResult függvénnyel lekérdezem a kívánt adatot. A függvény paraméterei a következőek:

1. A kérdéses forgalomtechnikai paraméter (volume: forgalomnagyság)
2. A figyelembeveendő járműkategóriák (0: mind)
3. Azt adja meg, hogy az adott útnak hányadik méterétől kezdjen mérni a program
4. Az adott út sávjainak száma (a 'perlane' kapcsoló miatt lehetett minden útnál 1-et írni)
5. Kumulatív értékelés (false az alapbeállítás)
6. Akkor használatos, ha a _variant_t változót tömbként kívánjuk kezelni, nekem nem volt rá szükségem, ezért nem került be a paraméterek közé

Az értéket ezután átírtam egy int típusú változóba, majd a cout függvény segítségévek kiírtam a képernyőre, az fout segítségével pedig a 't_main' függvényben megnyitott fájlba.

Az utakon mért további két paraméter lekérése ugyan így történik, a különbség csak annyi, hogy a GetSegmentResult függvény első paraméterét átírom forgalomsűrűség esetén DENSITY-re, az adott úton mért sebesség esetén SPEED-re.

A dc-kről a következő függvényekkel kérdezem le az adatokat:

```
void occupancy(int dcszam)
{
spData_3 = spDatas->GetDataCollectionByNumber(dcszam);
_variant_t occ = spData_3->GetResult("OCCUPANCYRATE", "SUM", 0);
int occupancy = occ;
cout<<dcszam;
cout<<". adatgyujton mert foglaltsag: ";
cout<<occupancy;
cout<<" [%]"<<endl;
fout<<" - ";
fout<<occupancy<<endl;
}

void dcspeed(int dcszam)
{
spData_3 = spDatas->GetDataCollectionByNumber(dcszam);
_variant_t spdmax = spData_3->GetResult("SPEED", "MAX", 0);
int dcspeedmax = spdmax;
_variant_t spdmin = spData_3->GetResult("SPEED", "MIN", 0);
int dcspeedmin = spdmin;
_variant_t spdavg = spData_3->GetResult("SPEED", "MEAN", 0);
int dcspeedavg = spdavg;
cout<<dcszam;
cout<<". adatgyujton mert sebessegek: ";
cout<<"max: ";
cout<<dcspeedmax;
cout<<" [km/h]";
cout<<" min: ";
cout<<dcspeedmin;
cout<<" [km/h]";
cout<<" atlag: ";
cout<<dcspeedavg;
```

```
cout<<" [km/h]"<<endl;
fout<<dcspeedmax;
fout<<"-";
fout<<dcspeedmin;
fout<<"-";
fout<<dcspeedavg;
}
```

Ezeknek a függvényeknek egyelőre szintén nincs visszatérési értékük, a későbbiek folyamán ez is módosításra kerül. Itt az adott dc számát adom át paraméterként meghíváskor. Ugyan úgy meg kell csinálni a pointer (`spData_3`) értékadását, majd pedig a dc-khez tartozó `GetResult` függvénnyel lekérdezem az értéket. A függvény paraméterei:

1. A kérdéses forgalomtechnikai paraméter (occupancyrate: foglaltság [%], speed: sebesség)
2. Funkció neve (min, max, mean /átlag/, frequencies /gyakoriságok/)
3. Vizsgálandó járműtípusok (0: mind)
4. Sima vagy tömbváltozó, szintén nem használjuk

Ezek után az értékeket átírom a megfelelő változókba és kiírom a képernyőre, valamint az `adatok.txt` fájlba. A sebesség lekérdezésekor egy függvényen belül kérdezem le a minimális, maximális és átlagsebességet. A kiírás mind a képernyőre, mind a fájlba formázva, tagoltan történik a jobb átláthatóság és a későbbi könnyebb felhasználhatóság kedvéért.

10.5 Socket kezelő kliens

A socketet kezelő kliens megvalósításához először is további két fájl csatolása volt szükséges a programhoz, ezek a René Nyffenegger által már elkészített Socket.h és a Socket.ccp[13.]. Ezek az interneten megtalálható és ingyenesen felhasználható fájlok, így ezek részletezésére most nem térnék ki. Amikor ezeket importálásra kerültek, akkor következett a kliens megírása. Ehhez először definiálni kellett egy host-ot, ami a szervergép IP címét takarja, majd keresni kellett egy szabad portot, ami használható a kommunikációhoz, itt végül a 7750-es port mellett döntöttem.

A kliens egy külön függvényben egy külön szálként került megírásra, ami annyit jelent, hogy meghívás után folyamatosan, a program többi részével párhuzamosan fut, így lehetővé téve, hogy azonnal érzékelje, ha üzenetet kap a másik programból.

```
DWORD ClientThread (LPVOID lpdwThreadParam )
{
      try {
            SocketClient vmsManager(HOST, PORT);
            stringstream out;
            out<<"\n"<<oBalra<<" "<<oEgyenesen<<" "<<oJobbra<<"\n";
            vmsManager.SendLine(out.str());
            while (isRunning) {
              string msg = vmsManager.ReceiveLine();
              istringstream input(msg);
            double x;
```

```
            if (input >> x)
                 Balra = x;

            if (input >> x)
                 Egyenesen = x;

            if (input >> x)
                 Jobbra = x;

            //Itt kell ertelmezni a kapott sort!
              cout << Balra <<" "<< Egyenesen <<" "<< Jobbra << endl;
              cout.flush();
            }
      } catch (...) {
            cerr <<"Nem tudok csatlakozni a VMS manager-hez"<< endl;
      }
      return 0;
}
```

Az egész egy try függvénybe kerül, ami nevéből adódóan megpróbálja megvalósítani a benne található programot, ha nem sikerül, akkor pedig hibaüzenetet dob, amit a catch függvény képes elkapni és jelen esetben tájékoztatást ad a felhasználónak. A `SocketClient` egy külön osztály, ami a Socket.h-ben került létrehozásra, ebbe az osztályba kerül új elemként a vmsManager nevű változó, ami nevét a Java nyelven megírt grafikus programról kapta, amihez, mint szerverhez kapcsolódik. Ezt egy `stringstream` típusú változó létrehozása követi, ebbe a változóba kerülnek bele a szimulációból kiolvasott eredeti fordulási ráták, amiket a program tovább küld a VJT-ket vezérlő programnak. A `SendLine` a `SocketClient` függvénye, ami a paraméterként megadott változót elküldi a megadott IP címre, a megadott porton keresztül. Látható, hogy az elküldendő értékek egymás után, 1 db szóközzel elválasztva kerülnek be a változóba, ami azt a célt szolgálja, hogy a másik oldali program könnyen szétbonthassa az üzenetet és azonosíthassa, hogy melyik szám mit jelent.

Ezt követi az üzenet fogadása rész, amit egy `while` függvénybe foglaltam, amelynek az `isRunning` változót adtam meg feltételként. Ez egy boolean, vagyis logikai változó. A program elindításakor igaz (true) értéket kap, ezzel teljesíti a ciklus függvény feltételét, majd a program végén hamis (false) érték kerül beállításra, a `while` ciklus tehát a program elejétől a végéig fut, amíg az `isRunning` változó igaz értékű. Ezt követően behozok egy msg nevű string (azaz szöveg) típusú változót, amelynek a `SocketClient` osztály `ReceiveLine` függvényével a socketen keresztül kapott adatot adom meg értékként. A msg változót ezután átadom egy `istringstream` típusú`input`változónak, amely képes arra, hogy a kapott üzenet elejéről elindulva az első szóköz karakterig talált értéket beírja a megfelelő változóba, majd a szóköz után folytatja és a következő szóközig lévő értéket szintén beírja egy változóba és így halad tovább az üzenet végéig vagy addig, amennyit a felhasználó ki akar olvasni belőle. A kapott üzenetből kilvasott értékeket egy `double` típusú `x` változóba írom, amiből ezután átkerülnek a megfelelő forgalmi paraméterbe.

10.6 Forgalmi paraméterek módosítása

```
spRD = spRDs->GetRoutingDecisionByNumber(3);
//3-as számú útválasztó döntés
spRoutes = spRD->GetRoutes();

spRoute = spRoutes->GetRouteByNumber(1);
spRoute->GetAttValue1("RELATIVEFLOW",&oBalra);

spRoute = spRoutes->GetRouteByNumber(2);
spRoute->GetAttValue1("RELATIVEFLOW",&oEgyenesen);

spRoute = spRoutes->GetRouteByNumber(3);
spRoute->GetAttValue1("RELATIVEFLOW",&oJobbra);
```

Ez a kód részlet felel azért, hogy az adott paramétert módosítani lehessen a COM felületen írt programon keresztül. A Vissim minden elemére, minden beállítható paraméterre egy megfelelő pointerrel (mutatóval) lehet hivatkozni, így először ezeket kellett meghatározni. A munka során a fordulási ráták kerültek befolyásolásra, így az ezekhez tartozó pointereket kellett meghatározni. A fordulási rátákhoz négyféle pointer tartozik:

- Routing Decisions
- Routing Decision
- Routes
- Route

A pointerek hierarchikus rendje is ennek felel meg (23. ábra: Interfészek fastruktúrája (forrás: PTV AG – VISSIM_COM Manual [9.])). Legfelül a RoutingDecisions csoport van, ami magába foglalja az összesen olyan pontot, ami az úthálózaton telepítésre került, ezeknél a pontoknál döntik el a járművek, hogy a következő lehetőségnél melyik útvonalat használják tovább haladásra. Ezeken belül találhatóak a RoutingDecision-ök, amik egy-egy konkrét döntési pontra mutatnak. Egy ilyen döntési ponton belül több (legalább két) lehetőség van, amik közül választhatnak a járművek, ezeket a lehetőségeket foglalja magába a Routes nevű csoport és erre mutat a Routes pointer. Ezen a csoporton belül pedig a Route pointerek mutatnak az egy-egy konkrét választási lehetőségre. A kívánt fordulási ráta beállításához tehát először meg kell adni, hogy a RoutingDecisions-ökön belül melyik RoutingDecision-ről (döntési pontról) van szó és azon belül a Routes csoportból melyik Route (maga a döntés) értékét kívánom változtatni. Jelen esetben mivel pusztán tesztelési célokat szolgál ez a program, így csak a 3-as sorszámú RoutingDecision-ön belüli három darab Route értékét változtattam. Mindegyiknek a Socketenkeresztül érkező üzenetben kapott értéket állítja be a program.

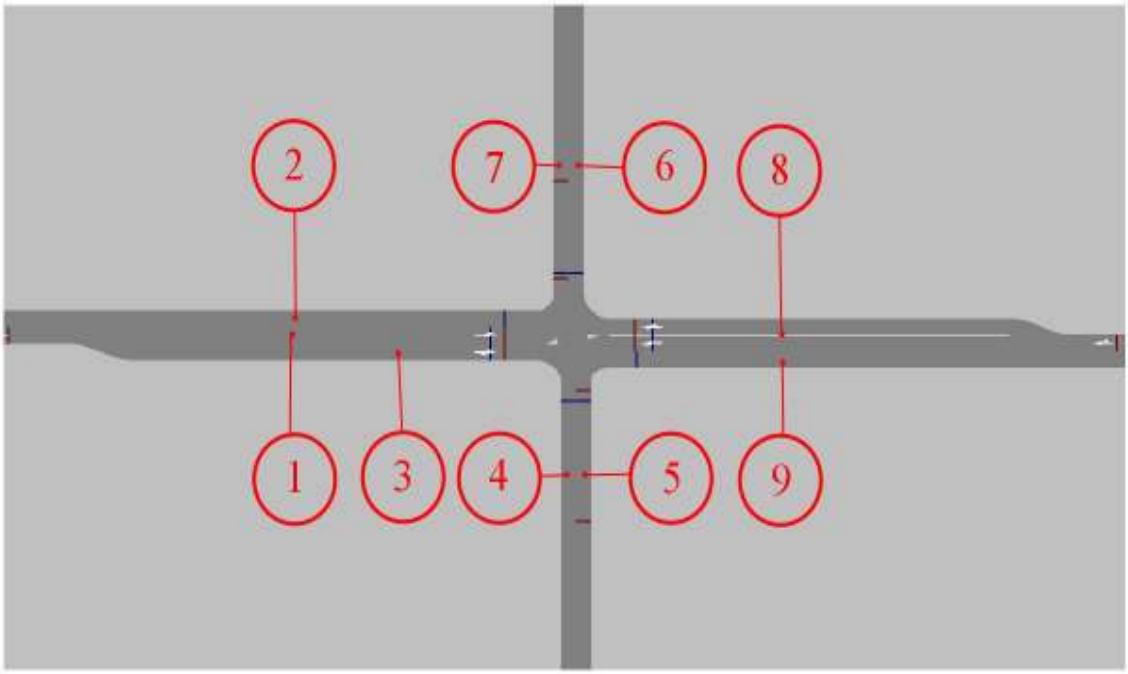
11. Szimulációs eredmények

A forgalmi és gazdasági kutatások alapján a VJT-k használata jelentősen képes javítani a forgalom minőségét és gazdasági mérlegét, tehát igazolható az az álláspont, hogy érdemes volt belekezdeni a terület fejlesztésére irányuló munkálatokba.

Munkám során sikerült elkészíteni egy programot, amely a Futurit protokollra épülve képes utasításokat küldeni a VJT-knek és olyan állapotba került, ahonnan egy-két hónapon belül jelentős mértékben továbbfejleszthető. A program tesztelésére szánt Vissim szimuláció maradéktalanul elkészült, esetleges további simítások még végrehajthatóak, de funkcionális értékét már nem fogják érdemben befolyásolni. A szimulációhoz sikerült elkészíteni azt a COM felületen megírt programot, amelynek segítségével lehetőségem van a szimulált forgalom paramétereinek lekérdezésére, illetve akár szimuláció közbeni befolyásolására. Ahhoz, hogy a VJT-ket vezérlő program működését ténylegesen ellenőrizni lehessen vele meg kellett oldani a két program kommunikációját. Ezt socketek használatával sikerült megvalósítani, ennek keretében a Vissimhez tartozó programba bekerült egy kliens elem, a VJT-ket vezérlő Libet, pedig eredeti funkciójának megfelelően beleépítettem egy külön erre a célra megírt grafikus kezelőfelületű programba, ami magában foglal egy szervert is. A grafikus programot elindítva az lefoglal és megnyit egy előre megadott socketet, majd várja, hogy egy kliens csatlakozzon hozzá. A COM felületen írt program az indulásakor automatikusan kapcsolódik ehhez a szerverhez és innentől megkezdődik a két program kommunikációja. A példaprogrammal annak a vizsgálatát tettem lehetővé, hogy egy VJT-vel való útvonalajánlás milyen módon képes befolyásolni a többi útszakasz forgalmát.

A vizsgálatot az alábbi módon végeztem:

A szimulációt a következő 4 ágú csomóponton futtattam:



24. ábra Vizsgált csomópont

A számok az egyes utak azonosító számát jelölik. A vizsgálat során az 5-ös számú bemenő iránynak tettem a következő útvonalajánlásokat:

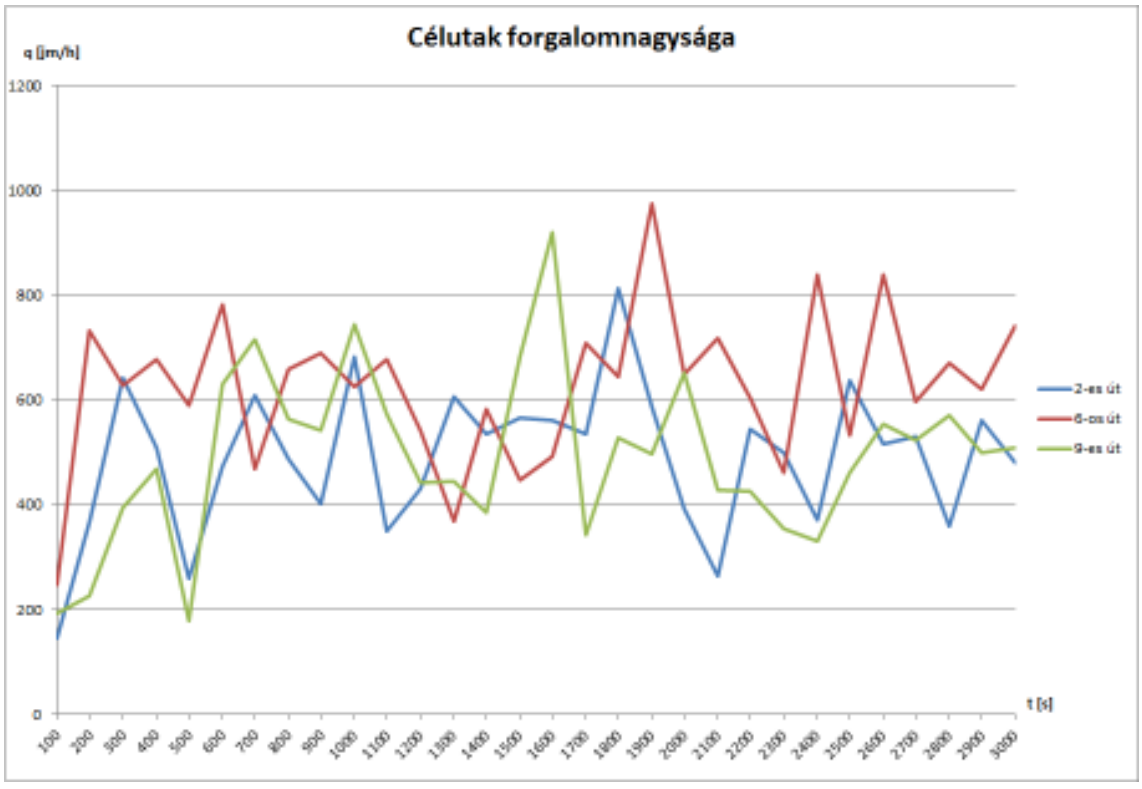
- balra ajánlott
- egyenesen ajánlott
- jobbra ajánlott
- balra nem ajánlott
- egyenesen nem ajánlott
- jobbra nem ajánlott

Továbbá azt is beállítottam, hogy a közlekedők hány százaléka vegye figyelembe az ajánlást.

Így összesen négyszer futtattam le a tesztet:

1. 0% senki nem veszi figyelembe, ez egyúttal az alapállapot, mintha nem is adtam volna ajánlást
2. 30%
3. 60%
4. 100% tehát mindenki azt csinálta, amit előírt neki a VJT

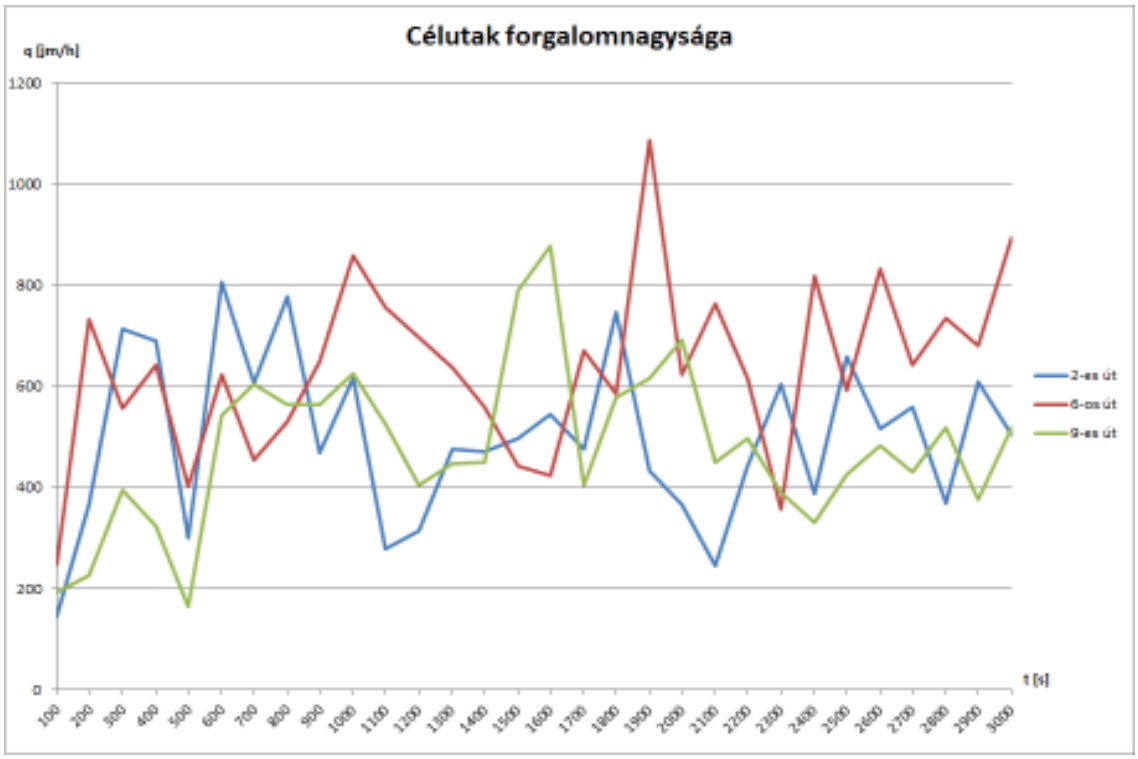
A művelet során a 2-es, 6-os és 9-es számú kimenő utak forgalomnagyságát vizsgáltam és a következő eredményeket kaptam:



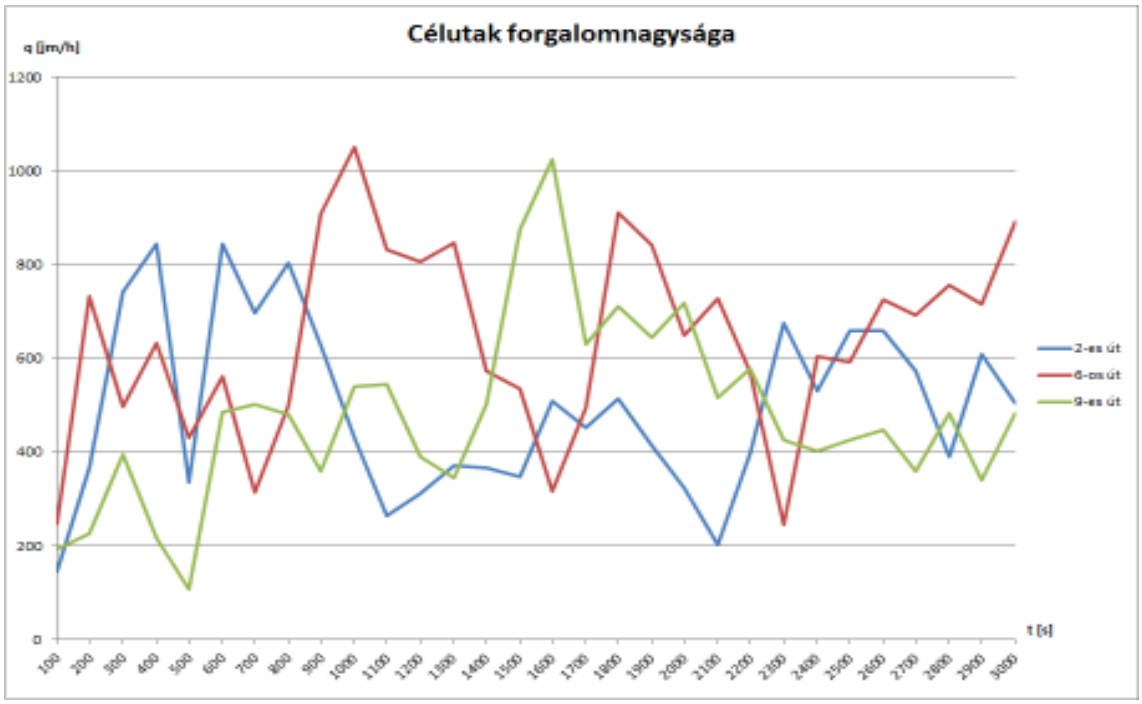
25. ábra Alap forgalomnagyságok

3000 másodperces szimuláció alatt, összesen 300 mérést végeztem, a jobb átláthatóság kedvéért, azonban ebből előállítottam 30 átlagértéket. A még ennek ellenére is erős ugrások annak köszönhetőek, hogy a csomópontot jelzőlámpa irányította és a közbenső idők betartása miatt gyakran esett olyan időpillanatra a mérés, amikor egy váltás során az átlagosnál kevesebb jármű tartózkodott az adott útszakaszon.

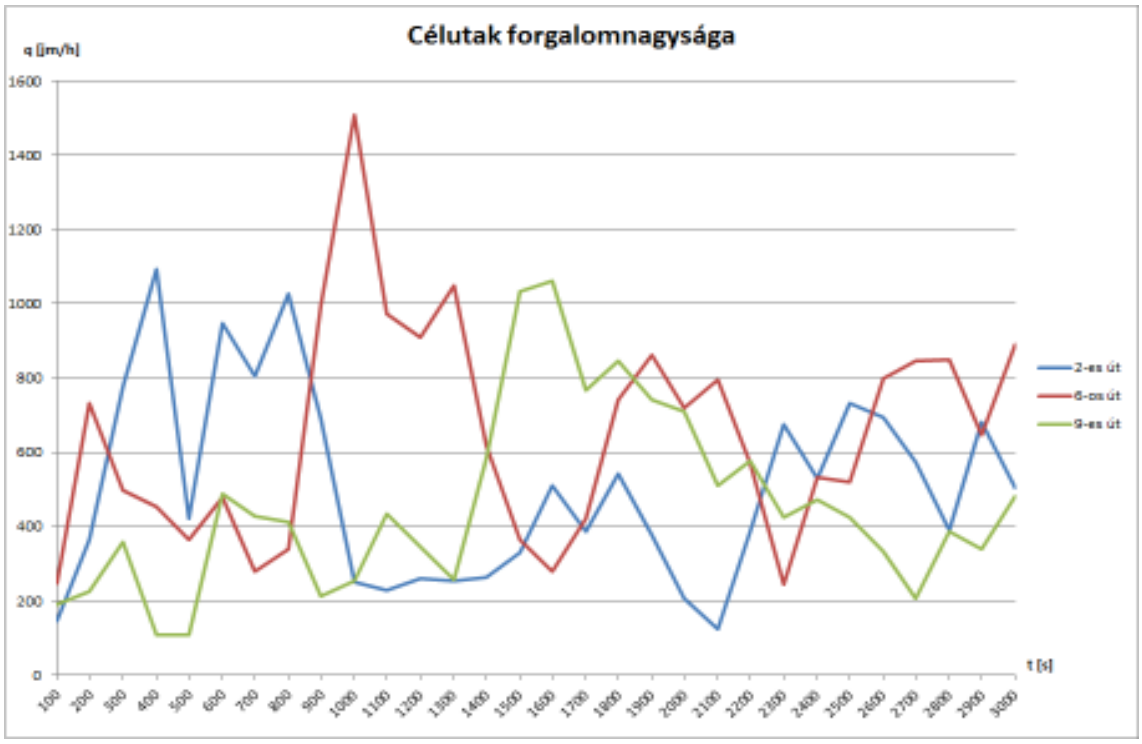
Befolyásolt forgalom esetén a következő eredmények adódtak:



26. ábra Forgalomnagyságok 30%-os elfogadás esetén



27. ábra Forgalomnagyságok 60%-os elfogadás esetén



28. ábra Forgalomnagyságok 100%-os elfogadás esetén

Egy-egy szimuláció lefuttatása során a következő beállításokat használtam:

- a követési arányt fixen beállítottam 0, 30, 60 és 100%-ra
- 200 mp elteltével adtam ki a balra ajánlott jelzést, addig az alapértékekkel futott
- 800 mp után adtam ki az egyenesen ajánlott jelzést
- 1300 mp után következett a jobbra ajánlott
- 1700 mp-nél a balra nem ajánlott jelzést aktiváltam
- 2100 mp elteltével adtam ki az egyenesen nem ajánlott jelzést
- 2400 mp után következett a jobbra nem ajánlott
- 2900 és 3000 mp között ismét visszaállítottam az eredeti értékeket

A szimuláció futása közben tíz másodpercenként végeztem méréseket. Ezek ismeretében már értelmezhető az a megfigyelés, hogy az ajánlásoknak megfelelően a kétszázadik másodperc után a 2-es úton mért forgalomnagyság megnövekedett, míg a másik kettő lecsökkent, a nyolcadik pont után a 6-os úton mért érték nőtt meg és a 2-es, csökken le a 9-es közelébe, azezerháromszázadik másodperctől kezdve pedig a 9-es növekedett meg. Ezután következtek a nemleges ajánlások, ahol a görbék pont fordítva viselkedtek, a nem ajánlott irányokon mért forgalomnagyságok a megfelelő időpillanatok után lecsökkentek.

A grafikonokon észrevehető egy minimális késés, ami két okból következik:

1. a hálózaton, az elágazás előtt telepíteni kell egy úgynevezett routingdecision pontot, a járművek ezt a pontot elérve döntik el, hogy melyik irányba forduljanak. Miután kiadtam az ajánlást, az ezen a ponton már áthaladt járművek nem változtatták meg a döntésüket, így ők még a korábban életben lévő fordulási ráták alapján döntöttek.

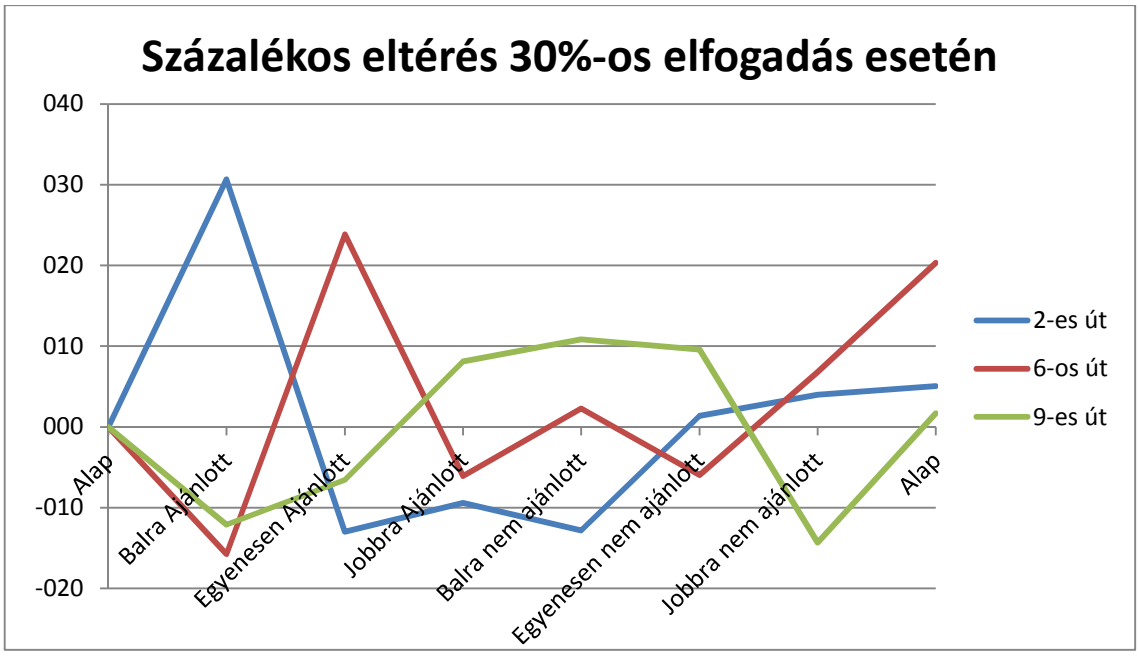
2. amennyiben az ajánlás kiadása olyan időpillanatra esett, amikor a befolyásolt iránynak piros jelzése volt, akkor az ajánlás hatása csak a zöld jelzés megjelenésekor következett be.

A késések mellett egy másik anomália is látható a grafikonokon, ez pedig a görbék erős ugrálása. Ez annak köszönhető, hogy a vizsgálat során csak egy behaladó irány került befolyásolásra, így amikor ennek az iránynak volt zöld jelzése, akkor a kívánt irányba hangolta el a görbéket. Amikor azonban a többi iránynak volt szabad jelzése, akkor azok az eredeti, nem módosított fordulási rátáknak megfelelően visszahúzták a görbéket az eredeti értékek közelébe.

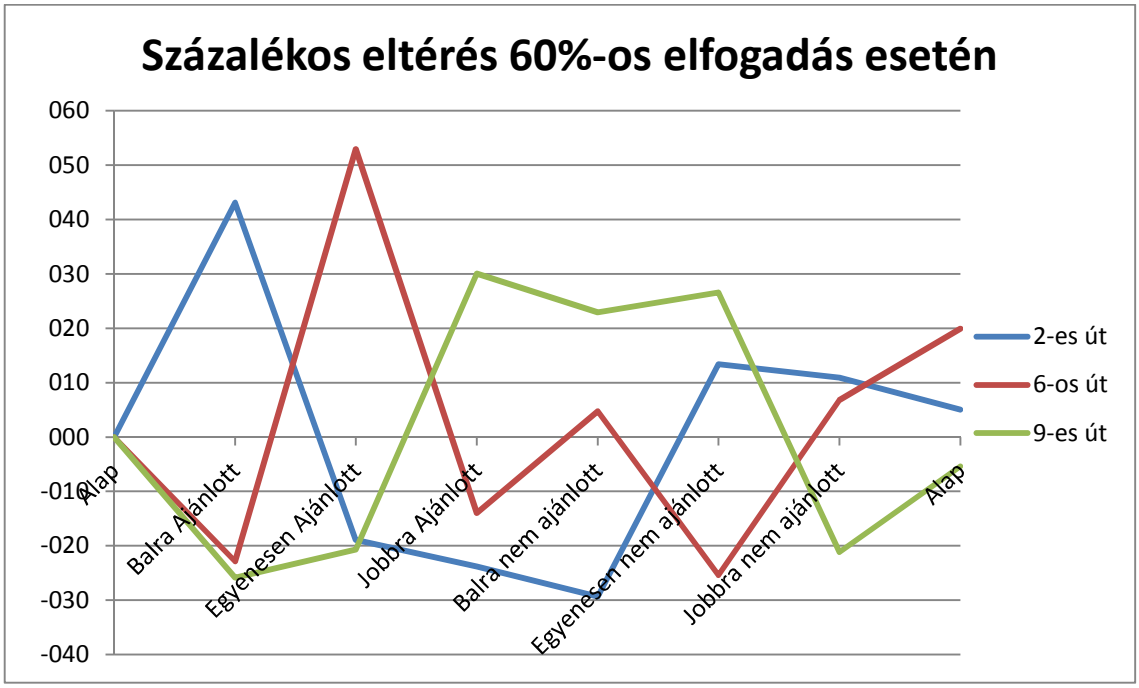
A fenti grafikonokon pontosan követhető a forgalom változása, a könnyebb értelmezhetőség kedvéért azonban meghatározásra kerültek az egyes elfogadási szinteken tapasztalt százalékos változások is. A következő táblázatban és grafikonokon az figyelhető meg, hogy adott elfogadás arány mellett, az eredetihez képest hány százalékkal sikerült befolyásolni a forgalom nagyságát az egyes útszakaszokon.

4. táblázat Az ajánlások által elért százalékos forgalomnagyság-változás, az elfogadási szinteknek megfelelően

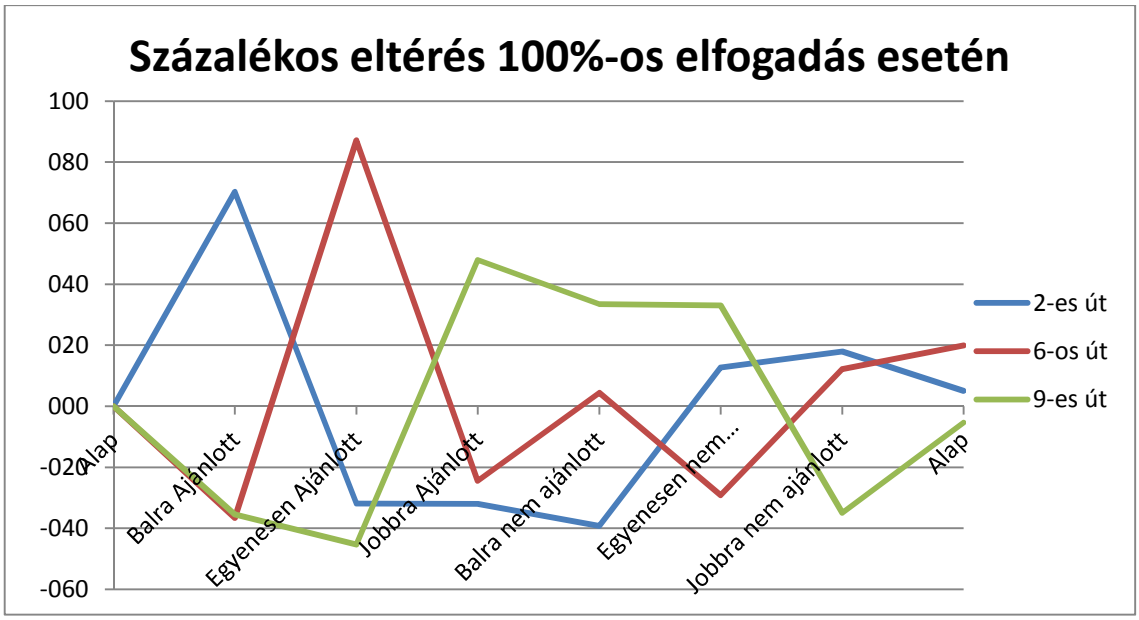
Eltérés (%-ban)								
30%-os elfogadás			60%-os elfogadás			100%-os elfogadás		
2-es út	6-os út	9-es út	2-es út	6-os út	9-es út	2-es út	6-os út	9-es út
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
30,65	-15,74	-12,13	43,10	-22,89	-25,86	70,24	-36,61	-35,48
-12,97	23,86	-6,54	-18,93	52,95	-20,71	-31,94	87,17	-45,28
-9,43	-6,10	8,11	-23,80	-13,95	30,07	-32,05	-24,40	47,93
-12,82	2,30	10,84	-29,35	4,74	22,92	-39,23	4,39	33,43
1,37	-5,98	9,56	13,40	-25,39	26,58	12,66	-29,15	33,04
3,99	6,80	-14,33	10,90	6,78	-21,18	17,95	12,21	-34,95
5,04	20,30	1,69	5,04	19,95	-5,39	5,00	19,98	-5,35



29. ábra A forgalomnagyságok százalékos változása 30%-os elfogadási arány mellett



30. ábra A forgalomnagyságok százalékos változása 60%-os elfogadási arány mellett



31. ábra A forgalomnagyságok százalékos változása 100%-os elfogadási arány mellett

A százalékosan megadott értékeken jól követhetőek az ajánlások. Egyértelműen látszik a forgalom növekedése, az adott útvonal ajánlása esetén és megfigyelhető a forgalom csökkenése, az útvonal választását nem ajánló jelzések esetén.

A vizsgálat során csak egy bemenő ágat irányítottam mégis sikerült az utakon legalább 10%-os, de jellemzően ennél nagyobb változást elérni már 30%-os elfogadási arány mellett is. A valóságban a bevezetést követő néhány hónapban természetesen nem lehet 30%-os elfogadási arányra számítani, viszont egy ág helyett, az összesnek tehetünk ajánlásokat. Továbbá megfelelő szabályozás esetén a közlekedők egyre megbízhatóbbnak fogják tartani az ajánlásokat és ez által növekedni fog az azokat elfogadók aránya. A 100%-os elfogadást, főleg alternatív útvonal ajánlásakor, a közlekedők eltérő mentalitása, úti célja és értékrendje miatt (főként arra vonatkozóan, hogy adott többlet távolságot mekkora időnyereségért cserébe hajlandóak megtenni) nem valószínű, hogy valaha is elérhetünk, de mint látható már 30% esetén is jelentős mértékű változást lehet elérni, ami pedig egy reálisan elvárható hatékonyság.

Az ajánlás megvalósítását a már említett fordulási ráták beállításával oldottam meg, ami azt fejezi ki, hogy egy adott bejövő útról melyik irányba a járművek hány százaléka halad tovább. Ezek alaphelyzetben egyenlően oszlanak meg az irányok között, míg az ajánlás kiadásakor az elfogadás függvényében arányosan eltolódnak valamelyik irány előnyére vagy hátrányára. A pontos értékek meghatározására a következő függvényt alkalmaztam:

(1) $$x = (1 - KA) * a + KA * (a + b + c) * \frac{x}{a+b+c}$$

Ahol:

- x, y, z: a ténylegesen beállításra kerülő fordulási ráták
- e, f, g: az adott irányhoz tartozó elvárt fordulási ráta (bal, egyenes, jobb)
- KA: követési arány, vagyis, hogy a közlekedők hány százaléka veszi figyelembe az ajánlást
- a, b, c: az eredeti (1, 1, 1) fordulási ráták

A hatfél ajánláshoz megadtam a hozzájuk tartozó kívánt fordulási rátákat:

- balra ajánlott (1, 0, 0)
- egyenesen ajánlott (0, 1, 0)
- jobbra ajánlott (0, 0, 1)
- balra nem ajánlott (0, 1, 1)
- egyenesen nem ajánlott (1, 0, 1)
- jobbra nem ajánlott (1, 1, 0)

Ezeket az értékeket, a követési arányt, valamint a szimulációban használt eredeti fordulási rátákat átadva a függvénynek, 0 és 100 között bármilyen egész értékű követési arányra a megfelelő pontos számhármast fogja eredményül adni. A képlet magába foglalja azokat, akik eredetileg (is) a kijelölt irányt választották volna és azokat is, akik az ajánlás nélkül (is) más irányt választottak volna, majd az eredeti fordulási rátákat a kívánt fordulás rátáknak és a követési aránynak megfelelően újra szétosztja az irányok között. Az eredeti értékeket a COM felületen írt program segítségével olvasom ki a szimuláció indulásakor és átadom az ezt a képletet tartalmazó programnak, így ez mindig hiteles adatokkal tud dolgozni, nem szükséges manuálisan korrigálni az eltéréseket.

Példa:

Az egyenes irányt ajánljuk és 50%-os elfogadást feltételezünk.

- Az eredeti fordulási ráták:
  - a (bal)=1
  - b (egyenes)=1
  - c (jobb)=1

- Kívánt fordulási ráták:
  - e (bal)=0
  - f (egyenes)=1
  - g (jobb)=0

- Követési arány:
  - KA=0,5

Ebben az esetben:

$$x = (1 - 0{,}5) * 1 + 0{,}5 * (1 + 1 + 1) * \frac{0}{0 + 1 + 0} = 0{,}5$$

$$y = (1 - 0{,}5) * 1 + 0{,}5 * (1 + 1 + 1) * \frac{1}{0 + 1 + 0} = 2$$

$$z = (1 - 0{,}5) * 1 + 0{,}5 * (1 + 1 + 1) * \frac{0}{0 + 1 + 0} = 0{,}5$$

Az eredményeken jól látható, hogy a bal és jobb irány eredeti fordulási rátái 1-ről 0,5-re módosultak, vagyis ténylegesen az 50%-uk vette figyelembe az ajánlást, az egyenes irány értéke pedig ezzel a 2*50%-kal megnövekedett.

Ezzel sikerült megvizsgálni, hogy a VJT-k milyen hatással lehetnek a forgalomra, továbbá a munkálatok során azt is sikerült megvalósítani, hogy a grafikus program együttműködve a táblák vezérlésére megírt programmal, az egyes ajánlások aktiválása során azonnal megjeleníti az adott ajánláshoz készített grafikus ábrát, tehát a táblákkal való kommunikáció megvalósítása is sikeres volt.

A VJT-k felhasználásának hatásaira már számos tanulmány született, Magyarországon azonban egyelőre nincsenek kihasználva ezek a lehetőségek. A VJT-k egyáltalán, vagy csak nagyon kis részben vannak összekapcsolva dinamikus rendszerekkel. Az általam elkészülő program lehetővé teszi, hogy a SwarcoFuturitVJT-ket ezek után bármilyen intelligens rendszerrel összekapcsolják. Ezzel lehetőség nyílik, hogy a gyakorlatban is megvalósíthassák azokat az alkalmazásokat, amelynek forgalomra gyakorolt pozitív hatásait, már több elméleti tanulmány is alátámasztotta.

12. Továbbfejlesztési lehetőségek

A kutatási eredményekből kiindulva mindenképp érdemes további figyelmet fordítani a rendszer tovább fejlesztésére.

12.1 Vezérlőprogram

Rövid távú tervek között szerepel a vezérlő program bővítése, mely lehetővé tenné több csomagból álló üzenetek küldését, illetve a tábláktól érkező válaszüzenetek fogadását és értelmezését, valamint a lehetséges parancsok integrálását a programba és onnan azok dinamikus kiválasztását, hogy mindig a megfelelő üzenet kerüljön kiküldésre.

Hosszú távú tervek között szerepel, hogy a program által megteremtett lehetőséget kihasználva olyan információkat hozzak létre, amelyek ténylegesen alkalmasak a forgalom befolyásolására. Ebbe beletartoznak a közlekedőknek szánt grafikus és szöveges információk, valamint olyan feladatok is, melyek labor körülmények között a szimuláció, valós felhasználás során pedig forgalomirányító berendezések működésének módosítására alkalmasak (pl sebesség korlátozás, forgalomfüggő fázistervek).

12.2 Vissim szimuláció

A későbbiekben szeretnék egy újabb szimulációt is elkészíteni, amely nem csak a program azon képességét vizsgálná, hogy alkalmas-e külső forrásokból érkező adatok kezelésére, hanem azt is, hogy a táblák és a program az előállított információkkal és adatokkal hogyan képesek befolyásolni a forgalmat.

12.3 COM program

A COM felületen megírt programban szükség lesz bővítő módosításokra:

- Több információ lekérdezése
- Visszahatás a szimulációra, a táblákra kiadott információk függvényében a szimuláció paramétereinek módosítása

12.4 Grafikus tesztprogram

A pusztán a vezérlő program kipróbálásra szánt úgynevezett GUI (GraficalUserInterface – Grafikus Felhasználói Felület) annyira jól sikerült,hogy alkalmas a Vissim COM felületen keresztül történő kezelésének azon funkcióját kihasználni, hogy a szimuláció futása közben is képes gyorsan módosítani akár nagyobb mennyiségű paramétert is, így ennek a munkának a lezárása után, egy új projekt keretében mindenképp érdemesnek találtam a továbbfejlesztésre. Ezzel a Vissim-mel

kapcsolatos kutatások, kísérletek során a kezünkbe kerülne egy eszköz, amely lényegesen megkönnyítené a szimulációs programmal való munkát.

13. Köszönetnyilvánítás

- Tettamanti Tamásnak az egész féléves belső konzulensi munkájáért.
- Brech Ferencnek (Swarco) a külső konzulensi szerep elvállalásáért.
- A Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedésautomatikai Tanszékének az általuk biztosított eszközökért.
- A PTV AG TrafficMobilityLogistic-nek a rendelkezésemre bocsátott Vissim szimulációs szoftverért.
- Ludvig Ádámnak a gyakorlati munkában nyújtott segítségéért.

14. Irodalomjegyzék

[1.] Dr. Tóth János - Közúti informatika jegyzet

[2.] KTI, Levegő Munkacsoport – A közúti és vasúti közlekedés társadalmi mérlege 2010.

[3.] MarkG. M.Brocken,Martie J.M.vanderVlist– TrafficControlwithVariableMessageSigns, 2006.

[4.] Tettamanti Tamás – Autópálya forgalomszabályozás felhajtókorlátozásésváltoztatható sebességkorlátozás összehangolásával és fejlesztési lehetőségei 2007.

[5.] SwarcoTraffic Hungaria honlapja: http://www.swarco.com/sthu

[6.] Állami Autópálya Kezelő Zrt honlapján megtalálható: Új berendezések, javuló baleseti statisztikák az ÁAK Zrt. autópályáin című cikke 2006., http://www.autopalya.hu/Root/Autopalya-HU/AAK_Cikk/2006/4/SK_20060421

[7.] Dr. Varga István, Tettamanti Tamás – Közúti irányító és kommunikációs rendszerek I-II jegyzet 2011.

[8.] PTV AG – VISSIM_530_e: UserManual segédlet a Vissim szimulációs szoftver használatához 2011.

[9.] PTV AG – VISSIM_COM Manual(segédlet a COM felület használatához) 2011.

[10.] Java-hoz segítséget nyújtó adatbázis: http://www.oracle.com/technetwork/java/javase/overview/index.html

[11.] Swarco – Futurit 2.0 protocoldocumentation: protokoll leírás 2008.

[12.] Swarco – Productdocumentation: leírás a táblákról és az SSC szoftverről 2011.

[13.] Socket kezelés: http://www.adp-gmbh.ch/win/misc/sockets.html