NEUMANN JÁNOS
INFORMATIKAI KAR

PROJEKTMUNKA

Hallgatók nevei: Németh Ernő Nándor, Rácz Roland, Tajti Kristóf Richárd, Szarvas Ádám János

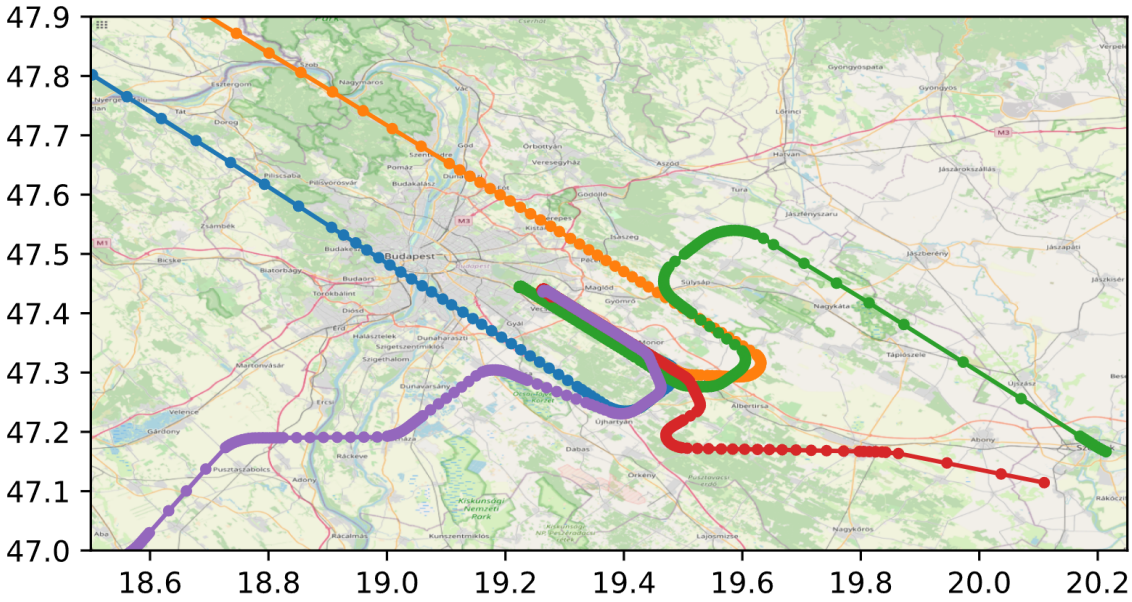
Csoport neve: MI 5

Mentor neve: Dr. Pitlik László

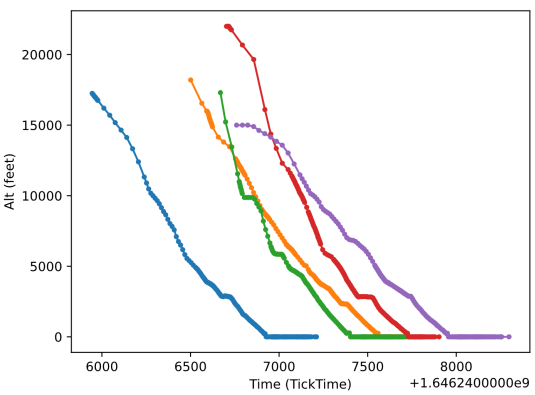
Tartalomjegyzék

1. Megoldandó probléma

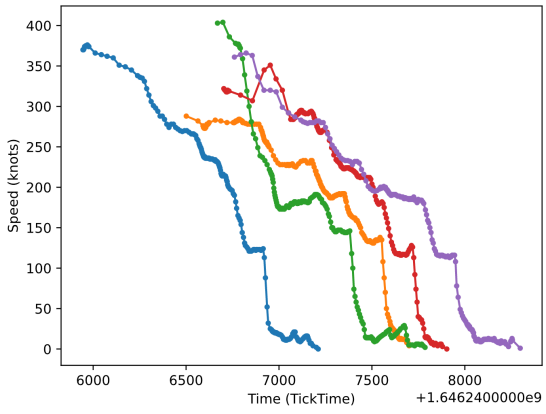
A probléma alapja az, hogy ha egy időben több repülőgép lép be a reptér előterébe, akkor a toronyban dolgozó légiirányító nem feltétlen a legjobb döntést és/vagy döntéseket választja az adott repülőgép csoportnak az irányítására. Mi erre a problémára szeretnénk egy komplex minőségbiztosítást (robot-repülésirányítót) kínálni. A célunk elérni, hogy a légiirányító értesüljön arról, ha a megszokottól valamilyen eltérés nem várt hibát eredményezhet, ill. egyáltalán egy adott konstelláció nagyobb (norma feletti) figyelmet igényel.



1-1. ábra. Járatok



1-2. ábra. Altitude/Time



1-3. ábra. Speed/Time

Az 1. ábrán egy nem optimális konstelláció látható. A pöttyök az adott hívójelű repülőgép útvonalát jelölik. A FR8394, CV6232 és a 6L5867 hívójelű gépek a leszállás előtt több kitérő manővert hajtottak végre, ami az útvonaluk és a repülési idő növekedését jelentette. A plusz manővereket az okozta, hogy voltak késések, illetve volt olyan repülő, ami a menetrendi érkezése előtt érkezett, ez eredményezte a torlódást. A probléma az optimalizálatlan landolásokkal a következők:

- befolyásolják más gépek indulását/érkezését,
- kellemetlenséget okoznak az utasoknak,
- a földi kiszolgáló személyzetnek hirtelen kell alkalmazkodnia, nem egyenletes a terhelésük,
- növelik környezeti terhelést (pl. az üzemanyag felhasználást,, zajterhelést),
- más gépeket kerülő útvonalak használatára kényszerítenek,
- biztonsági szempontból nem a legmegfelelőbbek...

2. Fókuszált funkció

Első körben arra szeretnénk fókuszálni, hogy a légiirányító lássa azt, ha eltér a normától, valamilyen vészhelyzetet okoz (például.: közel repülnek egymáshoz a repülőgépek, felesleges kerülőutakat végeztet el a pilótákkal). Azzal, hogy felügyeljük a légi irányító munkáját, minőség biztosítani tudjuk az irányítást. Ezen funkciók megvalósítása után szeretnénk a teljes légi irányítást is automatizálni, hogy a légiirányítónak csak felügyelnie keljen a repülőgépek mozgását.

3. Funkció katalógus

Az alábbi adatok kellenek ahhoz, hogy a fenti optimalizáltság komplex szinten érvényesüljön a fókuszált funkció támogatására, annak tudásanyagát többszörösen kiaknázva

- Repülőgépek pozíciója valós időben
- Időjárási adatok valós időben (például: szélirány, szél sebessége)
- Repülőgépek relatív helyzete (például: egymáshoz képest)
- Vizsgált légtér terheltsége

- Járatok validálása a koordinátáik (koordináták kilengése), magasság és sebességváltozás alapján
- Kvintettek összehasonlítása
- Mérni az irányító válaszidejét
- Percenkénti real-time előrejelzés
- Autopilot kényszerített bekapcsolása
- Zavaró repülők figyelembe vétele
- Jelzés a pilóta felé (legyen ébren, túl gyorsan jön, későn kezdte meg a landolást)

4. Program felépítése és működése

4.1. Szerver

A program egy Raspberry Pi 4-en fut. Ennek az eszköznek az előnyei közé tartozik, hogy olcsó a beszerzése és sok feladat ellátására is képes, mint például: lehet asztali és/vagy szerver számítógépként is használni. A Pi-n a Raspberry Pi OS fut, ez egy Linux disztibúció.

4.2. Programok és nyelvek

A kódot a feladat problémájából adódóan Python-ban írtuk, mert a beépített packageknek köszönhetően könnyen implementálhatók a függvények és könnyű összekapcsolni az adatbázissal. MySQL-t választottunk az adatbázishoz, mivel ingyenes a használata és elterjedt, illetve kompatibilis a Python-al. A kódok tárolására és verizókezelésére a Github-ot használjuk. A python kódot a PyCharm-ban, az adatbázist pedig dbForge-ban és phpMyAdmin-ban fejlesztjük.

4.3. Adatvagyonok és adatbázis

A projekt legelején csak a repülők helyzetét szerettük volna logolni. De ez nem elegendő, mivel a repülési útvonalak nagyban függnek az adott időjárástól, mint például: a futópályára csakis szembe széllel lehet fel-és leszállni, mivel a hátszél nem generál elég felhajtóerőt. Illetve a repülési útvonalak nagyban függnek attól, hogy a repülő mennyire tartja a menetrendjét. Ha például az összes járat akkor tudná megkezdeni a leszállást, mint ami a menetrendben ki van írva, akkor nem keletkezne torlódás a légtérben. A fentiek miatt három féle adatot logolunk:

- Időjárási adatok (METAR)
- Repülési adatok (OpenSky)
- Menetrendek (FlightRadar24)

4.3.1. METAR

A METAR adatok azaz, Meteorological Terminal Air Report, elengedhetetlenek lesz a jövőben a projektünk megfelelő felépítéséhez, hiszen a megfelelő repülési útvonalak megtervezéséhez és kivitelezéséhez is kritikus módon befolyásoló tényező az időjárás. Jelenleg meteorológiai adatokhoz még nem kötődik funkció a projekt első féléves állása szerint, de a jövőben a meteorológiai adatok alapján is képződnek majd OAM-ok.
A METAR adatok gyűjtésének első kérdése az volt mielőtt forrást kerestünk volna, hogy milyen sűrűséggel szeretnénk rögzíteni az adatokat az adatbázisunkban, majd egyértelmű okok miatt a fontos szempont volt a választásnál az is, hogy hány kontinensről tud adatokat szolgálni az adott oldal.

Több forrást is kipróbáltunk (pl.: FlightRadar24), többek között a repülési adatainkat szolgáltató FlightRadar-nak az időjárást rögzítő platformját is, de sajnos onnan a nem megfelelő jogosultságok miatt nem tudtuk automatizált módon megoldani lekérni az adatokat. Emellett még számos oldalt is kipróbáltunk, többek között az
https://aviationweather.gov/metar-t is ami inkább az Egyesült államok METAR adataira volt kihegyezve, de sok helyen az adatok nem megfelelő részletessége miatt zsákutcába futottunk.
Végül sikerült megtalálni a megfelelő forrást, ami a: www.getmetar.com
Ekkor nekikezdtünk egy olyan algoritmus megtervezésének, ami automatizált módon tölti majd le az adatokat nekünk. Ez oly mód működik, hogy amennyiben első meghívást

kezdünk, akkor létrehoz egy fejlécet, ami tartalmazza azokat az adatokat, amiket lekérünk a forrásunkról, ezek az alábbiak:

Adat	Mértékegység
Idő	Datetime
Felhő	Egyedi szöveges tájékoztató
Látási viszonyok	Méter
Szélirány	Fok
Szél sebessége	Csomó
Hőmérséklet	Celsius
Nyomás	mPa
Város és ország	Szöveges változó

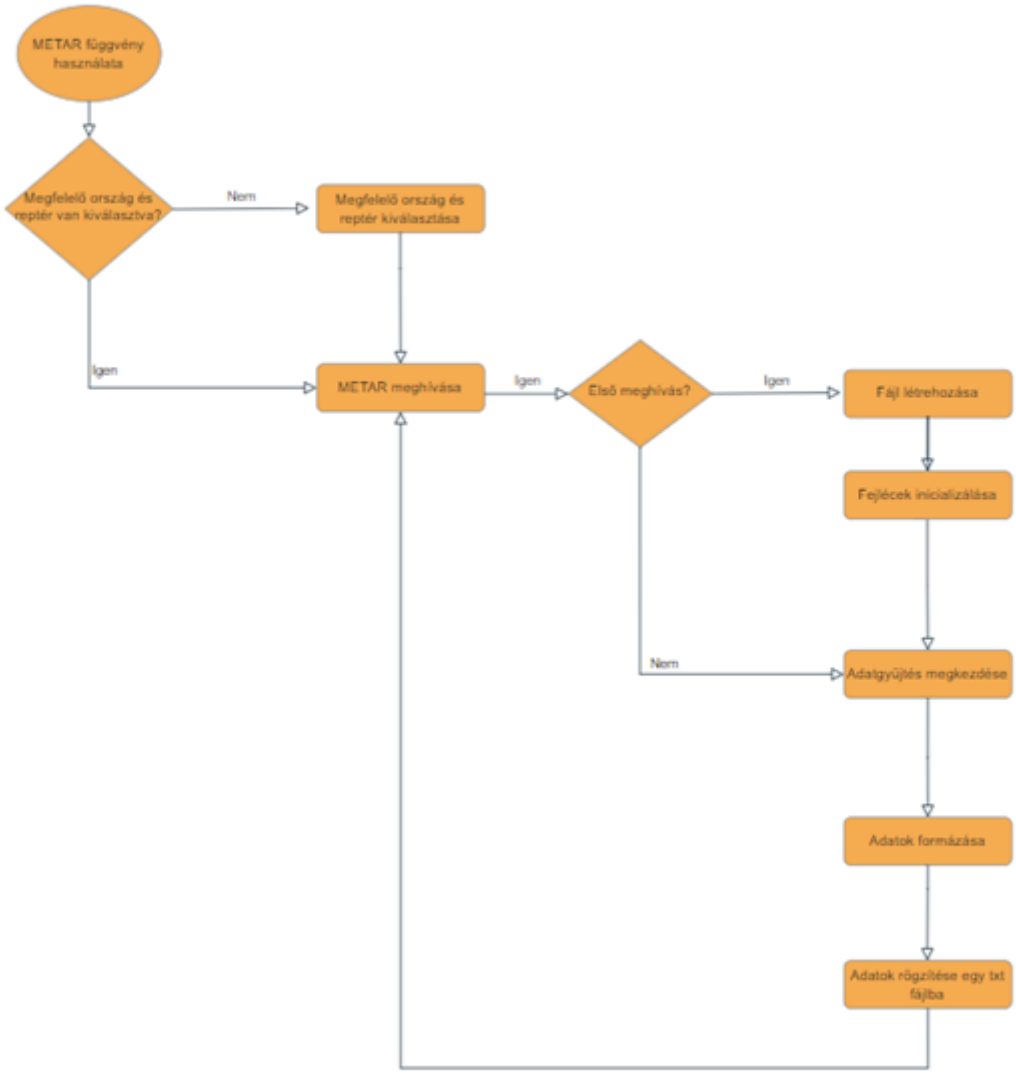
Majd ezeket az adatokat lekérjük a getmetar.com-ról és a szükség átalakításokat eszközölve fel is töltjük őket az adatbázisunkba.

4.3.2. Repülési adatok

A jelenleg meghatározott funkcionalitás, amit a projekt végén szeretnénk elérni, elsősorban a járatokhoz tartozó általános adatok (vö. mikor szállt fel, mikor fog leszállni menetrend szerint stb.), illetve az összehasonlítás alapját képző idősoros adatok beszerzését és tárolását vonta maga után. Az egyik legfontosabb kérdés tehát az volt, hogy milyen módon és milyen forrásból szerezzünk adatokat. Ezeknek a feltételeknek eleget téve, olyan szolgáltatások után kezdtünk el kutatni, ahol ezeket az adatokat könnyen el tudjuk érni és le tudjuk kérdezni. Viszonylag hamar szembesültünk azzal a problémával, hogy a legtöbb megbízható adathalmazt kínáló szolgáltató (FlightRadar24) nem ingyenes. Illetve külön üzleti tervet is kértek, és az árak sem voltak számunkra megfizethetők. Kitartó kutató munkánk gyümölcseként viszont sikerült 2 forrást találnunk, amelyek megbízhatók is és minden olyan adat, ami jelenleg számunkra kellhet ahhoz, hogy elkezdjük a munkát, elérhető rajtuk keresztül.

4.3.3. OpenSky-Network

Szerencsénk volt, hogy rátaláltunk erre az oldalra, mivel teljesen ingyen kínálja olyan adatok elérését, amelyek számunkra szükségesek, például a járatok gyorsulása, föld feletti távolsága. Nincs lekorlátozva, hogy mikor és mennyit tudunk lekérdezni. Továbbá



2.Ábra: Ábra: METAR lekérdezésének folyamatábrája. (Saját ábra)

részletes dokumentációt is találunk az API használatáról, és az adatok szerkezetéről, ami nagyban elősegíti a munkánkat. Habár vannak hiányosságai például a tervezett/tényleges felszállását és a tervezett/tényleges leszállását nem tudjuk elérni, ezeken kívül az adat igényünket nagyban lefedi.

4.3.4. FlightRadar24

A hiányzó adatokat, amit az OpenSky-Network-ről nem tudunk elérni például a járatok helyének koordinátáit idősorosan, erről az oldalról szerezzük be. Attól függetlenül, hogy sok ingyenes szolgáltatás közül tudunk válogatni például élő járat követés, nekünk mégis

egy fizetős csomagra esett a választásunk. Ennek oka egyszerűen annyi, hogy az ingyenes lekérdezések le vannak korlátozva, és egy nap csak egy bizonyos mennyiségű adatot tudunk elérni. Ellenben mi egész nap, korlátlan darabszámban akarunk lekérni információt.

4.3.5. Megvalósítás

A megfelelő források felkeresése után maga a kód megvalósítása következett. Az oldalakról API kérések segítségével nyerjük ki az adatokat. Habár ez elég egyszerű és gyors megoldást nyújt, sok olyan információt is kapunk, amelyek a jelenleg kitűzött célunkhoz, nem biztos, hogy relevánsak. Ilyenek például az adott repülőtér országához tartozó időzóna.
A későbbiekben viszont érdemes lehet megnézni, hogy amit számunkra fontos adatnak tituláltunk, ott a „robot" is ugyan erre a következtetésre jut-e, hogy tényleg ezek azok az adatok, amik ehhez a funkcióhoz elengedhetetlenek, vagy fel tudott fedezni olyan kapcsolatokat, és szabályszerűségeket, amikre mi emberi szemmel nem voltunk képesek (vö. funkció-katalógus). Ebben a félévben, elsősorban a célkitűzésünk maga az adatvagyon megszerzése és legalább egy alap-funkcionalitás megvalósítása volt.

Az alap-funkcionalitás nem más, mint az MI célirányosságának, minőségének, emberszerűségének garanciáját jelentő JÓSÁG-fogalom modellezni tudása volt. A döntéstámogatás alapja a NEM-JÓ (nem ideális) állapotok nem deklaratív, hanem adaptív (vö. GPS) kezelni tudása, vagyis egy olyan gondolatiság megalapozása, ahol LEGO-jelleggel lehet OAM-vezérelten egyetlen egy matematikai apparátust (vagyis a hasonlóság központba állított fogalmát) univerzálisan használni.

A következő félévekben viszont egyre növekvő komplexitásoknak (paramétereknek, attribútumoknak, objektumoknak, OAM-oknak az optimális értékei levezetésének a „robotra" bízása is felkerül a palettára - vö. funkció-katalógus).

Az adatok begyűjtését a FlightRadar24-ről kezdjük. Itt kapjuk meg az esetünkben a Budapest Liszt Ferenc nemzetközi repülőtérre (IATA: LHBP) érkező járatok adatait. Az API-től kapott válaszból ki kell szűrnünk a számunkra fontos adatokat.

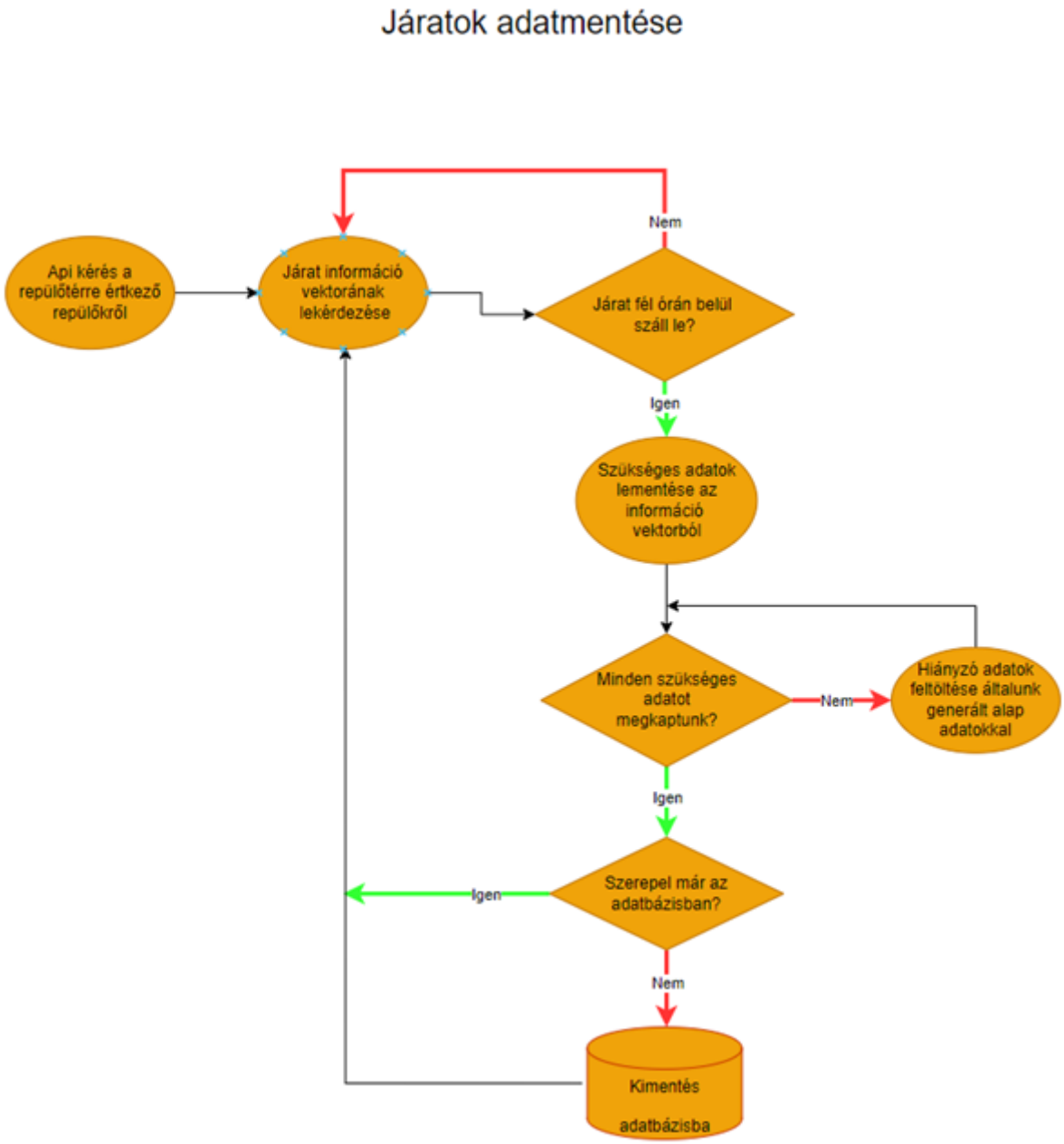
Még mielőtt elmentenénk minden adatot, meg kell vizsgálnunk, hogy a járat leszállása fél órán belülre van-e tervezve, azért fél órával dolgozunk, mert az adatok logolását végző script fél óránként indul újra és így a legegyszerűbb és leginkább erőforrás kímélő an-

nak az elkerülése, hogy az egyes járatokat többször is felvigyük az adatbázisba. Ezzel a módszerrel csak pár járatot kell felülvizsgálnunk, amelyek nagyon nagy késésben vannak. Ez a funkció erre a repülőtérre még annyira nem effektív, viszont úgy írtuk meg a kódot, hogy akár nagyobb és forgalmasabb repülőtereket is megfigyelhessünk vele, ahol viszont sokat nyerhetünk vele például a redundancia elleni küzdelemben.

Miután kiszűrtük a fél órán belül leszálló járatokat, a szükséges adatokat ki kell gyűjtenünk és megvizsgálni, hogy minden szükséges adatot megkaptunk-e és nincs-e hiányzó információ. Ha valami hiányzik, jelenleg a mi feladatunk, hogy ezeket pótoljuk valamilyen odaillő adattal például a meglévő adatok átlagát helyettesítjük be. Ugyancsak érdekes lehet ezeket az adatokat később a „robotra" bízni, mint ahogy már az előbb is említettem (vö. Funkció-katalógus - inkl. Érzékenységvizsgálat minden adatpótlási esetben).

Ezek után még egy utolsó ellenőrzés következik, ahol a már az adatbázisban elmentett járatokkal összehasonlítjuk és megvizsgáljuk, hogy van-e ugyanolyan elem. Itt fontos megjegyezni, hogy olyan adatok alapján hasonlítjuk össze a rekordokat, amik nem hiányozhatnak az adatok lekérése során, és nem mi állítjuk elő ezeket az információkat. Ilyen például a callsign és a dátum, amelyek minden esetben rendelkezésünkre állnak.

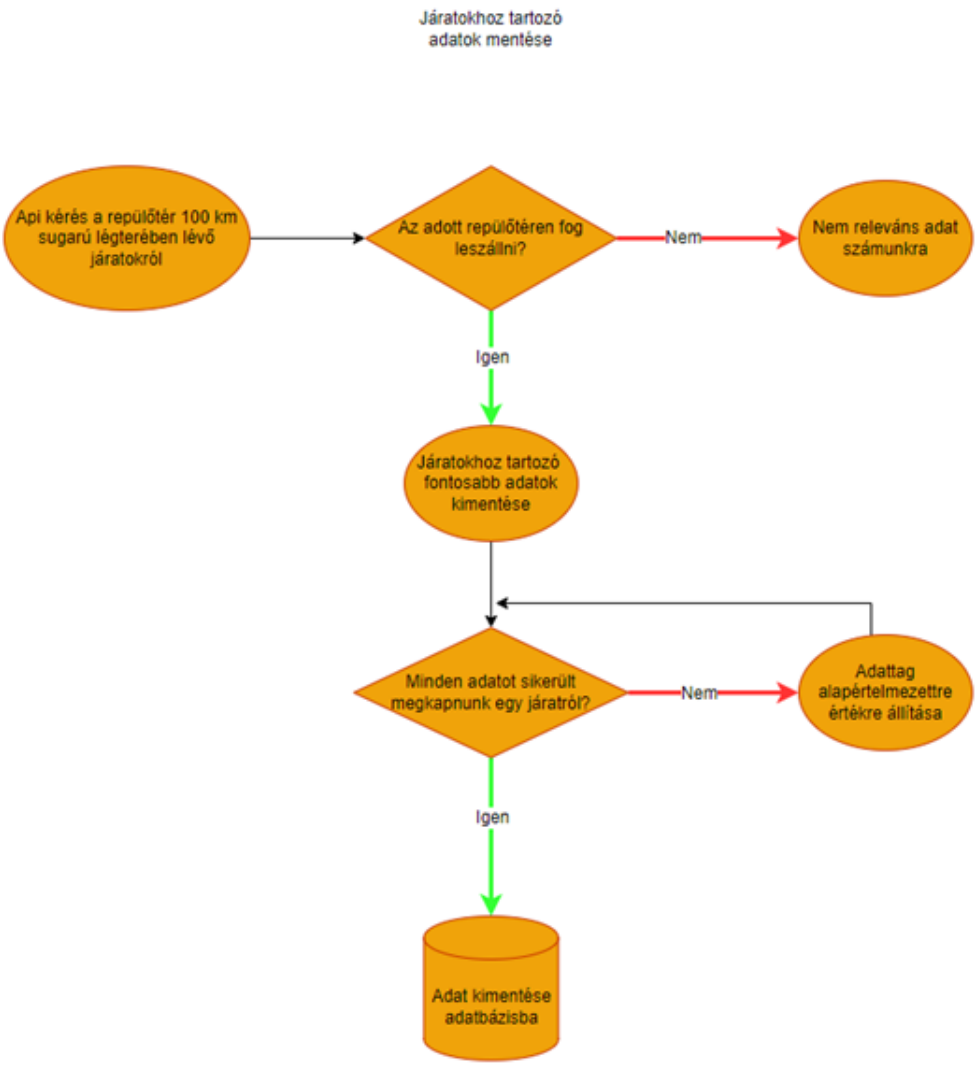
Az utolsó ellenőrzés után a script elküldi az adatbázis felé a lementendő adatokat, és újból kezdi a járatok szűrését (vö. 3. ábra).



3.Ábra: Járatok ellenőrzése és mentése adatbázisba. (Saját ábra, Jelmagyarázat: Piros vonal: Elágazásban lévő feltétel nem teljesült, Zöld vonal: Elágazásban szereplő feltétel teljesült, Fekete vonal: Feltétel nélkül teljesülő lépések)

4.3.6. Járatok

Következő lépésben a járatokhoz tartozó idősoros adatokat kérjük le az OpenSky-Network-ről. Itt minden másodpercben megkapjuk a reptér 100 km sugarú légterébe belépő repülőket. Fontos, hogy az API kéréssel azokról a járatokról is kapunk információt, amelyek nem ezen a reptéren szállnak le. Így az első szűrési pontunk az az, hogy azokról a járatokról szeretnénk az adatokat megkapni, amelyek az általunk kiválasztott repülőtéren fognak leszállni. A jelenlegi funkcióhoz, amit kitűztünk célul csak ezekre a repülőkgépekre van szükségünk.
Ezt úgy valósítjuk meg, hogy a már adatbázisba elmentett, fél órán belül leszálló járatoknál megkeressük, hogy van-e ugyanolyan callsign-al rendelkező repülőgép, mint ami most lépett be a légtérbe. Ha van egyezőség akkor az azt jelenti, hogy az a repülőgép itt (Budapesten) fog leszállni. Ezek után, ha megállapítottuk, hogy a járat ezen a repülőtéren fog leszállni, a további műveletek szinte megegyeznek a nem idősoros adatok kezelésével, azzal a kivétellel, hogy adatbázisba mentés előtt már nem ellenőrizzük le azt, hogy szerepele már ott bármelyik is (4. ábra)



4.Ábra: repülőgépek pillanatnyi adatainak lekérdezése és eltárolása az adatbázisban. (Saját ábra, Jelmagyarázat: Piros vonal: Elágazásban lévő feltétel nem teljesült, Zöld vonal: Elágazásban szereplő feltétel teljesült, Fekete vonal: Feltétel nélkül teljesülő lépések)

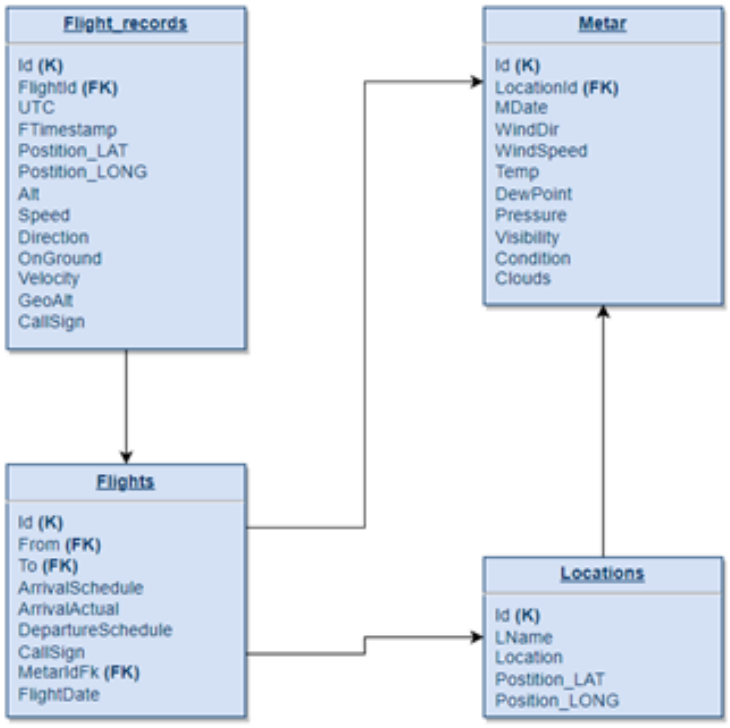
4.3.7. Adatbázis

Az folyamatos adatmentés és az új adatvagyon feldolgozása miatt szükségünk volt egy adatbázisra, amivel CSV-ben való adattárolás nem volt eléggé rugalmas számunkra. Az adatbázist MySQL nyelven írjuk, mivel széles körben elterjedt nyelv. Az adatbázis tervezésekor kiemelt figyelmet fordítunk arra, hogy az adatvagyont csak az adatbázison keresztül, függvények és metódusokon keresztül lehessen csak elérni, módosítani és törölni. A programból ne kelljen sima SQL lekérdezéseket írni, mivel ezek rontják az adatbiztonságot (például.: Drop database *), a kódot nehezebb debug-olni és a hibakeresés is

nehézzé válik. A függvényeknek és a metódusoknak még olyan előnyük is van, hogy elő tudják szűrni az adatokat a program számára, illetve nem kell egy lekérdezést többször megírni, s nem mellesleg gyorsabbak is minden más alternatív megoldáshoz képest. Felépítése:

Az adatbázis négy darab táblából áll: Flight records, Flights, Metar és Locations.

- Flight records: Ebben a táblában tároljuk el a repülések során keletkezett adatokat, például: 2022.04.26-20:50:32-kor az ABC123 hívójelű gép X koordinátában van, Y km/h a sebessége és Z méter a magassága. Ezt a táblát használjuk a leggyakrabban, mivel másodpercenként kérjük le a repülőgépek pozícióját.
- Flights: Ebben tároljuk el az érkező járatokat.
- Metar: Egy adott helyszínen (pl. Budapest) uralkodó időjárást tároljuk el benne. Ezeket az adatokat a pilóták és a légiforgalom irányítók használják, mivel például az uralkodó széliránnyal szemben kell a leszállást végrehajtaniuk a pilótáknak.
- Locations: Ebben a táblában vannak a repterek eltárolva. Ezekre mind szükségünk van, hogy azonosítani tudjuk, hogy melyik METAR record hova tartozik, illetve az adott járat honnan hová megy.



5.Ábra: Adatbázis felépítése. (Saját ábra)

Függvény	**Bemeneti Paraméterek**	**Leírás**
InsertLocation	LocName varchar(255), IDate datetime, WindDir int(3), WindSpeed int(3),Temp int(3), DevPoint int(3), Pressure int(4), Visibility varchar(255), Condition varchar(255), Clouds int(5)	A megadott paraméterekkel beszúrja az adott helyiséget.
InsertFlight	From varchar(255), To varchar(255), ArrivalSchedule datetime, ArrivalActual datetime, DepartureSchedule datetime, DepartureActual datetime, CallSign varchar(255), FlightDate datetime	Az adott paraméterekkel beszúrja a járatot. Ha a location táblában még nem szerepel a "_From" és/vagy a "_To", akkor ezt beszúrja a locations-be az új helységet. Illetve a járathoz párosít egy METAR-t.
InsertFlightRecord	CallSign varchar(10), Timestamp int(11), PositionLONG float(7,4), PositionLAT float(7,4), Alt float, Speed float, Direction float, OnGround tinyint(1), Velocity float, GeoAlt float	Beszúrja az adott repülőnek a pillanatnyi adatait és ha létezik hozzá járat a flight táblában, akkor ez hozzá párosítja a rekordhoz.
GetFlights BetweenTwo UnixDate	startDate int,endDate int	Visszaadja az összes repülési adatot a flight_records táblából, ami két UNIX idő között van.
GetFlightRecords Where AltNotZero	callsign varchar(255), startDate int, endDate int	Visszadja annak a gépnek a repülési adatait a flight_records táblából, aminek a hívójele megegyezik a paraméterben megadott_callsign-nal és a két date között van.

GetAllFlight RecordsBetweenTwoDatetime WhereAltNotZero	startDate int,endDate int	Visszaadja az összes adatot a flight_records táblából, ahol a magasság nem egyenlő nullával és a két idő között van a record.
GetCallSign Between	startDate int,endDate int	Visszaadja az összes hívójelet két idő
GetXtet	Ntet int, startDate int, endDate int, timeRange int	Visszaadja az összes repülési adatot két idő között, ha a két idő között volt N darab repülő a levegőben. Különben nullal tér vissza.
GetAndInsert Location	LocName varchar(255)	Visszatér egy location IDval. Ha nem létezik, akkor beszúrja és annak az ID-jával tér vissza.
NumberOf FlightsBetween TwoDates	startDate int,endDate int	Visszaadja a repülők számát két idő között.

5. Függvények

Az OAM-ok (-attribútum mátrix) megvalósításánál nemcsak a repülők (ok) log-adatai játszanak fontos szerepet, hanem az azokból kiszámított/származtatott attribútumok értékei is. Első lépésként beolvassuk a megfelelő repülőjáratok bizonyos log-adatait (vö. Következő lista). Megfelelő repülőjáratokon azt értjük, amikor 5 egymást követő gép Budapesten landol fél órán belül. Az adataik pedig a LHBP 100 km-es körzetében logolt adatok, (ahol a projekt jövője szempontjából az 5 db, a 30 perc, a 100 km és Budapest is változtatható paraméterként kerül értelmezésre a későbbiekben), melyek magukba foglalják a következőket:

Time (másodpercek 1970. 01. 01. óta)	másodperc
UTC	YYYY-MM-DD'T'HH:MM:SS'Z'
Callsign (azonosító)	alfanumerikus azonosító
Latitude (földrajzi szélesség)	fok
Longitude (földrajzi hosszúság)	fok
Altitude (magasság)	méter
Speed (sebesség)	méter/szekundum
Direction (irány)	fok

5-1. táblázat. Paraméterek

5.1. get_timeStamp_between_first_and_last()

Egyrészt fontosnak találtuk megadni az első és utolsó gép landolásai közti időkülönbséget. Ez bármiféle bonyodalom nélkül, egyszerűen kiszámítható. Csupán annyira kellett figyelni, hogy az utolsó időpillanat ne egy bármilyen adat legyen, hanem az az első adat, amikor a repülő éppen landolt, azaz az Altitude értéke elsőként 0 láb. Az érték pontosságát akár fejszámolással is tudjuk ellenőrizni. Minél kisebb ezen időintervallum [get_timeStamp_between_first_and_last()], annál nagyobb annak kockázata, hogy a kvintett kapcsán nem kívánatos események lépnek fel.

5.2. get_biggest_difference_between_timestamp()

Egy másik fontos függvénynek tekintettük a várható landolástól számított legnagyobb eltérést. Ennek a kiszámításához első lépésben landolási idő szerint kellett rendezni a Flight records táblákat, amiben a repülők adatai tárolódnak (vö. Adatbázis felépítés), majd venni szintén azt az első értéket, ahol a repülő magassága 0 méter (= már éppen földet ért). Ezt követően ki kellett számolni, hogy mi a várt, ideális eltérés. Ehhez a korábban már kiszámított, első-utolsó járat landolási különbségére, valamint a repülők számánál eggyel kevesebbre van szükségünk, ami esetünkben 5-1 = 4. E két érték hányadosa megadja, hogy mennyi a két gép-landolás között eltelt ideális időintervallum. Így, hogy már tudjuk ezt az értéket, egy egyszerű képlet segítségével kiszámíthatjuk a várttól való eltéréseket:

Cikluson belül:

$$diffsFromExpected[i] = landolási\_időpillanat[i] - ideális\_időintervallum + első\_gép\_landolási\_időpillanata$$

Ezt követően az adott adatszerkezeten (diffsFromExpected) végig iterálva megnézzük, hogy melyiknek a legnagyobb az abszolútértéke. Az abszolútérték abból a szempontból fontos, hogy lehet, hogy a gép a vártnál korábban érkezik, de lehet, hogy késik. Amint megkaptuk az abszolút értékek közül a legnagyobbat, visszaadjuk azt, mint legkiugróbb értéket. Az érték ellenőrzéséhez matematikai műveletek pontos elvégzésére van szükség. Minél nagyobb a get_biggest_difference_between_timestamp() értéke, annál nagyobb annak kockázata, hogy a kvintett kapcsán nem kívánatos események lépnek fel, mivel annál jobban eltér az abszolút ideális landolási időponttól.

5.3. get_biggest_distance()

Ez a függvény azt írja le, hogy a mért környezetbe (vö. 100 km-es sugarú körbe/félgömbbe) való beérkezéstől a végső (landolási) irány beálltáig a repülőgép mekkora legrövidebb távolságot tehetett volna meg. Tehát a kezdeti koordinátákat (Latitude, Longitude), valamint a vég koordinátákat kigyűjtöttük külön változókba. A végső irányt úgy határoztuk meg, hogy a landolás előtti utolsó 30 iránynak vettük a mediánját. A távolságok kiszámításához egy beépített függvényt hívtunk segítségül, melynek pontosságát különböző internetes kalkulátorok igazolják. Minél kisebb a get_biggest_distance() értéke, annál optimálisabbak a repülők levegőben megtett útjai.

5.4. get_biggest_average_declaration()

További fontos elemnek találtuk meghatározni a repülők lassulását a zónába lépéstől egészen a végső irány beálltáig (melyet az előző fejezetben részleteztünk). A gépek különböző lassulási értékeit könnyedén kiszámíthatjuk a következő képlet segítségével: Cikluson belül:

$$\mathbf{Declaration[i] = (v2–v1)/(t2–t1)}$$

Ez egy általánosan elfogadott és bizonyított képlet, így helyességét papíron és kalkulátorral is lehet bizonyítani. Minél kisebb a get_biggest_average_declaration() értéke, annál biztonságosabban kivitelezhető a leszállás, ellenben minél nagyobb, a repülők annál nagyobb kockázatnak vannak kitéve mind biztonsági, mind landolási szempontból.

5.5. get_minimum_minimalize_time()

Ez a függvény egy korábban kiszámított értékből, a lassulásból, valamint a kezdeti- és végsebességből számítja ki, hogy mi az a minimális idő, ami alatt a zóna-landolási irány távot meg lehetett volna tenni. Az előzőhöz hasonlóan itt is egy általános, ám sokak által bizonyított képletről van szó, így könnyen ellenőrizhető: Cikluson belül:

$$Idő[i] = (végsebesség[i] - gép\_landolási\_időpillanata[i])/lassulás[i]$$

Minél kisebb a get_minimum_minimalize_time() értéke, annál jobban tudják tartani a repülők a menetrendszeri érkezést.

```
Az első és utolsó gép landolásai közti időkülönbség:
1087
-----------------------------------------------------------------------------------------------------------
A várható landolástól számított legnagyobb eltérés:
217.25
-----------------------------------------------------------------------------------------------------------
A mért környezetbe való beérkezéstől a végső (landolási) irány beálltáig a legrövidebb távolság:
82279.2
-----------------------------------------------------------------------------------------------------------
A nagyobb lassulás (negatív gyorsulás) értéke:
-0.36
-----------------------------------------------------------------------------------------------------------
A legkisebb idő alatt lehet megtenni a távot:
1066.67
```

6. Ábra: Kvintettek adatai (Program egyik kimenete)

További attribútumok a mindenkori emberi szakértők által definiálhatók jelenleg. A robot általi önálló attribútum-definiálás, mint funkcionalitás egyelőre meghaladja a munkacsoport által kitűzött reális célok szintjét.

6. Objektum Attribútum Mátrix

Oszlopok:

- 1. kvintettek azonosítói (0-10)
- 2-6. Az objektumok attribútumai (vö. Függvények fejezet)
- 7. A használt függvény jellegéből adódó input értékek

Az utolsó három sor az oszlopok értelmezésében játszik szerepet:

- unit: Az adatok mértékegységét adja meg.
- direction: Az objektumok értékeinek a monotonitásához párosítható rizikófaktort kifejező bináris számok (vö. Függvények fejezet).
- remark: A direction bináris értékeinek szöveges értelmezése.

A megkapott nyers adatokból mátrixot képzünk ez lesz az OAM (lásd 7. ábra). Sokat könnyíthetünk az olvashatóságán és egyúttal a tudásunkat is jobban reprezentálhatjuk, ha ezekből a feldolgozatlan értékekből a direction sort figyelembe véve egy rangsor táblázatot gyártunk (lásd 8.ábra).
Ezt a táblát már egyrész optimalizálatlanul (azaz naivan pl.Hőtérképként is) jobban átlátjuk, másrészt pedig a függvénynek/online elemző motornak (vö. COCO) is át tudjuk adni.

kvintett_id	get_timeStamp_between_first_and_last	get_biggest_difference_between_timestamp	get_biggest_distance	get_biggest_average_deceleration	get_minimum_minimalize_time	Y0
0	1532	509	132763	-230	37	1000
1	1033	237	115055	-260	292	1000
2	1747	351	88486	-180	35	1000
3	1218	226	100079	-190	70	1000
4	918	293	134435	-180	244	1000
5	614	408	94139	-220	392	1000
6	1496	719	133744	-170	247	1000
7	1480	966	74139	-190	155	1000
8	1220	405	116290	-170	149	1000
9	1712	1141	138073	-180	85	1000
10	1262	422	79741	-200	22	1000
unit	sec	sec	meter	mm/s2	sec	score
direction	1	0	0	1	0	
remark	the less the more risky	the more the more risky	the more the more risky	the less the more risky	the more the more risky	

7. Ábra: 11 db objektumunk (kvintetthez) tartozó nyers adatot

							the more the more risky!	
kvintett_id	get_timeStamp_between_first_and_last	get_biggest_difference_between_timestamp	get_biggest_distance	get_biggest_average_deceleration	get_minimum_minimalize_time	Y0	modell	Validitás
0	9	4	4	2	9	1000	1006.7	1
1	3	10	6	1	2	1000	1005.2	1
2	11	8	9	7	10	1000	981.3	1
3	4	11	7	5	8	1000	992.2	1
4	2	9	2	7	4	1000	1004.2	1
5	1	6	8	3	1	1000	1014.7	1
6	8	3	3	10	3	1000	998.7	0
7	7	2	11	5	5	1000	998.7	1
8	5	7	5	10	6	1000	994.2	1
9	10	1	1	7	7	1000	1008.2	1
10	6	5	10	4	11	1000	995.7	1

8. Ábra: 11 db objektumunk (kvintetthez) tartozó rangsorolt adatot és a függvény eredményét

Azt szerettük volna bebizonyítani ebben a félévben, hogy a kvintettek világában is léteztethető a jóság fogalma. Azaz meg lehet-e mondani az adott szituációban, hogy a kapott adatok alapján vannak rizikósabb és kevésbé rizikós helyzetek a légiirányítók/ repülőjáratok számára. Ezeknek az állapotoknak a meghatározásában szeretnénk majd real-time jelleggel segítséget nyújtani nekik, viszont ez még a következő félévek tervei között van, hiszen ehhez a jelenlegi attribútumképzéshez használt függvényeket leszállás-függetlenné kell tenni.

Most kizárólag már landolt járatok adataival dolgoztunk. A függvény eredményeit a 8.ábrán, az utolsó előtti oszlopban láthatjuk. A függvényünk lényege, hogy az egyes objektumokhoz tartozó értékeket az Y0 oszlopban található számhoz (norma) próbálja közelíteni, esetünkben 1000-hez.

Láthatjuk, hogy ez az esetek többségében nem jön össze, még akkor sem ha egy +-1-es kerekítési hiba határt megengedünk. Ebből arra a következtetésre jutunk, hogy a kvintetteink, különböznek egymástól, azaz létezik a jóság fogalma/skálája és vannak rizikósabb és kevésbé rizikós helyzetek. Minél nagyobb egy becsült kockázat-index annál veszélyesebb a szituáció. Ezt a cellák színezésével is érzékeltetjük (vö. minél pirosabb, annál kockázatosabb/veszélyesebb). Képzeljük el az eredményekhez tartozó színeket úgy,

hogy az irányítótoronyban mindig olyan színnel világít egy lámpa amilyen kockázatos egy kvintett irányítása (piros-kockázatos, sárga-normaszerű, zöld- rel. kis mértékű kockázat). Ez jelzi az irányítóknak, hogy nagyobb odafigyelést igényel-e egy adott szituáció (itt nyilván alapul véve azt, hogy minden helyzet úm. “nagy” odafigyelést kíván meg). A validitás oszlopot az eredeti irányokkal kialakított és az eredeti rangsoros mátrix inverzéből kapott eredmények összehasonlításából kapjuk. Ha az inverz értékek becslési hibái előjelesen az ellentettjei az eredeti outputok becslési hibáinak akkor valid az eredmény (1) ellenkező esetben nem valid (0). A mi helyzetünkben ez annyit jelent, hogy a toronyban fel van-e kapcsolva a lámpa vagy sem (vö. szürkeség) -természetesen, ha fel van kapcsolva akkor a megfelelő színnel világít.

7. ITB-aspektusok

Az input adatok validitására nagy hangsúlyt kell fektetnünk és folyamatosan tanulmányoznunk kell azok helyességét. A robot csak azokból az adatokból képes tudást és értékrendet felépíteni amit mi szolgáltatunk számára. Ha manipulált adatokkal látjuk el akkor ez a tudás és értékrend torz lesz és pont az ellentetjévé fog mutálódni, mint amire mi szeretnénk használni, mondhatni használhatatlanná válik számunkra. Például a beérkező GPS adatokat úgy tudjuk validálni, hogy az időrendben egymás után lévő GPS koordinátákat összehasonlítjuk és megnézzük az eltéréseket. Ha nagy a koordináták közötti eltérés, akkor egy flag-el megjelöljük a járatot. A mi célunk az, hogy előre tudjunk jelezni a kritikus helyzetekben a légi irányítók számára, és nem az, hogy ezeket az eseményeket elszalasszuk. Invalid adatokkal pedig ennek a kockázata lényegesen megnő. Fontos megérteni, hogy ilyen helyzetekben nem vonhatjuk meg a vállunkat és bújhatunk ki a felelősségvállalás alól, hanem igen is vállalnunk kell a következményeit annak a "robotnak" a tetteiért amit mi alkottunk. Ezért kell az adatvagyon tisztaságát, helyességét és biztonságát fenntartani.

8. Önértékelés

https://miau.my-x.hu/miau/284/mi_munkacsoportok_projektek_ertekelese.xlsx

8.1. Piacképesség/Startup-potenciál

Amennyiben a probléma végére érünk, már nem csak a LHBP-re tudjuk alkalmazni a javaslatokat, hanem apróbb változtatások után bármely repülőtérre képesek leszünk optimalizálni a repülőjáratok útját és landolását. Ezt figyelembe véve a program abszolút piacképesnek tekinthető, hiszen nagyon sokat tudna segíteni a légiforgalom irányító szakembereknek a munkájukban, olyan esetekben, mikor egyidejűleg nagyobb mennyiségű repülő érkezik egy reptérre leszállási szándékkal, melyre a repülés folyamatos térhódításával egyre sűrűbben előforduló problémaként kell tekintenünk. Jelenlegi funkcióként implementálható egy olyasfajta visszajelzés a légiirányítóknak, melyek azzonali képet adnak az éppeni teljesítményről. Ezt színekkel érhetjük el legegyszerűbb módon, miszerint is optimális útválasztásnál egy zöld, optimálistól eltérő esetekben sárga és végzetes hibáknál pedig piros fénnyel jelzünk vissza.

8.2. Real-time jelleg

Alapvető ötletünk megvalósításához szükség van, arra, hogy a program valós időben tudja lekezelni a beérkező gépeket és így tudja segíteni és optimalizálni a légiforgalom irányítók munkáját, azonban a program tanulási és képzési célokra is felhasználható lenne, így már megtörtént eseményeket vissza lehet majd vele játszani és ellenőrizni azt, hogy egy adott szituáció mennyire lett jól lekezelve a légiforgalom irányítók által, illetve teljesen kezdők oktatására is használható lenne a szoftver. A piacképességnél említett színvisszajelzés abszolút tekinthető valós idejű szabályozásra alkalmasnak, hiszen irányítás közben tud színvisszajelzéssel szolgálni.

8.3. Jövőkép

A következő félévekben teljesen működőképes állapotba szeretnénk hozni a projektünket, hogy valós körülmények között is megállja a helyét. Ennek eléréséhez folyamatosan próbálunk az iparban dolgozó szakemberektől tanácsokat kérni, együttműködni velük és olyan irányba vinni tovább a programunkat, hogy minél inkább valós problémákat tudjunk vele megoldani. Miután sikerült megvalósítanunk az alapötletünket számos új, segítő funkció bevezetésével tudjuk fejleszteni a programot. Például:

1. Repülők teljes távoli vezérlése:...
2. Autopilot kényszerített bekapcsolása: klasszifikáció probléma, melyre alkalmas a jelenleg is alkalmazott tudásreprezentációs megoldás adaptálva
3. Jelzés a pilóta felé (gyorsan megy, későn kezdte meg a landolást, stb.): tény-becslés összehasonlítás, melyre alkalmas a jelenleg is alkalmazott tudásreprezentációs megoldás adaptálva
4. Légiutaskísérőknek engedélyt adni bizonyos veszély jelzésére az irányítótorony felé: klasszifikáció probléma, melyre alkalmas a jelenleg is alkalmazott tudásreprezentációs megoldás adaptálva
5. Repülő lassítása: tény-becslés összehasonlítás, melyre alkalmas a jelenleg is alkalmazott tudásreprezentációs megoldás adaptálva
6. Leszállás szimulálása:...
7. Zavaró repülők figyelembe vétele: ...

9. Jövőben kifejtendő témakörök

- MI aspektusok
- GPS aspektusok
- Alternatív megoldások
- Turing tesztek
- Benchmark-ok
- Konzisztencia-alakzatok / minőségbiztosítás
- Függvény Szimmetria-alapú validitás

10. Mellékletek

METAR - https://library.wmo.int/doc_num.php?explnum_id=10474
METAR dekódólása - https://metar-taf.com/explanation
METAR logok - www.getmetar.com
Repülési adatok - https://opensky-network.org/
OpenSky-Network API - https://openskynetwork.github.io/opensky-api/
OpenSky-Network Dokumentáció - https://opensky-network.org/data/impala
FlightRadar24 API - https://pypi.org/project/FlightRadarAPI/
Gyorsulási képlet -
https://hu.cathedralcollege.org/calcular-la-aceleracin-4676
Gyorsulás bizonyítása -
https://www.physicsforums.com/threads/constant-acceleration-proof-dv-dt-a.396383/
Gyorsulás ellenőrzése -
https://www.omnicalculator.com/physics/acceleration
Mértékegységek - https://opensky-network.org/data/impala
Koordináták közötti - https://pypi.org/project/haversine/
Mentor álltal nyújtott segédletek - https://miau.my-x.hu/myx-free/