Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Megbízható kommunikációs kapcsolattal rendelkező földi irányító állomás fejlesztése UAV-hoz

Szakdolgozat

Készítette
Böjti Paszkál

Konzulensek
Vörös András, Dr. Bartha Tamás

2013. november 23.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Böjti Paszkál*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (Böjti Paszkál, Megbízható kommunikációs kapcsolattal rendelkező földi irányító állomás fejlesztése UAV-hoz, angol és magyar nyelvű tartalmi kivonat, 2013, Vörös András, Dr. Bartha Tamás) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2013. november 23.

_____

Böjti Paszkál
hallgató

Kivonat

Napjainkban egyre nagyobb teret hódít a pilóta nélküli légi járművek alkalmazása. Az 1960-as években a hadszíntéren jelentek meg először, ahol megfigyelésre, felderítésre és olyan feladatokra használták, ahol kockázatos lett volna emberi életet veszélyeztetni. Az utóbbi években praktikussága, alacsony üzemeltetési költségei miatt más területeken is hasznosnak bizonyult ez a technológia, pl. geológia mintázatok kutatása, mely az emberi perspektívából nehezen észlelhető, tűzoltósági alakulatok koordinálása, otthoni hobby felhasználás.

Az MTA-SZTAKI Rendszer és Irányításelméleti Kutatólaboratóriumában kidolgozott szabályozó algoritmusok gyakorlatba való átültetésére egy pilóta nélküli járművet hoztak létre, mely a biztonságos üzemeltetés mellett, illetve az esetlegesen előfordulható hibák ellen redundáns hardware elemekkel védekezik. Szakdolgozatom keretében ennek a repülőnek a földi állomását dolgoztam ki, mely a redundánsan küldött rádiójelek feldolgozására és megfelelő megjelenítésre használandó. A földi személyzet mozgó térkép alapú vizualizáción láthatja az aktív útvonalpontokat és a gép útvonalát, a diagnosztikai adatokat és esetleges hibákat egy másik nézetben áttekintheti. Lehetőség nyílik repülési terv meghatározására és feltöltésére a repülőre, melynek fordulópontjait követi.

Abstract

Nowadays..

Bevezető

[1]A pilóta nélküli légi jármű gondolata egészen a XX. század elejére nyúlik vissza, mikor az I. világháborúban egy olyan távirányítású repülőt alkottak, mely robbanószerrel a fedélzeten a célpontba csapódva okozott kárt. Később a vietnámi háborúban több mint 3000 küldetésben vett részt ilyen repülő és mindössze 554 veszett oda. A technológiai korlátok miatt a fő funkcionalitása videó felvétel készítése egy meghatározott útvonalon (általában egyenes vonal, körökkel kiegészítve) repülve, majd a bázisra való visszaérkezés. A rádiótechnika fejlődése miatt egyre összetettebb feladatok elvégzésére lettek képesek. A nagyobb átviteli sebességnek köszönhetően valós időben, monitoron keresztül kezelheti az operátor a távirányítású repülőt. A mai UAV-k több üzemmódot is támogatnak, egyik az előbb említett távirányítás, másik a fedélzeti intelligenciára hagyatkozó. Mind a hagyományos repülőiparban, mind ebben az érában, szükséges és célszerű az emberi terhelés csökkentése, utasszállító gépek esetében is a robotpilóta elvégez minden olyan korrekciót, melyet azelőtt a pilóta folyamatos figyelésével, koncentrációjával lehetett elérni. A modern integrációnak köszönhetően, olyan fejlett feldolgozóegységgel dolgozhatjuk fel az adatokat, melyeknek nem jelentős a fogyasztása, nem foglalnak sok helyet. A szenzorokból érkező információkra, az algoritmusoknak köszönhetően úgy képes reagálni, hogy az nem veszélyezteti a repülő levegőben maradását. Ám hiába a fejlett hardware, a valóban automatikus üzemeltetés még mindig távoli cél, emberi beavatkozás mindig is kelleni fog olyan helyzetekben melyre nincs előre felkészítve az intelligenciája. Ilyen esetekben létfontosságú, hogy az operátor lássa a gép aktuális pozícióját, diagnosztikai adatait.

Feladatom egy olyan grafikus felhasználói felülettel ellátott földi irányító állomás kifejlesztése egy biztonságkritikus robotrepülőhöz, ami képes redundáns kommunikációs csatornán küldött adatok kezelésére és megjelenítésére. Ehhez szükséges megoldanom a kapcsolatot biztosító modem jeleinek vételét. Mivel hibatűrő kialakítása révén redundánsan szerelt, így a párhuzamos csatornákon érkező adatok egymástól való eltérésének a kijelzése is megvalósítandó. Az eltérésnek több oka is lehetséges, előfordulhat, hogy az egyik meghibásodásából következően nem küld semmilyen értéket vagy amit küld az teljesen hibás, Továbbá, ez a program a földön tartózkodó személyzet kiszolgálására készül, így ami a legfontosabb, hogy ők a repülővel kapcsolatos információkat könnyen értelmezhessék. Így szükséges megoldani, hogy a kialakítandó felület felhasználóbarát legyen, ehhez több nézeti oldal szükséges:

- Egy áttekintő képernyő, mely a legfontosabb adatokat jeleníti meg a repülővel kapcsolatban (pozíció, sebesség, irány)

- Egy tervező modul, amin a lerepülendő útvonalhoz tartozó fordulópontok kijelölése lehetséges
- Egy diagnosztikai nézet, melyen az alacsonyabb prioritású adatok tekinthetőek át

1. fejezet

Előzmények

1.1. Motiváció

Az MTA SZTAKI Rendszer és Irányításelméleti Kutatólaboratóriumában folyó kutatások eredményének demonstrálására szükség volt egy kézzelfogható eszköz megalkotására. A szabályozó algoritmusok működésének bemutatásának az egyik leglátványosabb módja egy repülőgép irányítása. Egy stabil repülési tulajdonságokkal bíró repülő levegőben tartása sem triviális, szükség van a kormányszervek harmonikus mozgatására, a tolóerő szabályzására. Ezen túlmenően ha feladatokkal látjuk el, pl. fordulópontokat követve feltérképezni az alatta lévő területet, már számolni kell a széllel, mely eltérítheti az útvonaláról, felszálló légáramlatokkal, melyek ellen gyors reagálással kell válaszolnia. Mivel egy teljesen felszerelt repülő összeállítása, felprogramozása nagy szakértelmet, sok időt, energiát igényel és nem utolsó sorban anyagi ráfordítást, egy esetleges meghibásodás jelentős kárt okozna. Ezen okok miatt felmerült az igény a repülő megbízhatóságának növelésére, így a most folyamatban lévő „Nagy megbízhatóságú pilóta nélküli légijármű projekt" keretében egy olyan avionikai rendszer fejlesztése is folyik, amelyben cél minden egyes repülőgép alrendszer meghibásodásának diagnosztizálása, a diagnosztikai információkat felhasználva a repülőgép átkonfigurálása.

1.2. Repülőgép felépítése

A jelenlegi repülő egy saját építésű 3.2 m fesztávolságú modell, melybe diagnosztikai és hibadetekciós célokra egyedi tervezésű komponensek kerültek. A kommerciális célokra szánt szervo motorok nem szolgálnak elég információval a kitérésükről, fogyasztásukról, ezért ezek méréséről gondoskodni kellet.

1.1. ábra

1.3. Hibatűrés

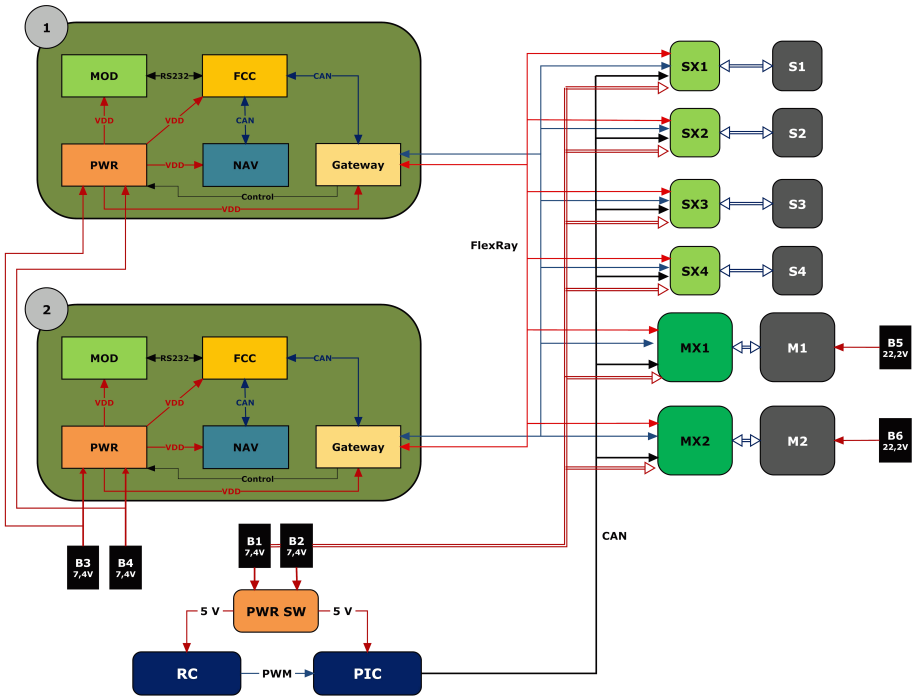
Elsődleges szempont, hogy egy komponens meghibásodása ne okozza a gép vesztét, tehát a rendszerben ne maradjon SPOF [1]. Ezt redundanciával érhetjük el, mely során a repülőgépen duplikáljuk azokat az elemeket, melyek nélkülözhetetlenek a levegőben maradásához:

- motor
- kormányfelületek és ezeket mozgató motorok
- tápellátás
- központi számítógép

A repülő méretéből adódóan térbeli szeparációval nem lehetséges a megbízhatóságot növelni, mint pl. harci repülőgépek fedélzeti számítógépeinél, melyek a találat kockázata miatt elszórva, akár 4–5x-ve vannak.

1.2. ábrán látható a kialakított architektúra. A szenzorokból érkező jelek a duplázott központi feldolgozó egységekbe jutnak, ahol egy–egy Gateway felelős a kommunikáció lebonyolításában. A fogadott jelek feldolgozása[2] után a szabályozó algoritmusok kiadják a megfelelő utasítást a kormányszerveknek. Ilyen utasítás lehet pl, a csűrőlapok adott fokban történő elmozdítása, motor fordulatszámának változtatása. A motorok, úgy mint a központi egység, külön tápellátást kapnak. A dolgozat szempontjából egyik legfontosabb elem a kommunikációért felelős modem, mely soros porton keresztül kapcsolódik az FCC-hez. A vevő oldalon hasonló modem, szintén soros porton küldi a földi állomásnak a vett jelet, a köztük lévő vezeték nélküli kommunikáció saját szabványa a gyártónak. A választás [3]XBee-PRO 868 típusú modemre esett, mely alacsony fogyasztása és nagy hatótávolsága miatt ideális egy ilyen környezetbe.

[1] Single Point Of Failure, olyan meghibásodás, mely ha bekövetkezik, az egész rendszer leállásához vezet
[2] FCC



1.2. ábra

1.4. Vezeték nélküli modem

A kiválasztott modem kétféleképpen képes kommunikálni:

- API csomagküldés
- AT transzparens

API módban egy eszköz több eszköztől tud csomagokat venni, ha egy csomag megérkezett a küldőtől a fogadóig, egy ACK[3] üzenettel válaszol, ha ezt nem kapja meg, újraküldi. Egy csomag többek közt tartalmazza a küldő és a fogadó címét, lehetőség van broadcast üzenetek küldésére is, melyet minden eszköz megkap. Továbbá az adatintegritás céljából checksum mezőjében összesítve van egy csomag adata.

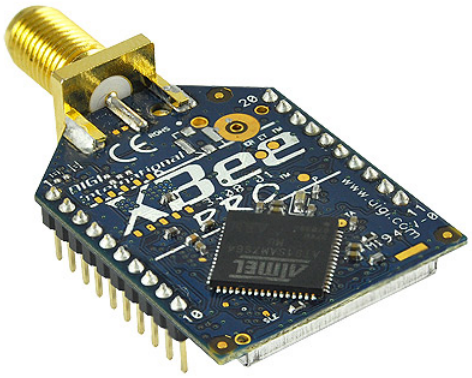
AT mód nem más mint egy vezeték nélküli kapcsolat 2 sorosport közt. Nem csinál mást, mint a soros portján bejövő adatokat egy XBee szabvány szerint rádiójelekké alakítja, melyet a virtuális kapcsolat végpontja fogad és visszaalakítja soros portra. Ez pont-pont kommunikációra hivatott.

1.5. Földi állomások

Egy UAV vezetéséhez elengedhetetlen egy bázis, ahonnan a földi személyzet irányítja, monitorozhatja a repülést. Általában több funkciót lát el:

- Küldetés tervezés: útvonal meghatározása, illetve a célpontok kijelölése

[3] Acknowledgement, nyugtázó üzenet



1.3. ábra

- Adatok megjelenítése: megfelelő módon kijelezni a repülőgép állapotát, esetleges hibáit.

Számos megoldás született a földi állomás GUI[4]-jának kialakítására. Elsődleges követelmény, hogy az operátor mindig a legfontosabb információkat láthassa, ehhez szoftverergonómiailag kell megtervezni a műszerek, adatok elrendezését. Az alábbiakban összehasonlításra kerülnek a megjelenítési felületek.

1.5.1. ArduPilot

Az Ardupilot projekt létrejöttének motivációja, hogy otthoni körülmények között, nem ipari alkatrészekből bárki összeállíthasson egy autonóm járművet, mely az előre betáplált utasításokat végrehajtja. Központi eleme az Arduino cég által készített ATMel mikroprocesszorra épített platform, mely felhasználóbarátabbá teszi a mikroprocesszor programozását. Magas szintű utasításokkal segíti az eszközzel való barátkozást, nincs szükség assembly szintű tudásra. Erre a platformra hozták létre az ArduPilot programot, képes vezérelni többféle autonóm járművet:

- ArduPlane néven futó változat: robotrepülőgép
- ArduCopter: 1, 3, 4, 6, 8, propelleres helikopter
- Ardurover: 4 kerekű autó

Sikerességének fő oka, a nyílt forráskód, Arduino panel alacsony ára (5 ezer Ft) és a projekt mögött álló lelkes közösség.

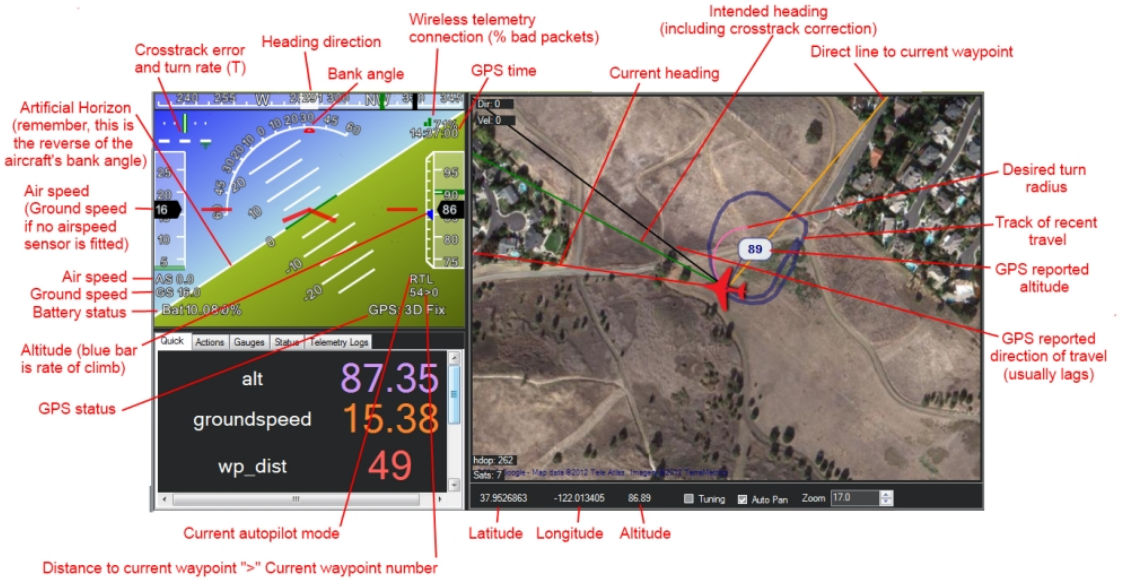
Ennek a közösségnek köszönhetően született a Mission Planner nevű szoftver, mely teljes körű támogatást nyújt a járművekkel kapcsolatban.

Főképernyő

Több nézet segítségével könnyen átlátható a funkcionalitása, repülés szempontjából a főképernyőn a legfontosabb adatok találhatóak. A repülőgép orientációját, sebességét, irányát

[4]Graphics User Interface, grafikus megjelenítés

egy műhorizonton láthatja a felhasználó, ez vizuálisan szemlélteti, hogy hány fokos bedöntéssel repül, milyen állásszöggel emelkedik. Alatta lévő területen előre beállított adatokat jelenít meg, pl. GPS magasság, GPS sebesség, szélirány(valós mágneses irány és a sebesség vektor különbségéből számítható). A legnagyobb területet a térkép foglalja el, melyen az aktuális irány, lerepült útvonal, fordulópontok láthatóak. Megfigyelhető, hogy ez kapja arányaiban a legnagyobb területet.



1.4. ábra

Ez a nézet akkor jöhet jól, mikor olyan meghibásodás történik, melyre nincs felkészítve az irányítóegység és szükséges lehet a kézi üzemmódra váltás. Előfordulhat ilyen esetben, hogy nincs vizuális rálátás az operátor és a repülőgép között, ekkor csak az itt látható műszerek és térkép alapján szükséges irányítania. Szerencsére ez az avionikában már bizonyított, hogy műszerek segítségével, „vakon" is lehetséges repülni, itt azonban ez az eset csak pár percig szükséges.

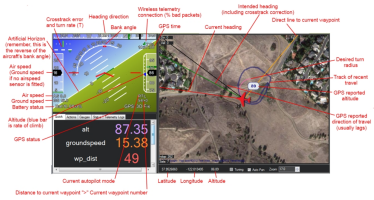
Tervező és útvonalfeltöltő

Másik nézet lehetőséget biztosít útvonalpontok kijelölésére, az útvonalpontokhoz egy listából kiválasztható, hogy az adott pont start-, forduló-, végpont és egyéb lehetőségek. Az útvonalpontok pozícióját egérrel módosíthatjuk, törölhetjük. Bal felső sarokban a kijelölt tervnek hosszát láthatjuk, ez segítséget nyújt, nehogy túlhaladjuk a rádiókapcsolat és a repülő hatósugarát. Az elkészített tervet feltölthetjük a csatlakoztatott eszközre.

1.5. ábra

Beállítások

Itt lehetőség van a nyelv és mértékegységek kiválasztására, naplózás sűrűségének meghatározására. Egy másik oldalon az aktuális járművet lehet beállítani, mivel más adatok fontosak egy földi és egy légi jármű esetében, továbbá a kommunikációs protokoll is változó. Nem külön oldalon, hanem mindig látható a csatlakozás gomb, melynél a soros port és a jelsebesség beállítása után a csalatkozás gombbal csatlakozik a program a portra.



1.6. ábra

Terminál

Ezen az oldalon az USB-n csatlakoztatott eszközt lehet parancssoron konfigurálni. Az eszköz saját LOG fájljait letölteni.

Egyéb képernyők

Lehetőség van még az elmentett log fájlok szimulálására, van egy súgó oldal a program használatával kapcsolatban, illetve egy támogatói gomb, mely átirányít egy PayPal oldalra, ahol különböző összeggel támogathatjuk a fejlesztést.

Összegzés

Felhasználói szempontból barátságos felületet biztosít a különböző nézetekkel, gombok átgondolt elhelyezésével. Nem a program hibája, de nincs felkészítve párhuzamos csatornák kezelésére, ez az ArduPilot projekt egyszerűségének köszönhető, mivel nem használ redundanciát. De mivel a repülő, melyhez a program készül, nagy megbízhatóságú, így szükséges megoldani a 2 port kezelését, és az azokon érkező adatok feldolgozását. Jó ötlet a csatlakozás gomb mindig látható elhelyezése, a főképernyőn térképen ábrázolni a repülő helyzetét, illetve a műhorizont. Azonban hibadiagnosztikai kijelzés nincs megoldva, melynek szintén fontos a megvalósítása.

1.5.2. Paparazzi

Ez egy több platformos GCS, mely különböző projektekben létrehozott repülőkhöz használható valós-idejű monitorozásra. Könnyen testre szabható az igényeknek megfelelően.

2. fejezet

Tervezés

2.1. Kommunikáció

A kommunikáció csatornánként 2 db modem segítségével történik, a modemek egymás közt vezeték nélkül csatlakoznak, felhasználói oldalon soros portot biztosítanak. Így az elkészítendő programnak elég csak a soros port jeleinek vételével foglalkoznia. A modemek adatátviteli sebessége változó lehet, így azt a felhasználó egy listából választhatja csatlakozás előtt.

2.2. Adatok fogadása

A redundanciából következően külön, külön kell kezelni a 2 párhuzamos csatornán kapott adatokat. Mivel aszinkron módon érkeznek a csomagok, így kettő tároló kell, mely a legutóbb küldötteket tárolja.

2.2.1. Protokoll

Az adatokat a repülőgép 2 Hz-s frekvenciával küldi, ezek csomagokban érkeznek, melyeknek a felépítése:

A skálázás és offset képzés azért szükséges, hogy az adott szélességen (8, 16, 32 bit)

bájt index	leírás	típus	skálázás	offset
1	start	char(fix 'U')		
2	start	char(fix 'U')		
3	start	char(fix 'T')		
4	idő	uint32	10000	0
8	magasság	uin16	0x7fff/1000	
...				
TODO	északi irány 1/2	unsigned short	0x7FFF/400	200
28	keleti irány 1/2	unsigned short	0x7FFF/400	200
30	lefelé irány 1/2	unsigned short	0x7FFF/400	200
...				

2.1. táblázat. *Fogadás protokollja*

minél több biten legyen ábrázolva egy érték, mivel kis változások esetén a Hamming-távolság[1] kicsi lenne az eredeti számábrázoláson. Ahol szükésges, ott a visszakódolás az alábbi formában történik :
$eredeti = (nyersadat/skalazas) - offset$
Az értékek megfelelő kiválasztása a minél nagyobb szétszóráshoz szükséges.

2.3. Adatok küldése

A repülőgép által lerepülendő feladat útvonalpontjait hasonlóképpen, mint az adatok fogadását, vezeték nélküli csatornán küldjük fel. A feltöltendő adat küldésének protokollja létfontosságú, mivel ha valamilyen hiba kerül a kommunikációba akkor az akár végzetes is lehet. Gondolok itt olyan hibára, hogy egy fordulópont koordinátája úgy kerül feltöltésre, hogy az kiesik a repülő hatósugarából és ezzel nem számolva, lemerül a tápellátást szolgáló akkumulátor. Az ilyen hibák ellen célszerű a feltöltés protokolljába hibadetektálást építeni, hogy ezek a feldolgozás(esetlegesen felszállás) előtt kiderüljenek ki.

2.3.1. Protokoll

Több megoldás is lehetséges a fordulópontok feltöltésére:

- Rögzített maximális darabszám elküldése egy csomagban
- Változó darabszám esetén egy fordulópont egy csomagban

Az első megoldásban rögzítenénk a fordulópontok maximális számát. Mely azt eredményezné, hogy egy csomagban el lehetne küldeni az egész lerepülendő feladatot. Ha egy pont koordinátájának ábrázolására elég 2*4 byte, így ha feltételezünk egy MAXPOINT byte-os csomagot, akkor abba beleférne kb. MAXPOINT darab. Egy csomag állna egy fejlécből, a feltöltendő pontok számából, a pontokból, és egy checkszámból.

Másik lehetőségnél N db-t lehetne feltölteni: egy csomag szerkezete: fejléc, küldendő pontok száma, aktuális pont sorszáma, koordinátái, checkszám. Ha a küldendő pontok száma és az aktuális pont sorszáma megegyezik és megérkezett minden csomag akkor ACK-val válaszol ha kész a feltöltés.

Mivel az eddig használt megoldásban a kódba „bele volt égetve” az útvonalterv, mely 5-6 pontot tartalmazott, az első megoldás tűnik kedvezőbbnek. Fogadó oldalon is könnyebb egy ilyen lehetőségre felkészíteni. Ha esetlegesen a jövőben több fordulópontont feltöltésére lesz igény, az is megoldható kis módosítással.

A modem is duplikált, így a 2 soros portra csatlakoztatott modemmel redundánsan küldjük el a csomagot. TODO TODO??? Ha a csomag sértetlenül megérkezett, ACK jelzéssel válaszol.

[1] Bináris számok XOR képzésével kapott 1-esek száma

bájt index	leírás	típus	skálázás	offset
0	start	bájt(fix 'G')		
1	start	bájt(fix 'P')		
2	start	bájt(fix 'S')		
3	pontok száma	bájt		
4	pontok[0].lat	uin32	0x7fff/360	180
...				
8	pontok[0].lon	uin32	0x7fff/360	180
...				
12	pontok[1].lat	uin32	0x7fff/360	180
...				
16	pontok[1].lon	uin32	0x7fff/360	180
...				
78	checksum 1/2	uin16		
79	checksum 2/2	uin16		

2.2. táblázat. *Küldés protokollja*

2.4. Grafikus felület

Az előző fejezetben ismertetett grafikus felületekből levonva a következtetéseket, nyilvánvaló, hogy a GUI kialakításában fontos a repülőgép aktuális pozíciójának térképen való mutatása, az repülési állapot könnyen értelmezhető megjelenítése, illetve az esetlegesen előforduló problémák feltűnő jelzése.

2.4.1. Főképernyő

A főképernyőn látható lesz a repülőgép aktuális pozíciója és iránya. A pozicionálást segítendő, egy térkép lesz egy repülőgép ikon hátterében. Ez a rész a képernyő kb. 2/3-át fogja elfoglalni. Az oldalsó sávban a „Glass Cockpit" kerül kialakításra, ez egy műhorizont, mely a gép által küldött orientációs adatokból az operátor számára informatívan ábrázolja a repülési állapotot. Ez a nézet tartalmazza még a gép aktuális sebességét, irányát, melyet egy iránytű segítségével láthat. Valószínűleg ez a képernyő lesz legnagyobb százalékban használva, így a kritikus hibákról itt kell feltűnő értesítést adni.

2.4.2. Tervezés képernyő

Ezen a képernyőn a felhasználó kijelölheti a lerepülendő útvonalhoz tartozó fordulópontokat, melyet csatlakozás után aszinkron módon feltölthet a repülőre. Mivel a kommunikációs protokoll MAXPOINT pontot enged meg, így ennél többet itt ki sem jelölhet, a lerakott pontok helyét a megszokott Google Maps-hoz hasonló módon hosszan kattintva átrakhatónak kell lennie, illetve köztes pontoknak törölhetőeknek kell lenniük.

2.4.3. Diagnosztikai képernyő

A 2 porton érkező dekódolt értékek látszódnának 2 oszlopban, mellettük egy hibaérték, mely a különböző hibatípusok hibaszámának összege lenne.

Hibatípusok

- beragadás
- túl nagy változás
- túl nagy különbség a 2 vett értéken

Ezek feldolgozására 2 FIFO sort kell alkalmazni, melyek visszamenőleg tárolják a beérkező értékeket. Ez azért szükséges, mivel így a túlságosan kiugró értékeket detektálni lehet, illetve, ha az egész sorban ugyanazok az értékek vannak, gyanús a beragadás esélye. A harmadik esetben sajnos nem lehetséges a „jó" kiválasztása, mivel nem tudjuk, melyik modemből érkezett adat a megfelelő. Ezt csak háromszorozással és többségi szavazással lehetne megoldani.

3. fejezet

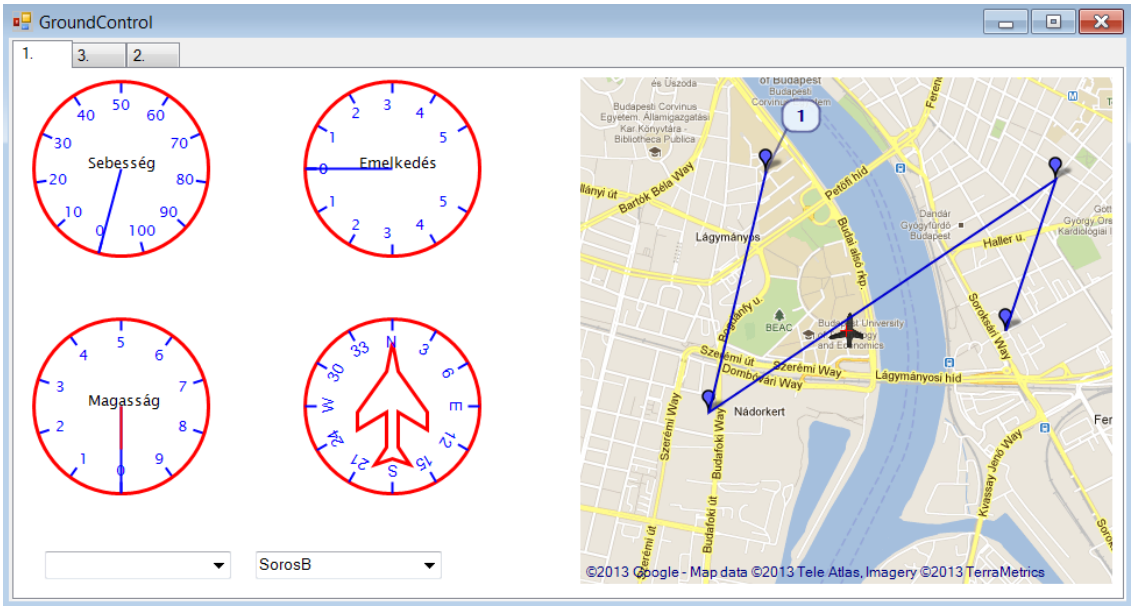
Megvalósítás

Az előző fejezetekben összegyűjtöttem a megvalósításhoz szükséges információkat, tervezési lépéseket. Ebben a fejezetben a konkrét implementációt fogom bemutatni.

3.1. .NET

C# nyelven szükséges a program implementálása, a .NET keretrendszer szerencsére sok API[1]-t biztosít a fejlesztők számára, hogy megkönnyítse és gyorsítsa folyamatot. Így a grafikus megjelenítéshez GDI-t használhatok, illetve a soros porti kommunikációra a SerialPort osztály adta lehetőségek.

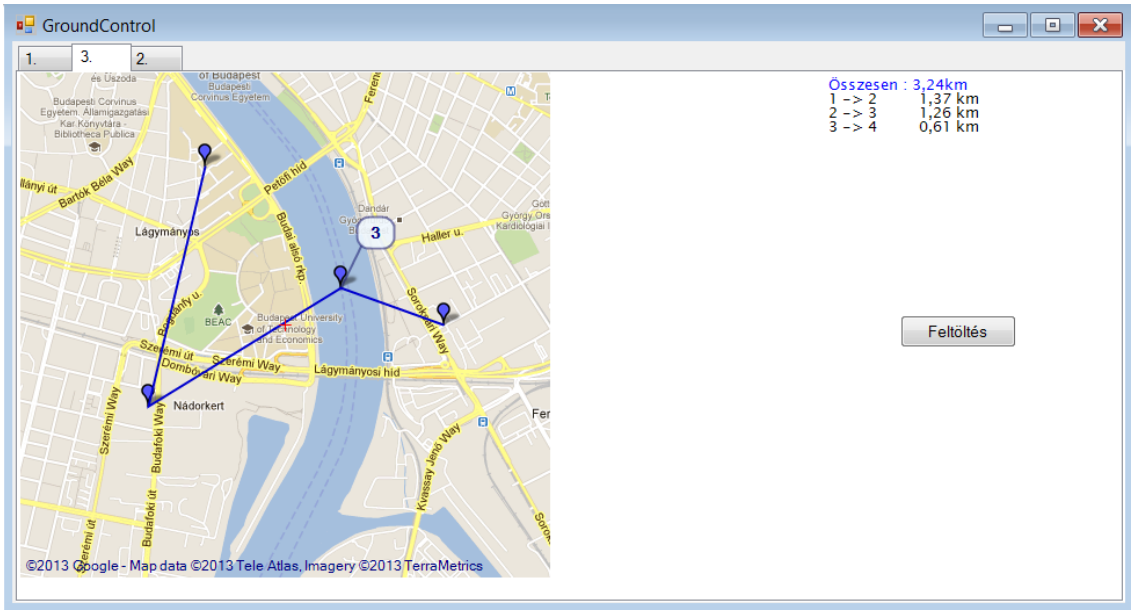
3.2. Főképernyő



3.1. ábra

[1]Application Programming Interface, mely előre megírt komponensek használatához biztosít interfészt

3.3. Tervezés képernyő

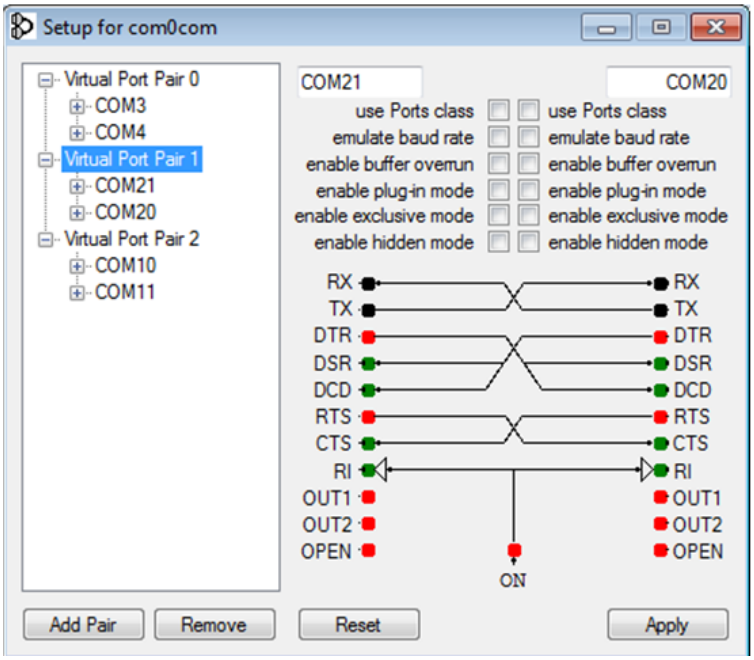


3.2. ábra

3.4. Diagnosztikai képernyő

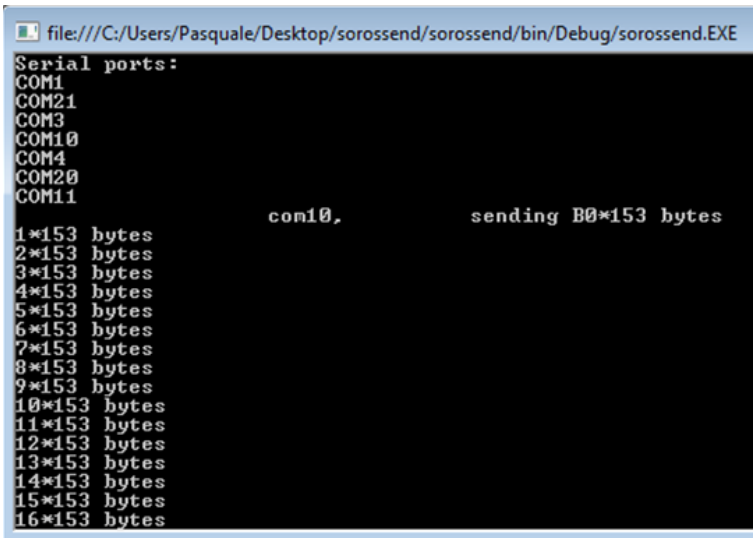
3.5. Soros port

A modemből érkező adatokat soros porton keresztül fogadja a program, a tesztkörnyezet felállításához HIL adatok szolgáltak. A küldött log fájlokat egy programmal beolvasom és egy [5]null-modem segítségével sorosporton keresztül küldöm a megfelelő portra.



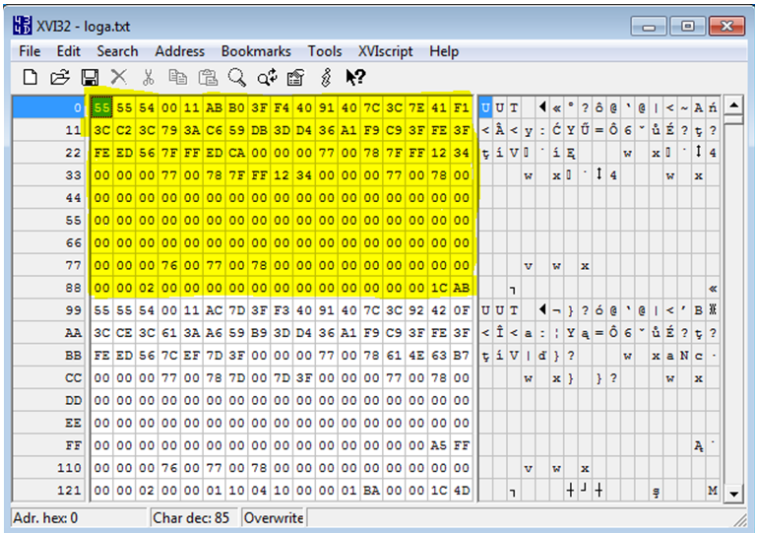
3.3. ábra

Beállítottam 2 párt, COM20-COM21 és COM10-COM11 közt, a páros küldöm, páratlanon fogadom az üzeneteket.



3.4. ábra

153 bájtos egy csomag, melyet egy UUT 3 bájtos fejléc és egy 2 bájtos checksum zár. A checksum a hasznos bájtok 16 bitre csonkolt összege. Minden fogadott csomagnál, a feldolgozás előtt kiszámolom az összeget és ellenőrzöm, az egyezést, a rossz csomagok egyelőre eldobásra kerülnek.



3.5. ábra

Fogadó oldalon a két sorosport aszinkron ír 1-1 byte tömböt, melyből egy dekódoló függvénnyel nyerjük ki a sebesség, pozíció, irány, stb. adatokat.

```
public double[] Decode(byte[] array)
```

Ebben a függvényben ellenőrzöm, a checksum-ot, illetve a kezdő UUT bájt hármast. Mivel bájtosával lehet feldolgozni az adatokat, így pl. a 4 bájtos időbélyeget 4 db egymás után jövő bájtból kell összerakni:

```
uint ido = (uint)array[3]<<24 | (uint)array[4]<<16 |
(uint)array[5]<<8 | (uint)array[6];
```

Ugyanígy folytatódik az adatok feldolgozása, az előre megadott protokoll szerint.

4. fejezet

Értékelés

Megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztetési lehetőségek

5. fejezet

Összefoglalás

SZUMM

A program megalkotásához nem használok automatikus kódgeneráló és formális módszereket támogató segédeszközöket, így a fejlesztési folyamatba 1000 sorból kb.10 hiba[2] maradhat a rendszerben. Ami odafigyeléssel és teszteléssel, illetve a tervezés során elvégzendő validációval és verifikációval lejjebb vihető. Az implementáció során elkövetett meghibásodásból, pl. egy számláló rosszul megírt növeléséből, ha ráfut a vezérlés akkor hiba keletkezik, mely a felhasználó szempontjából hibajelenséget okoz. A hatásláncot a meghibásodási tényező csökkentésével és hibajelenség kialakulásának megakadályozásával lehet befolyásolni. Előbbit ellenőrzéssel és teszteléssel, utóbbit helyes kivételkezeléssel. Több hibatípus léphet fel, egyik az előre ismert, másik az előre nem ismert. Az előre ismert hibatípusokat optimálisan lehet kezelni a tervezés során, az előre nem ismertek ellen megfelelő rendszerstukatúra kialakítása szükséges. Esetünkben előre ismert hiba lehet

- egy vagy kettő csatorna kiesése
- csomagvesztés
- küldött érték beragadása
- küldött érték túl gyors változása

Irodalomjegyzék

[1] http://www.azom.com/article.aspx?ArticleID=10156, 2013. november 19, 10:00

[2] http://www.inf.mit.bme.hu/sites/default/files/materials/category/kategória/oktatás/bsc-tárgyak/rendszermodellezés/13/Hibamodellezes.pdf, 2013. november 17, 19:00

[3] http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-868, 2013. október 29, 14:00

[4] http://msdn.microsoft.com/en-us/library/system.io.ports.serialport.aspx, 2013. március 20, 10:00

[5] http://com0com.sourceforge.net/, 2013. május 4, 10:00

Függelék

F.1. Függelék1