Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

SMOG-1 műhold földi állomás kliens szoftver fejlesztése

Szakdolgozat

Készítette
Kálmán Tibor

Konzulens
Dudás Levente

2016

Tartalomjegyzék

Ábrák jegyzéke

Rövidítések jegyzéke

BER Bit Error Rate

CPU Central Processing Unit

DVB-T Digital Video Broadcasting - Terrestrial — Digitális Földfelszíni Műsorszórás

FFT Fast Fourier Transform

FIFO First In First Out

FIR Finite Impulse Response

FM Frequency Modulation

FSK Frequency Shift Keying

GMSK Gaussian Minimal Shift Keying

HTTP HyperText Transfer Protocol

IP Internet Protocol

LEO Low Earth Orbit

NORAD North American Aerospace Defense Command

OBC On-Board Computer — Fedélzeti Számítógép

PER Packet Error Rate

PPM Parts-Per-Million

REST Representational State Transfer

SDR Software Defined Radio

TCP Transmission Control Protocol

TCXO Temperature Compensated Crystal Oscillator — Hőmérséklet Kompenzált Kristály Oszcillátor

TLE Two-Line Element Set

USB Universal Serial Bus

UTC Coordinated Universal Time — Koordinált Világidő

VFO Variable Frequency Oscillator

HALLGATÓI NYILATKOZAT

Alulírott *Kálmán Tibor*, 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 (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) 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, 2016. december 8.

Kálmán Tibor
hallgató

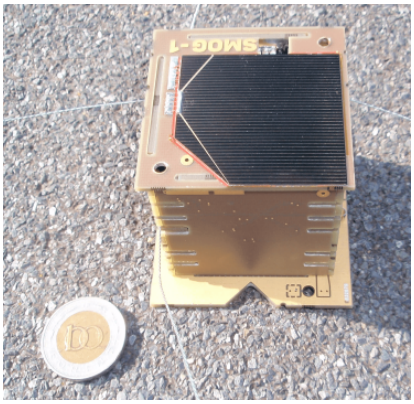
Kivonat

Szakdolgozatom során egy platformfüggetlen kliens szoftvert készítettem a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatóinak és oktatóinak fejlesztése alatt álló PocketQube osztályú ($50 \times 50 \times 50$ mm) SMOG-1 műholdhoz.

A műhold elsődleges küldetése a földfelszíni adók által kibocsájtott DVB-T sávbeli elektromágneses sugárzás mérése, a világűrben[1][2][3]. Reményeink szerint a mérési eredmények alapján az földi antennák kialakítása tovább lesz javítható, hiszen ami elektromágneses szmog jelenleg az űrbe kijut, az feleslegesen lett kisugározva. Másodlagos küldetésként totál ionizáló dózist is fog mérni a műhold.

A SMOG-1 az olasz UniSat-7 belsejében, egy orosz gyártmányú Dnyepr rakéta orrába fog kerülni, mely rakéta várhatóan 550-650 km-es alacsony Föld körüli pályára fogja juttatni őket. Az űrbe jutás után az UniSat-7 kilöki magából többet közt a SMOG-1-et is, mely végül olyan pályára fog állni, mely körülbelül napi 4-6-szor fog Magyarország fölött áthaladni, az egyes áthaladások pedig 2-12 perc hosszúak lesznek. Ahhoz, hogy az áthaladások során az E épület tetején lévő állomással követni tudjuk a műholdat, szükség lesz az antenna azimutjának és elevációjának folyamatos szabályozására, továbbá a Dopplerhatás miatt a vételi frekvenciát is folyamatosan korrigálni kell majd. A Föld különböző pontjain élő rádióamatőrök is fogják a szoftver segítségével a műholdat követni, jelét venni, és számunkra eljuttatni a mérési eredményeket, telemetriát. Ez a módszer a Masat-1 üzemeltetésénél is bevált, és azt reméljük, hogy most is eredményes lesz.

A készített szoftver tervezési szempontjai között szerepelt az, hogy platformfüggetlen legyen, képes legyen felhasználói beavatkozás nélkül működni, és idővel nyílt forráskódúvá válhasson.



1. ábra. *A fejlesztés alatt álló SMOG-1 PocketQube műhold*

Abstract

As my BSc thesis I've developed a platform independent client software for the PocketQube class ($50 \times 50 \times 50$ mm) SMOG-1 satellite that is currently under development at the Budapest University of Technology and Economics, by students and lecturers.

The primary mission of the satellite is the measurement of terrestrial electromagnetic radiation in the DVB-T band, in Space[1][2][3]. With our findings, we hope that antenna design can be further improved, since the electromagnetic smog currently being radiated into Space is a form of waste. As a secondary mission, the satellite will also measure Total Ionizing Dose.

SMOG-1 will be launched into Space inside the UniSat-7 satellite, using a Dnepr rocket, that is expected to put the satellites into Low Earth Orbit, at around 550-650 kilometers above the Earth's surface. In Space, UniSat-7 will eject SMOG-1 (among others) and our satellite will pass over Hungary about 4-6 times, each pass lasting between 2 and 12 minutes. To be able to track SMOG-1 during these passes with the primary ground station located on top of building E, the azimuth and the elevation of the antenna will have to be continuously controlled, and due to the Doppler effect, downlink frequency requires precise adjustment as well. Using the software, radio amateurs around the globe will be able to track the satellite, receive its signal, and send us measurement and telemetry data. This method worked very well during the operation of Masat-1 and we hope that it will be just as successful this time around.

Among the design principles of the software were platform independence, the capability of operating without user input and the possibility of open sourcing the project later on.

Bevezető

A Budapesti Műszaki és Gazdaságtudományi Egyetemen készül - reményeink szerint - az ország második műholdja, a SMOG-1. Ez a PocketQube osztályba tartozik, ezért mérete körülbelül $50 \times 50 \times 50$ mm lesz. Elsődleges küldetése elektroszmog mérése lesz az űrben, másodlagos feladata pedig ionizáló sugárzásmérés. A kommunikáció a rádióamatőr sávban történik majd, ezért bárki veheti majd a sugárzott jelet.

Az első magyar műhold, a Masat-1[4][5][6][7][8][9] 2012. február 13-án a Vega-1 hordozórakéta segítségével a világűrbe került. Ezt valószínűleg sokan tudják, azt viszont talán nem, hogy a pályára állásával nem értek véget a fejlesztői csapat teendői. Élete során folyamatosan követni kellett a műholdat, több okból is. Egyrészt azért, hogy pontosan tisztában legyenek a pillanatnyi állapotával, és azért is hogy a lehető legjobban kihasználják a rendelkezésre álló időt. A műholdkövetés fontos, ugyanakkor gyakran monoton feladat, ezért a csapat nagy része nem nagyon vett benne részt. Szerencsére a feladatok jó része automatizálható[10][11], ezért készült egy szoftver, mellyel ezeket el lehetett végezni.

A SMOG-1 pályára állása 2017-re várható és azt szeretnénk, ha a pályára állás pillanatában már elérhetőek lennének azok a szoftverek, melyekre szükség lesz a műhold élete során. Kelleni fog egy szoftver, mely segítségével bárki tudja a műholdat követni, a jelét dekódolni és a fejlesztő csapatnak továbbítani. Ezt készítettem el én, és még fogom is fejleszteni, amíg a műhold el nem készül, hiszen a műhold fejlesztése egy hosszú folyamat, ami még nem ért véget.

Így tehát a feladatom egy olyan platformfüggetlen szoftver készítése volt, mely ellátja ezeket a feladatokat. A fejlesztésnél természetesen a csapat korábbi tapasztalataira és megoldásaira is érdemes volt támaszkodni, hiszen van már némi tapasztalatuk műholdkövetésben. Fontos kiemelnem, hogy az általam készített szoftver egyáltalán nem tartalmaz a korábbi programból kódot, nem azt alakítottam át. Azért, hogy minél több embert elérhessen ez a szoftver, a platformfüggetlenséget erősen támogató Qt keretrendszer felhasználásával készült. A megcélzott platformok a Linux, a Mac OS X és a Windows. Fontos szempont volt az is, hogy nyílt forráskódú, szabadon hozzáférhető legyen a szoftver (a sikeres pályára állítást követően).

A szakdolgozatomban először pontosabban ismertetem a kihívásokat és azok okát, majd adok egy rövid áttekintést a Qt keretrendszer lényeges tulajdonságairól. A dolgozat ezt követő fejezeteiben egymás után ismertetem azon megoldásokat, melyek az egyes kihívásokra születtek. Lezárásképpen összegzem az eredményeket és ismertetem azt is, hogy milyen teendők várnak rám a SMOG-1 pályára állásáig.

1. fejezet

Alacsony pályás műholdkövetés kihívásai és eddigi megoldások

1.1. Alacsony pályás műholdkövetés kihívásai

Ahhoz, hogy a műhold pályájával járó kihívásokat jól átláthassuk, szükséges először is azt tisztázni, hogy mi is az, hogy alacsony pálya. Az alacsony Föld körüli pálya (más néven LEO)[12] olyan pálya, melyen az objektumok körülbelül 200 és 2000 kilométer közötti távolságban vannak a Föld felszínétől. Fontos előnye, hogy erre a pályára a legkönnyebb (és legolcsóbb) feljuttatni egy tárgyat, lévén, hogy ehhez kell a legkevesebb hajtóanyag. Másik előnye is a távolságból adódik, alacsony pályáról kell a legkevesebb energia a kommunikációhoz, hiszen itt kell a legkevesebb utat megtennie az üzeneteknek. Hátulütője, hogy a többi pályához képest kisebb az élettartama, hamarabb a légkörbe zuhannak róla a műholdak és egyéb szerkezetek. Ez persze nem mindig probléma, a mi esetünkben sem az, mivel a műhold tervezett élettartama kisebb, mint a pályáé. Nagyobb fejtörést okoz az, hogy a sikeres kommunikációhoz nagy nyereségű antennát kell használni a Földön és ezért viszonylag pontosan le kell követni a műholdat, legyen szó azimutról, elevációról, vagy éppenséggel a kapott vagy adott jel frekvenciájáról. A frekvencia azért érdekes, mert a Doppler-hatás ekkora sebességeknél ($\sim 7,5 km/sec$) olyan nagy eltérést okoz a névleges frekvenciától, hogy ha nem vennénk figyelembe, akkor lehetetlen lenne bármiféle kommunikáció megvalósítása.

Összességében tehát elmondható, hogy szükség van valamilyen módszerre, aminek segítségével a Föld felszínének tetszőleges pontjából, tetszőleges pillanatban meghatározható a műhold helyzete és sebessége. Mivel ezek a a nehézségek mindenkit érintenek, aki alacsony pályás műholdkövetéssel vagy vezérléssel foglakozik, ezért léteznek már megoldások. Az én feladatom nem ezek újrafeltalálása volt, hanem megismerésük, a számunkra legalkalmasabb open source eszköz kiválasztása, és annak integrálása, az antenna forgató és különféle vételre alkalmas eszközök vezérlése.

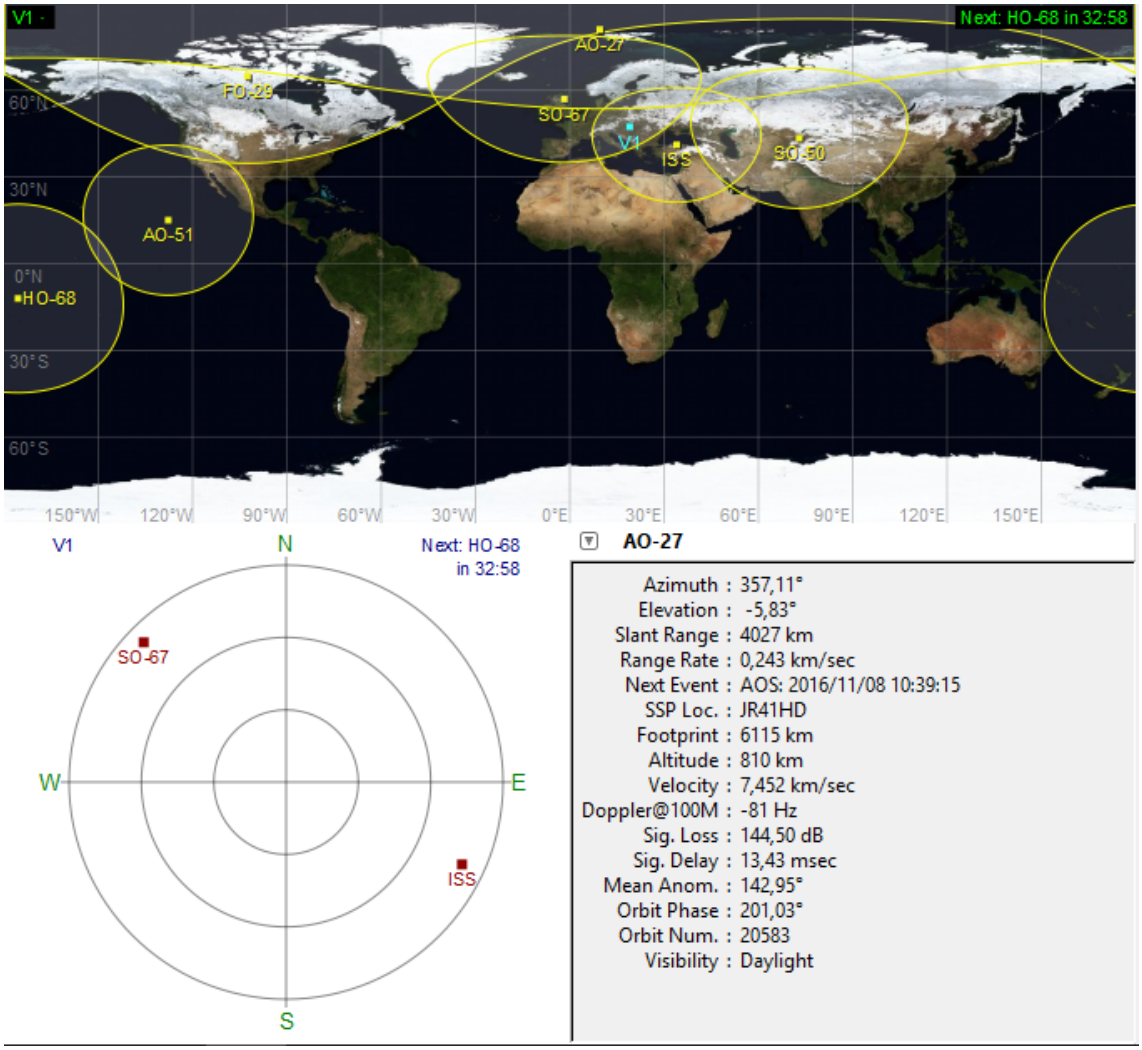
Nem mellékes az sem, hogy a kapott jelet valahogyan demodulálni és dekódolni is kell, hiszen ezek nélkül információt nem tudunk belőle kinyerni. Ezekhez olyan megoldások születtek, melyek mögött komoly szakmai háttér van, így természetesen nem én készítettem el őket, az én feladatom itt is ezek beépítése, majd a kapott adatok feldolgozása, megjelenítése

volt.

1.2. A csapat által eddig használt és fejlesztett megoldások

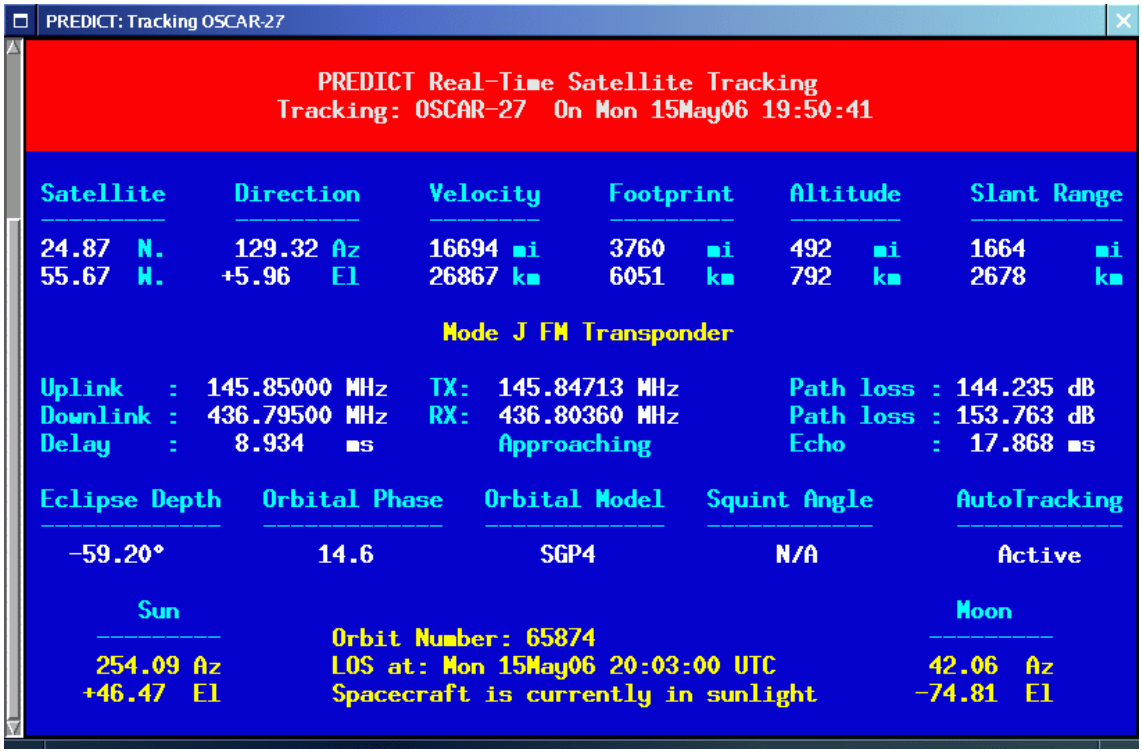
A SMOG-1-et készítő csapat néhány tagja a Masat-1 készítésében, követésében és vezérlésében is részt vett. Ahhoz is készült annak idején egy szoftver, mely rendelkezett a szükséges funkcionalitással, viszont a SMOG-1-es csapatnak a régi szoftver fejlesztői nem tagjai, így csupán a többiek tapasztalatai, tanácsai és elvárásai maradtak meg belőle. A program tehát képes volt felhasználóbarát módon prezentálni a vett adatokat, lehetett vele a műholdat is követni.

Rádióamatőrök által műholdkövetésre előszeretettel használt program a nyílt forráskódú Gpredict[13]. Ez képes a műhold pályaadatainak kiszámolására, a frekvenciák meghatározására, a pálya grafikus szemléltetésére és még számos egyéb dologra. Egy jól összerakott program, de az igényeinkhez képest túlzottan összetett, mi csak a nyers adatokra vagyunk kíváncsiak. Több munkával járna ezeket valamilyen módon kivezetni a programból, mint egy másik, egyszerűbb programot átalakítani.



1.1. ábra. *Gpredict használat közben*

Egy lépéssel közelebb kerülünk a nyers adatokhoz, ha a PREDICT[14] nevű programot nézzük. Ez nem egy mai fejlesztés, cserébe viszont tud mindent, amire szükségünk van a követéshez, elérhető a forráskódja és lightweight is. Egyszerű parancssoros felülettel rendelkezik, sőt legújabb verziója beszédszintetizátor segítségével képes minket hanggal is értesíteni a fontosabb eseményekről. Mivel open source, így ideális kiindulópont volt az én programomhoz.



1.2. ábra. *PREDICT az OSCAR-27 követése közben[14]*

Az E épület tetején lévő állomás[15] a PREDICT egy némileg módosított verzióját szokta futtatni, mely a pályaadatokat szöveges fájlba írja - ebbből egy másik program olvas másodpercenként. Ez a program a fájlból olvasott információk alapján vezérli az antenna forgatóját, szükség esetén még egy speciális manővert is végrehajt, melyet majd később ismertetek, az antenna forgató vezérléssel foglalkozó fejezetben. A többiek tapasztalata és javaslatai alapján tehát a PREDICT-et választottam kiindulási alapnak a feladathoz, azt módosítottam és láttam el további funkciókkal, ahogyan azt be is fogom mutatni hamarosan.

A hardware eszközök vezérlése saját fejezete(ke)t érdemel, ahol bemutatom őket, a velük járó kihívásokat és az elkészült megoldást is. Ezen vezérlési feladatok megoldása elengedhetetlen, hiszen a rádióamatőrök egy számottevő része automatizált követő állomást üzemeltet, ami felügyelet nélkül végzi ezeket. Természetesen ők általában már rendelkeznek kedvenc szoftverrel a követéshez, viszont ez a szoftver egy frissen készülő állomással jól párosítható, továbbá az elsődleges és másodlagos SMOG-1 követő és vezérlő állomáshoz is szükség van rá.

2. fejezet

Qt

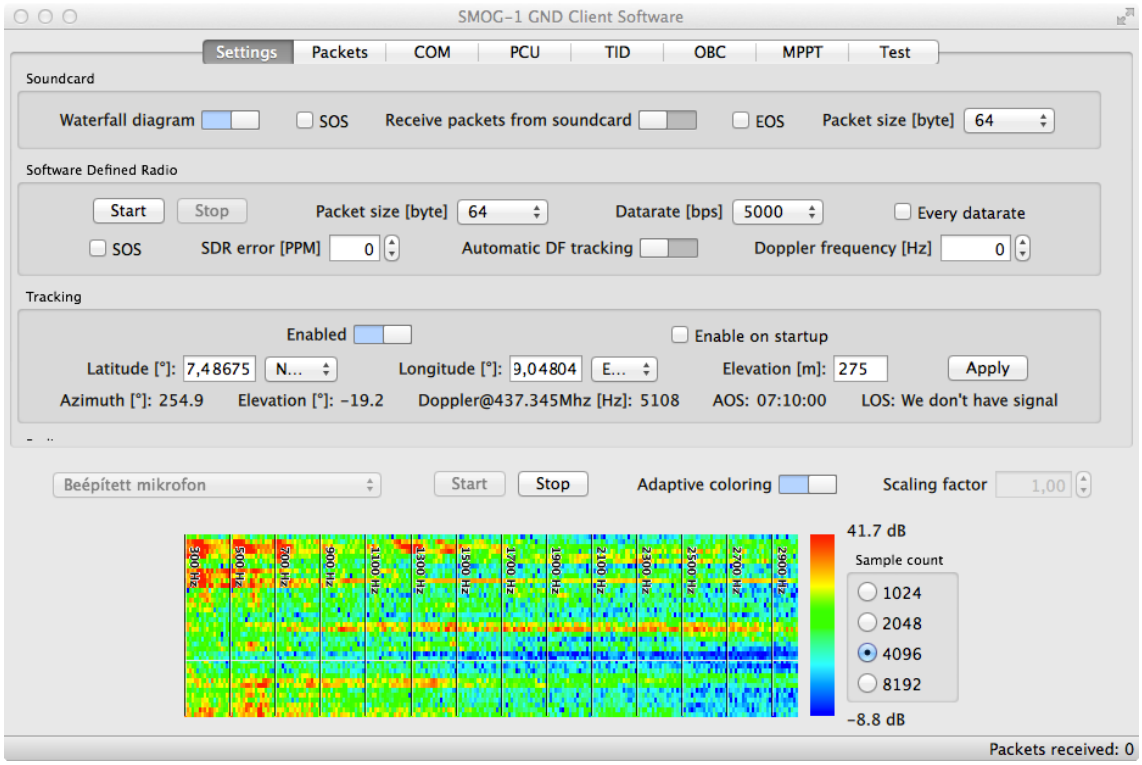
A Qt[16] egy kiforrott, ugyanakkor aktív fejlesztés alatt álló keretrendszer C++-hoz. Ebben a fejezetben a keretrendszer számomra legfontosabb jellemzőit és lehetőségeit ismertetem, kiemelt figyelmet fordítva azokra a lehetőségekre, melyek ideálissá teszik a feladathoz. Ide sorolható például a platformfüggetlenség, a C++ nyelv használatának opciója, a különböző hardverek és portok vezérlése, az ingyenes felhasználhatóság, a felhasználói felület gyors létrehozhatósága és végül, de nem utolsósorban a signal slot mechanizmus.

2.1. Platformfüggetlen

A feladatom egy platformfüggetlen szoftver elkészítése volt. Mac OS X-en, Windows-on és kiválasztott Linux disztribúciókon kell majd működnie, hogy minél több rádióamatőr használhassa különösebb probléma nélkül. A tervezési fázisban természetesen szóba jött a Java használata is. A Masat-1-hez készített programot Java-ban írták, viszont a SMOG-1 fejlesztői csapatnak sok rossz tapasztalata volt a nyelv platformfüggetlenségével kapcsolatosan, és én sem ragaszkodtam hozzá.

A Qt rengeteg kész modullal rendelkezik, melyek elfedik a fejlesztők elől az egyes platformok közti különbségeket, egységes interfészt biztosítva. Legyen szó hangeszközök, vagy soros portok kezeléséről, hálózati műveletek lebonyolításáról vagy akár az óra eléréséről. Mindhárom fő platformon elérhető hozzá fejlesztői környezet és egy jól konfigurált project ugyanúgy fordítható és futtathatható Linux-on, mint Windows-on. Ez könnyíti a verziókövetést is - a műhold egyes tesztjeinél jól jött, hogy bár én Windows-on fejlesztettem, a többiek Linux-on is mindig gond nélkül elérhették a legújabb változtatot.

A fejlesztés során két esetben kellett külön figyelmet szentelnem az egyes platformoknak. Ügyelnem kellett arra, hogy a felhasznált külső library-k elérhetőek legyenek a választott platformon, továbbá egy hardverrel kapcsolatos, felhasználói élményt javító feature elkészítéséhez szükség volt némi extra munkára.



2.1. ábra. *Beállítások oldala MAC OS X-en*

2.2. C++ nyelv használata

A műholdfejlesztő csapat túlnyomórészt villamosmérnökökből áll és többségük számára a legkényelmesebb az, ha C-ben programozhatnak. A C a villamoskar hallgatói számára egy közös nevező. Hatalmas előnyt jelent, ha fejlesztés közben felhasználatok mások által megírt komponenseket anélkül, hogy működésük minden részét érteném. Például a jel demodulálásáért felelős kódot C-ben írta a konzulensem, Dudás Levente, én pedig egy az egyben fel tudtam ezt használni a készülő szoftverhez, minimális átalakítás után.

A program elkészítéséhez szükséges külső library-k is általában C-ben készültek, ezért ezeket is használhattam. Ezzel nem csak időt spóroltan meg, hanem a sok év alatt, alaposan bejáratott megoldásokra is nyugodtan támaszkodhattam, hiszen mára bizonyítottan jól működnek.

2.3. Hardverek és portok kezelése

Ahogy azt korábban is említettem, az implementációs részletek jó részét elfedi előlünk a Qt, mely könnyen használható wrapper osztályokkal rendelkezik. Például egy platformfüggetlen, soros porton kommunikáló szoftvert nagyjából egy óra alatt el lehet készíteni. Rövid konfiguráció után tetszőleges adatot ki tudunk küldeni valamely porton.

Fontos volt az is, hogy hangeszközöket kényelmesen tudjunk kezelni, a rádió kimenetét a gépbe vezetve képesek legyünk a műholdtól venni. Ez sem jelentett nagy kihívást, a `Multimedia` modul sok kész megoldással rendelkezik, és általánosan kijelenthető, hogy a Qt-hez részletes dokumentáció[17] tartozik.

2.4. Ingyenes felhasználhatóság

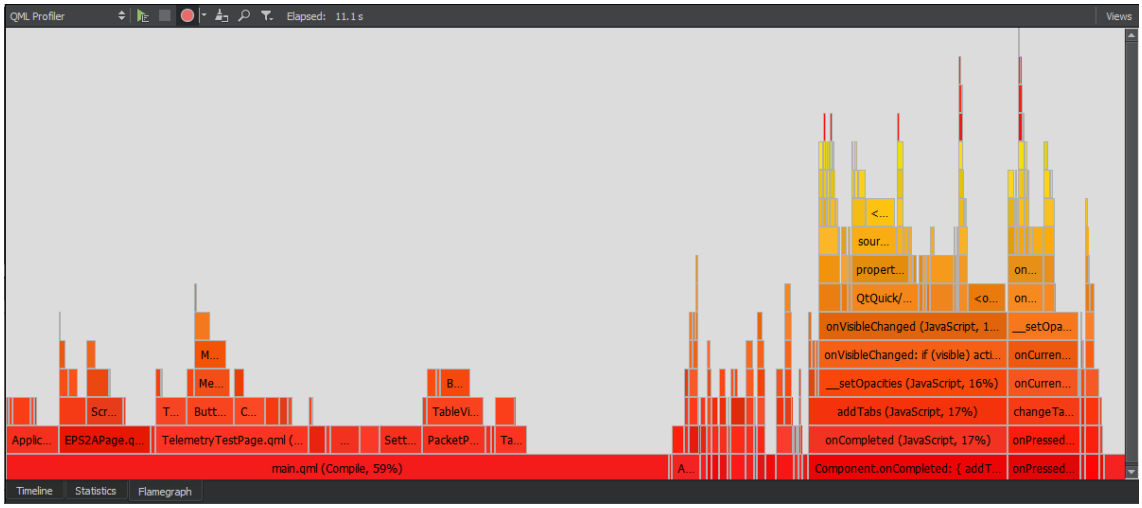
A műhold azt szem előtt tartva készül, hogy sikeres pályára állását követően (majdnem) minden szoftver és dokumentáció szabadon hozzáférhető lesz. Ebből következik, hogy az általam készített szoftvernek is teljes mértékben szabadon felhasználható komponensekből kell állnia. Ez érinti a felhasználható külső library-ket is, de érinti azt is, hogy milyen eszközökkel dolgozhatunk. A csapatból senki nem használ olyan szoftvert, mely pénzbe kerülne, ezzel is bizonyítván, hogy így is meg lehet csinálni. A Qt ebbe a környezetbe kiválóan beleillik, bizonyos feltételek mellett szabadon felhasználható, és szükség esetén a forráskódja is elérhető.

2.5. Felhasználói felület

A Qt keretrendszer részét alkotja a QML[18] nevű nyelv is. A QML-lel ismerkedve könnyen eszünkbe juthat mind a XAML, mind a JSON, mind pedig a JavaScript. Ezutóbbi igen fontos szerepet tölt be, lehetőségünk van JavaScript kódrészleteket használni QML-ben, így egész bonyolult logikával is elláthatjuk felhasználói felületünket. Érdemes viszont óvatosan gazdálkodni a JavaScript kóddal, mivel gyakran sokkal jobb teljesítményt érhetünk el, ha C++-ban futtatjuk a logikát megvalósító részeket.

Rendelkezésünkre állnak a `Qt Quick`, `a Qt Quick Controls`, `Qt Quick Dialogs` és `Qt Quick Layouts` modulok is, melyekkel néhány perc alatt egész igényesen kinéző felületet tudunk összerakni. Fontos előny az is, hogy az egyes platformokon beleillenek a rendszer arculatába, ugyanakkor tetszőleges módon testre is szabhatjuk a vezérlőket, ha erre van igényünk.

További pozitívum, hogy az ingyenesen használható Qt Creator IDE-ben van beépített QML profiler, melyben többféle nézet segítségével könnyen azonosíthatunk hotspotokat, melyekkel adott esetben érdemes lehet a továbbiakban foglalkozni. Előfordulhat, hogy rátalálunk egy olyan részre, mely logikát JavaScript helyett C++-ban megvalósítva számottevő gyorsulást érhetünk el.



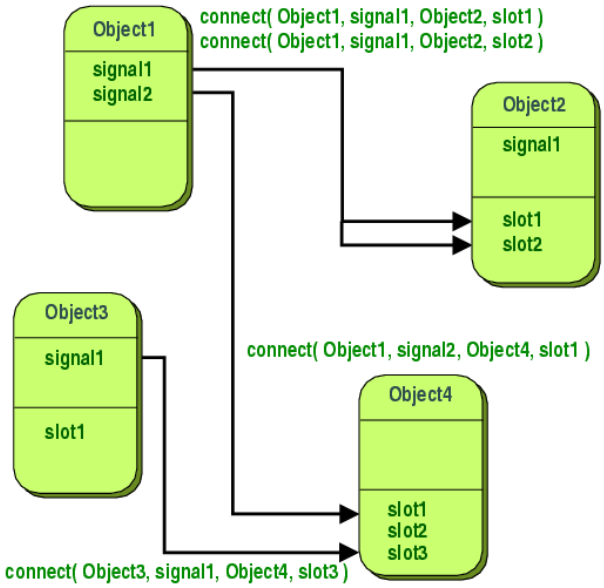
2.2. ábra. *QML Profiler Flamegraph nézete*

A C++-ban lévő modellünket is könnyen össze tudjuk kötni a QML-ben leírt felületünkkel a property rendszer segítségével, továbbá tudunk függvényt hívni, adatot olvasni és állítani is. Természetesen adatkötést QML-en belül is létre tudunk hozni és a signal slot mechanizmust is használhatjuk erre.

2.6. Signal slot mechanizmus

Felhasználói felületekkel foglalkozva gyakran felmerül az igény, hogy egy vezérlő megváltoztatása további komponensekre is hatással legyen, azt szeretnénk, hogy bizonyos objektumok tudjanak egymással kommunikálni. Erre megoldás például, ha callback-eket használunk. Egy esemény bekövetkezésekor az objektum meghívja a rá feliratkozott callback függvényeket, így azokon keresztül értesülünk az esemény bekövetkezéséről. Ez működhet is jól, viszont rendkívül körültekintően kell eljárnunk a tervezés szakaszában és mindig következetesnek kell lennünk, a függvény pointerek használata odafigyelést igényel.

Qt-ben van egy alternatíva a callback-ek használatára, a signal slot mechanizmus[19]. Egy objektum signal-t bocsájthat ki tetszőleges pillanatban, például egy esemény bekövetkezésekor. A signal-okat (jellemzően, de nem feltétlenül) más objektumok slot-jaihoz köthetjük. Ha egy slot-ba signal érkezik, akkor gyakorlatilag egy függvény kerül meghívásra, ahogy azt callback-eknél is csinálnánk. Nagy előny azonban, hogy a signal slot mechanizmus garantáltan type safe, csatlakozásnál a compiler is jelezni tudja nekünk, ha elrontottunk valamit.



2.3. ábra. *Signal slot mechanizmus[19]*

Egy signal tetszőleges slot-hoz lehet csatlakoztatva és egy slot-hoz tetszőleges helyről futhat be signal. Ez lehetővé teszi azt is, hogy egy objektum ne tudjon semmit sem arról, hogy mi csatlakozik hozzá vagy az mihez csatlakozik, a felelősségek nagyon tiszták tudnak lenni.

Az összeköttetés signal és slot között lehet `DirectConnection, QueuedConnection` és

`BlockingQueuedConnection`. `DirectConnection` esetében a végrehajtás akkor folytatódik a signal-t kibocsájtó objektumban, ha minden slot feldolgozása véget ért, ilyenkor a slot-ok végrehajtása a signal-t kibocsájtó szálon történik. Ez gyakran használható jól, ha event loop-tól függetlenül akarunk valamit végrehajtani. `QueuedConnection` használatakor a végrehajtás egyből folytatódik, nem várjuk meg a feldolgozást, hiszen az időben akár jóval később is bekövetkezhet. Az egyes slot-ok akkor kerülnek végrehajtásra, ha az event loop elér hozzájuk. Logikus az is, hogy ilyen esetben a slot végrehajtása azon a szálon történik, ahol a signal-t fogadó objektum van. A kettő egyfajta keveréke a `BlockingQueuedConnection`, melynél a slot akkor kerül végrehajtásra, ha az event loop elér hozzá, viszont addig a signal-t kibocsájtó szál blokkolva várakozik, míg a slot feldolgozásra nem kerül. Ekkor persze muszáj arra figyelni, hogy a signal-t kibocsájtó és slot-tal rendelkező objektum ne ugyanazon a szálon legyenek, hiszen ekkor deadlock alakulna ki.

A signal slot mechanizmus szerintem az egyik legszebb koncepció a Qt-ben. Rendkívül tisztán tudunk vele observer mintát megvalósítani, akár több szálon is. Ezt számos helyen fel lehet használni, küldhetünk például a felhasználói felületről és a felhasználói felületnek is üzenetet, adatot. A C++-ban írt kódon belül is hasznos tud lenni, ha például egyik objektum több másiknak szeretne adatot küldeni, vagy aszinkron módon szeretne működést kiváltani, ami nem ritka, ha hardverről van szó. Ezen felül pedig tényleg úgy gondolom, hogy a működés tisztasága és átláthatósága lehetővé teszi, hogy egymástól jól elkülönülő szoftver komponenseket készíthessünk, még akkor is, ha ebben a konkrét project-ben ez nem volt elsődleges fejlesztési szempont.

3. fejezet

Műholdkövetés TLE alapján

Ahogy azt korábban is említettem, a fejlesztés alatt álló SMOG-1 műholdat aktívan követni kell, hiszen nem geostacionárius pályára fog állni. Ahhoz, hogy pontosan előre tudjuk jelezni a műhold helyzetét szükségünk van valamilyen forrásból friss és feldolgozható adatokra az aktuális pályáról. Ebben a fejezetben ismertetem az adatok forrását, a feldolgozás módját és az implementációt is.

3.1. TLE - Two-Line Element Set

A NORAD (*North American Aerospace Defense Command*) érthető okokból folyamatosan figyeli az űrben lévő tárgyakat. Szerencsére a titkosnak nem minősülő objektumok pályaadatait 12-24 órás gyakorisággal szolgáltatja a CelesTrak-nek[20], ahol bárki által elérhető módon letölthető TLE formátumban[21].

A TLE a Föld körül keringő objektumok követéséhez használt de facto szabvány. A formátum lényege, hogy két sorban tartalmaz minden adatot, ami alapján megfelelő modell felhasználásával képesek vagyunk egy adott pillanatban az objektum helyzetét meghatározni. Tartalmaz még egyéb, kiegészítő adatokat is, mint például az objektum egyedi azonosítója és ellenőrző összegek. Tehát meg lehet határozni, hogy egy adott pillanatban hol van az objektum, mekkora frekvenciával szükséges korrigálnunk a Doppler-hatás miatt és még sok, számunkra kevésbé fontos paramétert. Ha azt is tudjuk, hogy mi hol vagyunk, akkor már azt is el tudjuk dönteni, hogy van-e rálátásunk a műholdra, és ha igen, akkor milyen azimut és eleváció alatt.

Ez persze nem kis kihívás, komoly tudományos háttérrel rendelkezik[22] a módszer, én nem is készítettem teljesen saját implementációt a feladathoz, hanem a már említett PREDICT-et alakítottam át.

3.2. A PREDICT átalakítása

A PREDICT jóval többre képes, mint amire nekünk szükségünk van, ezért első lépésben azt kellett kiderítenem, hogy pontosan mi is kell a forráskódból. Ehhez adott némi támpontot a csapat által használt PREDICT „változat”. Megnéztem, hogy hol és mit ír fájlba és innen elkezdtem visszafejteni.

Elsőként megnéztem, hogy a kiszemelt függvény mely más függvényeket hív meg, „rekurzívan". Nyilvánvaló módon csak és kizárólag olyan függvényekre van szükségem, melyek szerepelnek az így kialakult listában. Miután ezeket kiválogattam, előállt az a kódrészlet, mely szolgáltatja majd számunkra a szükséges adatokat, ha helyesen tápláljuk bemenő adattal. Érdekességként megemlítem, hogy a körülbelül 6500 soros forrást ebben a lépésben redukáltam ~ 2400 sorra, melybe már a későbbi kiegészítéseimet is beleszámoltam.

Második lépésben a PREDICT Windows-hoz és Linux-hoz tartozó forráskódjait vetettem össze egy diff checker-rel, hogy azonosíthassam azokat a részeket, melyeken javítanom kell, ha platformfüggetlen kódot szeretnék. Az eltérések egy része abból adódott, hogy a két forráskód eltérő verziókhoz tartozott, ez nyilván elkerülhetetlen ilyen esetben. Az egyetlen dolog, ami az általam kiválasztott kódrészletet érintette, az aktuális dátum előállítása volt. A `QDateTime` nevű Qt osztály használatával ezt sikerült könnyen orvosolni.

Harmadik lépésben áttanulmányoztam a kódot és a kiválogatott függvényeket és adatszerkezeteket úgy módosítottam, hogy a program csak annyira legyen képes, amire mi szeretnénk: azimut, eleváció és doppler frekvencia meghatározása adott pillanatban, adott helyről. A PREDICT általánosabb működésre volt képes, több műholdat is tudott egyszerre követni, számos olyan adatot szolgáltatott, melyekre nekünk nincsen szükségünk.

Negyedik lépésben beállítottam néhány konstans értéket (pl. 437,345 MHz), melyek a műhold élete során nem fognak változni, ezért nincs értelme azokat konfigurálhatónak hagyni. A PREDICT-ből átvett kód ezen beállított frekvenciaérték alapján szolgáltat Doppler-korrekcióhoz szükséges értékeket.

Ötödik és utolsó lépésben az átalakított kódhoz készítettem egy header fájlt is, mely segítségével olyan interfészt alakítottam ki, ami a mi felhasználásunkhoz szükséges. Ezen a ponton néhány saját függvényt is írtam, melyek kényelmessé teszik a kód felhasználását a program más részeiből is. A következő részben ezt mutatom be.

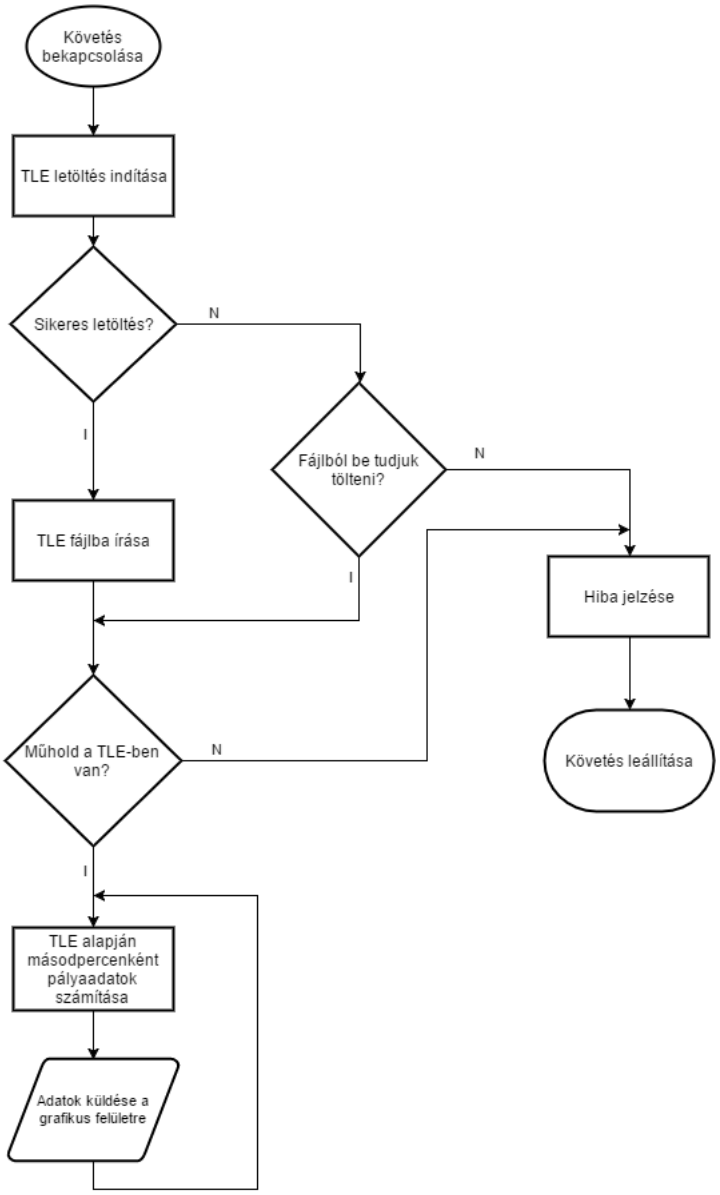
3.3. Az új kód és a program kapcsolata

Az elkészült kódot valamilyen módon a programomnak használnia is kell. Ehhez készítettem egy wrapper osztályt `Predicter` névvel, mely a `QObject` osztály leszármazottja, és mint ilyen, képes Qt specifikus dolgokra. Ebben a konkrét esetben a signal slot mechanizmust használtam fel. A signal slot mechanizmussal kommunikálok a felhasználói felület és a `Predicter` osztály között. Lehet állomáshelyszínt állítani, követést indítani és leállítani, továbbá egy speciális antenna manőverhez szükséges adatokat lekérni. Az osztály képes továbbá különféle signal-ok segítségével jelezni a grafikus felületnek, ha valamilyen problémába ütközik követés során.

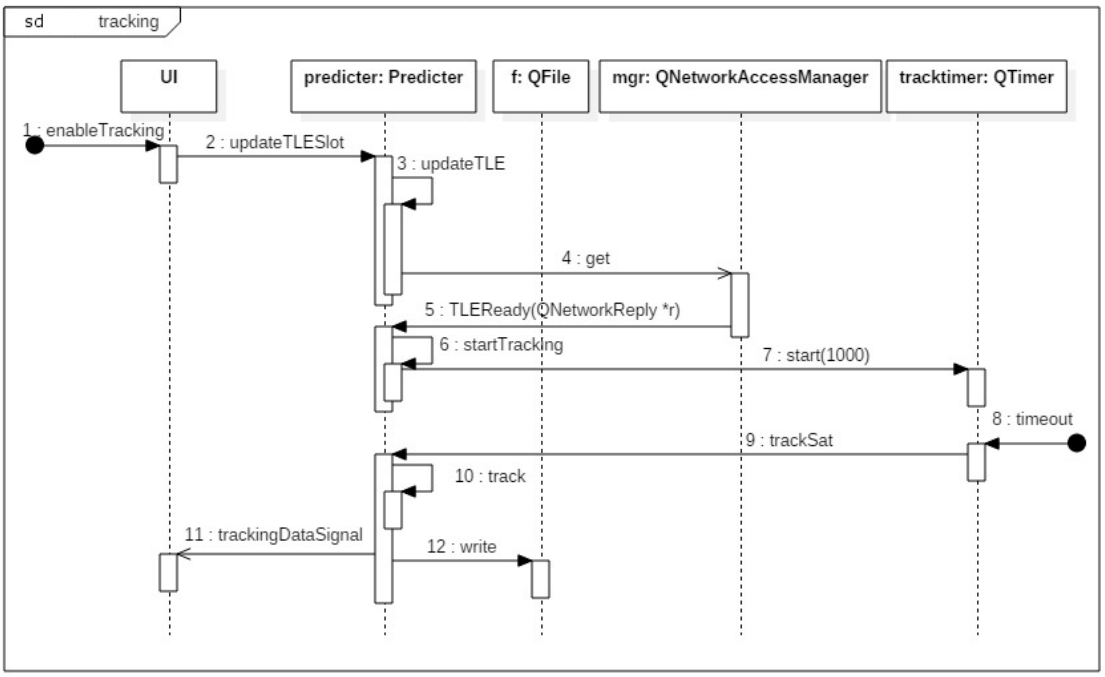
A követéshez szükség van a legfrissebb TLE adatokra, ezt a wrapper osztály tölti le a CelesTrak szerveréről. A letöltött fájlt elmenti az aktuális dátummal azonosítható módon, majd kikeresi belőle a műholdhoz tartozó két sort az azonosítója alapján. Ha nem sikerül a fájlt letölteni, akkor egy speciális elnevezésű fájlból próbál TLE-t olvasni. Ezt a fájlt minden sikeres letöltésnél felülírjuk, továbbá a felhasználó is kicserélheti kézzel a fájlt, ha valahonnan szerzett TLE adatokat a megfelelő formátumban. Az adatok újbóli letöltését

12 óránként megkíséreli folyamatos futás esetén, hiszen a program futása közben a NORAD közölhet frissebb számokat. Mind az adatok frissítése, mind a szerverről való letöltés signal slot alapon működik, hiszen alapvetően aszinkron műveletekről van szó. Az időzítéshez a `QTimer` osztály használható, a letöltéshez pedig a `QNetworkAccessManager`. Ezek jól megírt, jól dokumentált, platformfüggetlen osztályok a Qt keretrendszerben.

A követést tehát a felhasználó vezérli a grafikus felületről, ahol el tudja indítani, le tudja állítani, és tudja módosítani az állomás helyét is. Ezek után a kiszámolt követési adatok (azimut, eleváció, láthatóság, doppler frekvencia) a felhasználói felülethez kerülnek, ahol JavaScript-et használva ezekkel az adatokkal vezéreljük a hardvereket (forgatót, hardver- és szoftverrádiót), ha azok is be vannak kapcsolva.



3.1. ábra. *Folyamatábra a követésről*



3.2. ábra. *Szekvencia diagram a követés bekapcsolásának folyamatáról, ha elérhető a szerver*

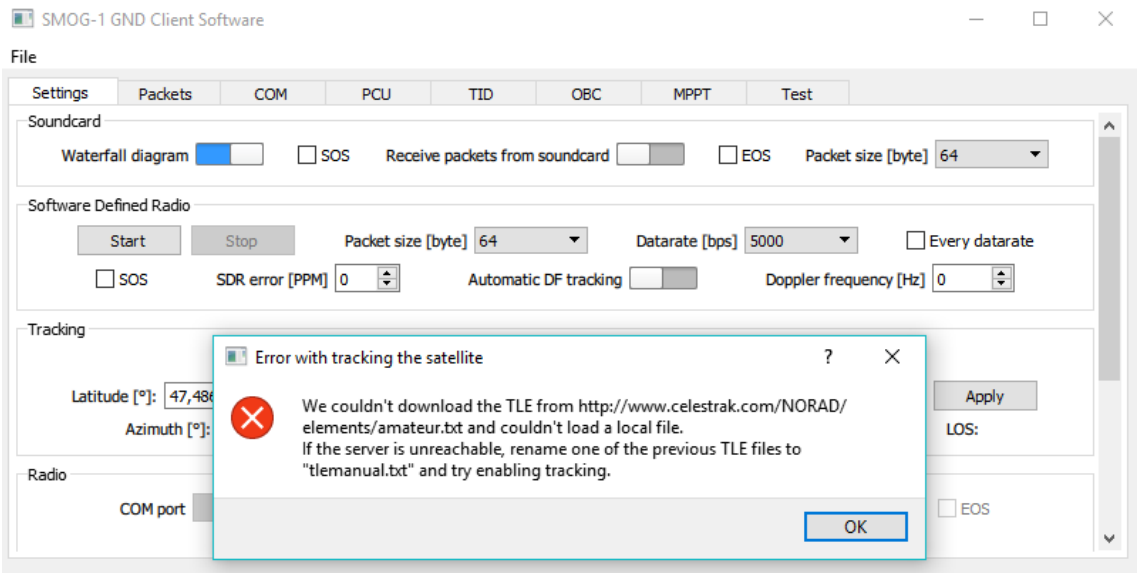
3.4. Tesztelés

A program készítése során, minden nagyobb funkció beépítése után tesztelés következett. Ezek eredményét az egyes fejezetekben ismertetni is fogom, hiszen valamilyen módon szeretném szemléltetni, hogy a meghozott döntések helyesek voltak, az elkészült kód teljesíti feladatát.

A műholdkövetés adja az alapját több másik funkciónak is (forgató vezérlés, rádió vezérlés, stb.), ezért bizonyítottan jól kell működnie. A probléma két részből áll - ahogy azt ismertettem -, a TLE adatok beszerzéséből és azok felhasználásából.

Az adatok letöltése lehet sikeres vagy sikertelen. Sikertelen próbálkozás esetén ésszerű keretek között a programnak kezelnie kell a problémát. Erre nem rossz megoldás az, hogy egy, a számítógépen lévő fájlt megnyit és megpróbálja abból kikeresni a szükséges két sort. Részben ez a működés is magyarázatot ad arra, hogy miért menti a wrapper osztály mindig fájlba az újonnan letöltött TLE pályaadatokat. Így a TLE host kiesése esetén is lesz egy fájl a gépen, aminek segítségével, nem tökéletes pontossággal, de elég jól képesek leszünk követni a műholdat. Ha a fájlból betöltés sem sikerül, akkor a program jelzi a felhasználónak, hogy olyan problémába ütközött, amit magától nem tud megoldani, felhasználói beavatkozás szükséges.

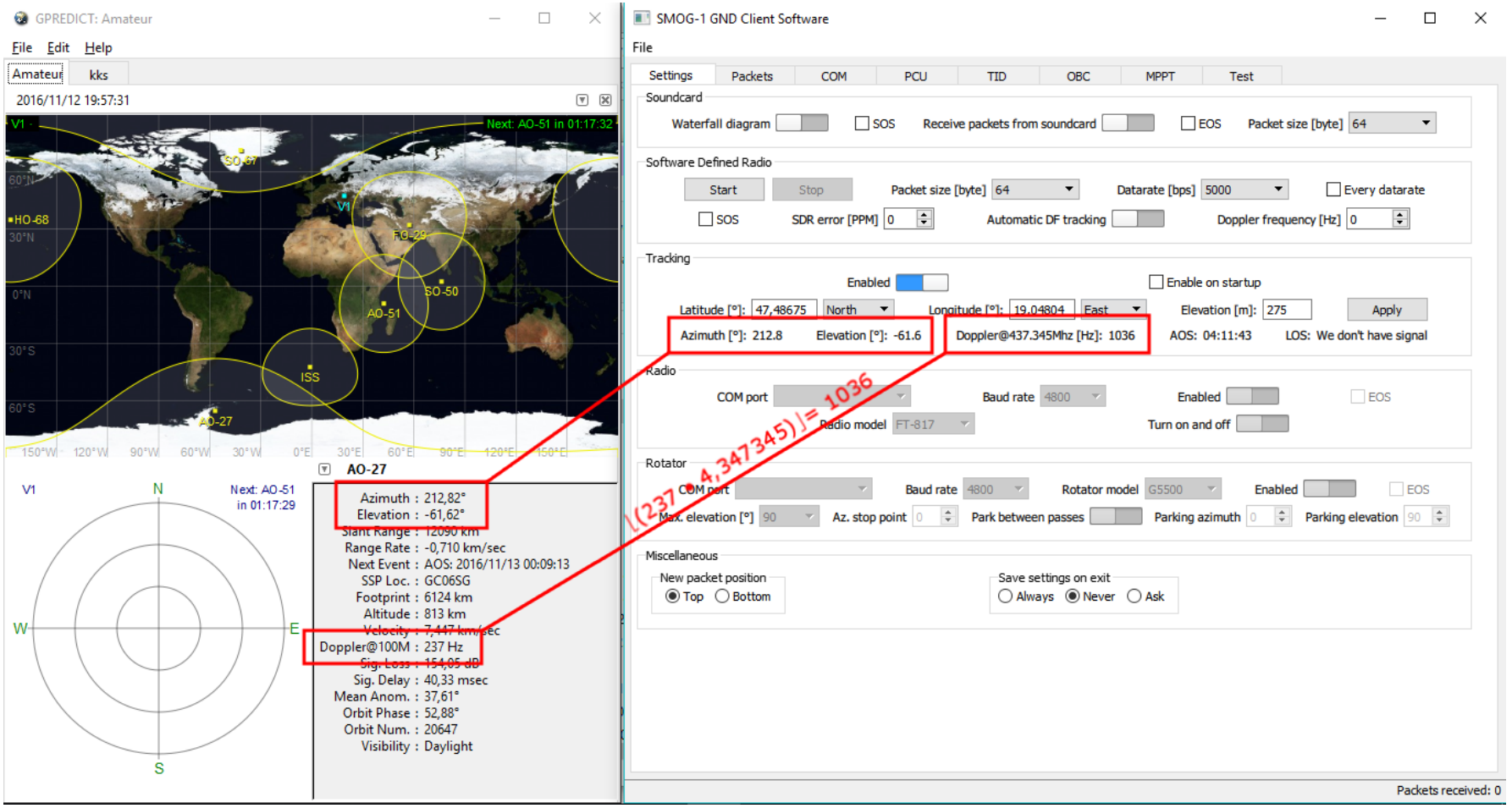
A sikertelen letöltést két módon szimuláltam, először internetkapcsolat nélkül indítottam követést, másodszor pedig a letöltendő fájl nevét átírtam, hogy ne sikerüljön annak letöltése a szerverről. Mindkét esetben képes volt a program fájlból olvasni. Amikor a fájl letöltése és a számítógépen lévő fájl betöltése is sikertelen volt, a program ezt jelezte nekem, ahogyan ez el is várható tőle.



3.3. ábra. *Hiba tracking indításakor*

Az adatok sikeres betöltése után azt kell ellenőriznünk, hogy valóban helyesen számolunk-e pályaadatokat és helyesen írjuk-e azokat ki egy olyan fájlba, melyet egy felhasználó tetszőlegesen tud majd használni.

A pályaadatok ellenőrzéséhez a kapott számokat a Gpredicttel hasonlítottam össze, úgy, hogy az AO-27-es[23] műholdat követtem mindkét programmal, azonos helyszínről. Mint az alábbi képen is látható, az általam előállított értékek megyegyeznek a Gpredict-ben szereplőekkel. A fájlba írás is jól működik, másodperces gyakorisággal frissül követés során.



3.4. ábra. *Pályaadatok a Gpredict és a programom számításai alapján*

4. fejezet

Vezérlés soros porton keresztül

Ha már ki tudjuk számítani a pályaadatokat, akkor következő lépésben ezeket el kell küldenünk a különböző hardvereszközöknek. Ez a fejezet a hardverrádiók és antenna forgató vezérlésével foglalkozik, a szoftverrádió (*SDR*) egy saját fejezetet kap.

Ha egy felhasználó nem elégedett a programom által nyújtott lehetőségekkel, akkor vagy implementál a saját eszközéhez egy osztályt és azt használja (ez nem túl nehéz), vagy a másodpercenként számítógépre írt fájlt használja, mely tartalmazza a szükséges pályaadatokat.

4.1. Soros port kezelése Qt keretrendszerrel

Mint sok gyakran előforduló use case-re, a soros port vezérlésére is van beépített modul (`serialport`) és két osztály a keretrendszerben. Ezekkel le tudunk kérni az éppen csatlakoztatott portokról információkat és egy új kapcsolatot is nyithatunk bármelyiken keresztül, mindezt platformtól függetlenül.

Ami mégis némi platformspecifikus kódot igényelt, az a hot plugging lehetővé tétele volt, de ez csupán kényelmi funkció a felhasználóknak, nem feltétlenül lett volna szükséges megvalósítani.

QML-ből hívható függvények segítségével vezérlem a rádiókat és a forgatót, melyek ha valamilyen problémába ütköznek, akkor azt signal kiadásával tudatják a program többi részével, jellemzően egy ablak jelenik meg a felhasználói felületen. Az eszközkezelő osztályok rendelkeznek egy `SerialPortHandler` objektummal, mely a tényleges soros porti kommunikációt végzi. Rendelkeznek egy `SerialPortSettings` objektummal is, mely a kiépített vagy kiépítendő kapcsolat paramétereit tartja számon. Ezeknek az objektumoknak küldi el az eszközkezelő objektum a soros portra kiküldendő adatot és a módosítandó beállításokat.

4.1.1. Eszközök felderítése

A Qt könnyen kezelhető osztályt biztosít számunkra, ha soros portokat, vagy - mint a későbbiekben látni fogjuk - audioeszközöket akarunk felderíteni. A `QSerialPortInfo` és a `QAudioDeviceInfo` osztályok egy-egy statikus publikus függvénye szolgáltat nekünk listát az elérhető eszközökről, és azok adatairól, mellyel megvizsgálhatjuk az eszközöket és

csatlakozhatunk is hozzájuk.

4.1.2. Hot plugging

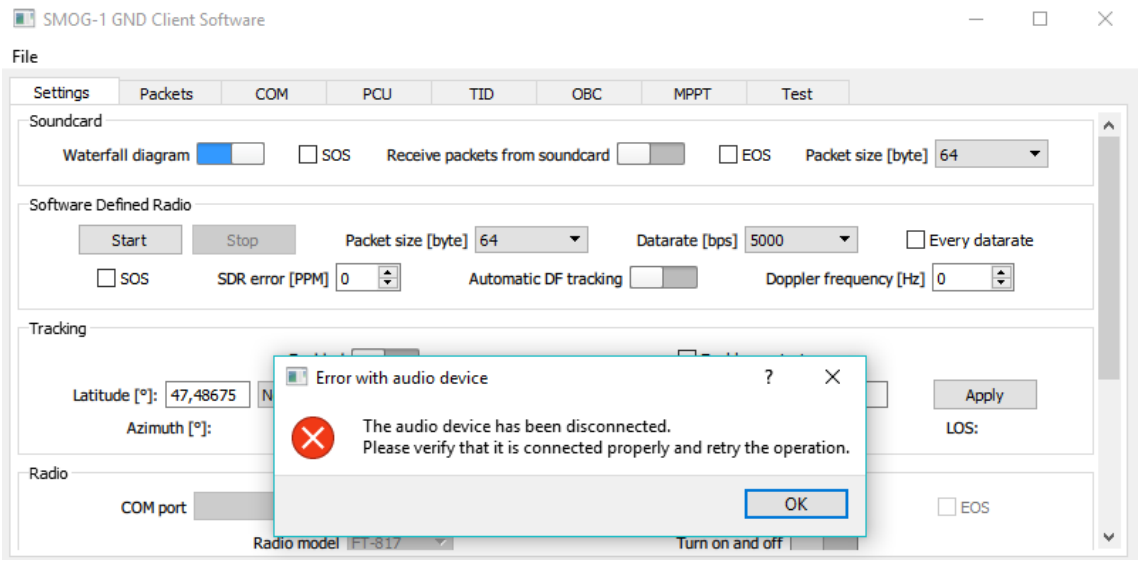
Elképzelhető az is, hogy a program futása közben megváltoznak az elérhető soros portok (hiszen USB soros port is létezik, sőt sokan használják) vagy az audioeszközök (pl. USB hangkártya). Ilyen esetben, ha nem foglalkozunk külön a kérdéskörrel, akkor azok csatlakoztatásnál azzal szembesülhetünk, hogy nem jelenik meg az eszközök listáján az új eszköz a programon belül. Ez még csak-csak elfogadható lenne, de lecsatlakozásnál akár valamilyen hibával le is állhatna a program. Mindenki jobban jár, ha ezekre a jelenségekre előre felkészülök és a program kezeli őket.

Használatban lévő eszköz lecsatlakoztatásánál valamilyen kivétel vagy hibajelenség keletkezik, melyet le kell kezelni. Ezt úgy kommunikálja a program a felhasználóval, hogy feldobja neki egy ablakban, hogy az eszköz elérhetetlenné vált, ellenőrizze a működését, és indítsa újra a műveletet.

A használaton kívüli eszköz lecsatlakoztatása, vagy eszköz csatlakoztatása máshogy működik. Ilyenkor érthető módon a program nem tud magától jelezni semmilyen módon, hiszen alapesetben nem figyeli aktívan a rendszerhez csatlakoztatott eszközöket. Ezt orvosoltuk Windows és Linux operációs rendszereknél egy-egy platformspecifikus megoldással, míg Mac OS X-nél egy `QTimer` néz rá időnként az eszközökre, hogy történt-e változás.

A Linux-os implementációt Kristóf Timur készítette, aki a műholdas csapaton belül a fedélzeti számítógépet tervezi és programozza, ezen kívül pedig tapasztalt Qt fejlesztésben is. A megvalósításhoz egy `udev monitor`-t használt, melyet egy `QSocketNotifier` objektum figyel. Ha a bekövetkezett esemény új eszköz bekerülése vagy eszköz eltávolítása, akkor egy signal jelez az eszközfelderítő osztálynak, ami ennek hatására poll-olja az eszközöket és frissíti a listát.

Windows-ra én készítettem el ezt a fajta logikát, a Qt jó eszközöket biztosított ehhez is. Lehetőségünk van eseményfiltert illeszteni a programunkhoz, mely képes az operációs rendszertől megkapott események feldolgozására. Ezek után már nem volt más dolgom, mint kikeresni, kitapasztalni, hogy milyen eseményeket kap a program eszköz csatlakoztatásánál vagy leválasztásánál Windows-on. A `VM_DEVICECHANGE`[24] üzenet pont ezt jelzi, így elég az ilyen üzenetekrefigyelni és érkezésük esetén signal-lal jelezni a korábban már említett eszközfelderítő osztálynak.

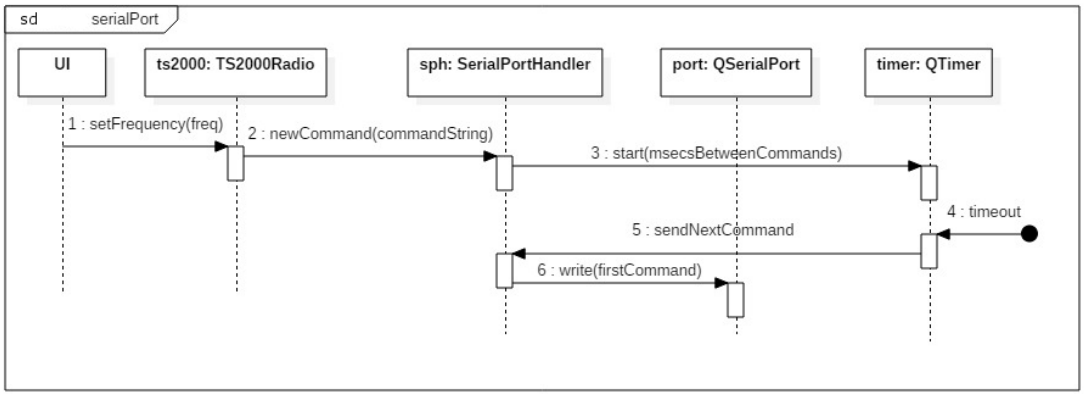


4.1. ábra. *Audio device lecsatlakozása használat közben*

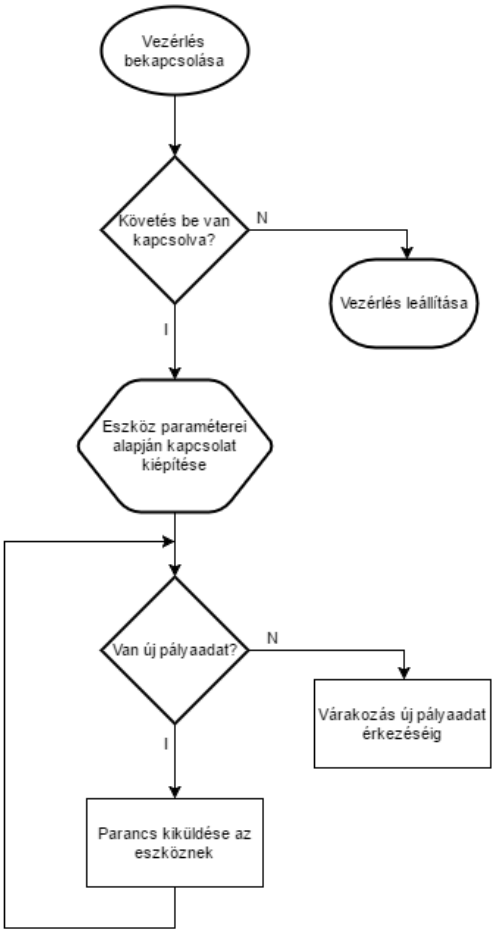
4.2. Hardverrádiók vezérlése

Ahhoz, hogy felhasználható jelet tudjunk venni, szükségünk van Doppler-korrekcióra. Ennek az az oka, hogy a demodulátor is csak véges határokon belül képes a névleges vivőfrekvenciától való eltérést kezelni. Ebből kifolyólag muszáj a vételre használt eszközt a pályaadatok alapján folyamatosan állítani, és itt jön képbe a rádiók vezérlése.

Két, rádióamatőrök által előszeretettel használt eszköz vezérlését készítettem el, a Yeasu FT-817-ét[25] (mely interfésze szerencsére egyezik az FT-897-évelel[26] is) és a Kenwood TS-2000-ét[27]. Fő feladatom tehát a frekvencia állítása. Az ehhez szükséges parancsot mindkét esetben a felhasználói kézikönyvek[28][29] alapján készítettem el. A TS-2000 vezérlésében semmi különleges nincs, így azt nem is részletezem.



4.2. ábra. *Szekvencia diagram frekvenciaállításról soros porton keresztül*

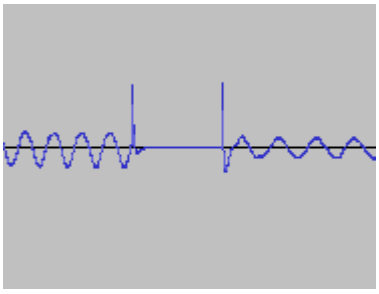


4.3. ábra. *Folyamatábra a soros porti vezérlésről*

4.2.1. Yaesu FT-817

Az FT-817 képes be- és kikapcsolásra, soros portról vezérelve amennyiben megfelelő áramellátás alatt áll. Ezt nem túl bonyolult elérni, csupán jól meghatározott üzeneteket kell neki kiküldeni, ha a műhold hamarosan felbukkan a horizonton, vagy véget ért az áthaladás.

Az FT-817 készítése során elkövettek egy nem dokumentált hibát, mely megnehezíti az eszköz használatát. Soros porton keresztüli frekvenciaállításnál a jel nem folytonos, akkor sem, ha tetszőlegesen kicsit változik a frekvencia. Az így kapott jelet nem vagyunk képesek jól demodulálni, információvesztéssel jár, ezért körültekintően kell eljárni a rádió vezérlésénél.

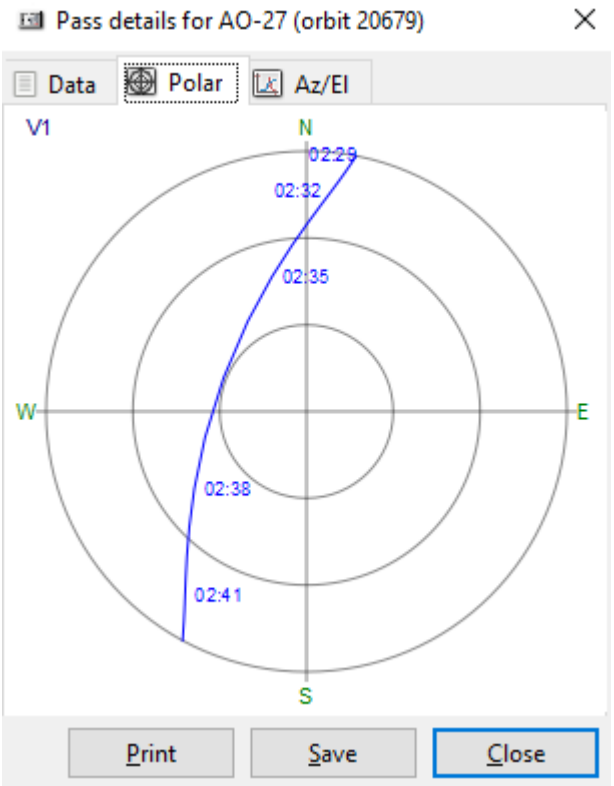


4.4. ábra. *Nem folytonos frekvencia váltási hiba FT-817 használata esetén*

Szerencsére két módon is állítható a frekvencia az FT-817-en. Egyrészt állíthatunk tetszőleges frekvenciát valamely VFO (*Variable-frequency oscillator*) beállításával, továbbá azt ± 10 kHz-es tartományon belül clarifier segítségével módosíthatjuk is. Clarifier használatával folytonos jelet kapunk, melyet már tudunk demodulálni. Így tehát az implementáció úgy működik, hogy ameddig csak lehet, clarifier segítségével állít frekvenciát. Amint kiér a clarifier működésének határára, új VFO frekvenciát állít úgy, hogy az új érték a clarifier tartomány szélére essen, hogy minél tovább tudjon majd clarifier segítségével módosítani. Bár ez nem tökéletes, a rádió sem az, és ez a megoldás egész hatékonyan képes demodulálható jelet szolgáltatni a demodulátor számára.

4.3. Forgató vezérlése

Forgatáshoz a Yaesu G-5500-as eszköz protokollját[30] használjuk. Nem túl bonyolult vezérelni, a rádiókhoz hasonlóan van ez is megvalósítva. Áthaladások között opcionálisan parkoló helyzetbe áll a forgató, majd a műhold közeledtével elkezd felé fordulni. Elvileg a forgató képes 360 fokban körbefordulni azimutban, de a gyakorlatban van egy pont (pl. a 0 azimut), amin nem tud átfordulni, ezért ha a műhold pályája során áthalad ezen a ponton, akkor a forgató tesz egy teljes fordulatot, hogy ismét követni tudja.



4.5. ábra. *A műhold pályája áthalad az északi, 0 azimut ponton*

Ez a csapat számára eddig is egy kezelendő probléma volt, melyhez már kidolgoztak egy megoldást.

4.3.1. Speciális manőver

Az algoritmus igen egyszerű. Csupán annyit kell tennünk, hogy ha a következő áthaladás ezt a pontot érinti, akkor 180 fokban megdöntjük az antennát elevációban, és *fordítva* vezéreljük a forgatót. Így tehát egy 0-s azimuton áthaladó pályából 180-as azimuton áthaladó pálya lesz, és ha a forgató képes 180 fokos elevációra, akkor a követés nem okoz problémát.

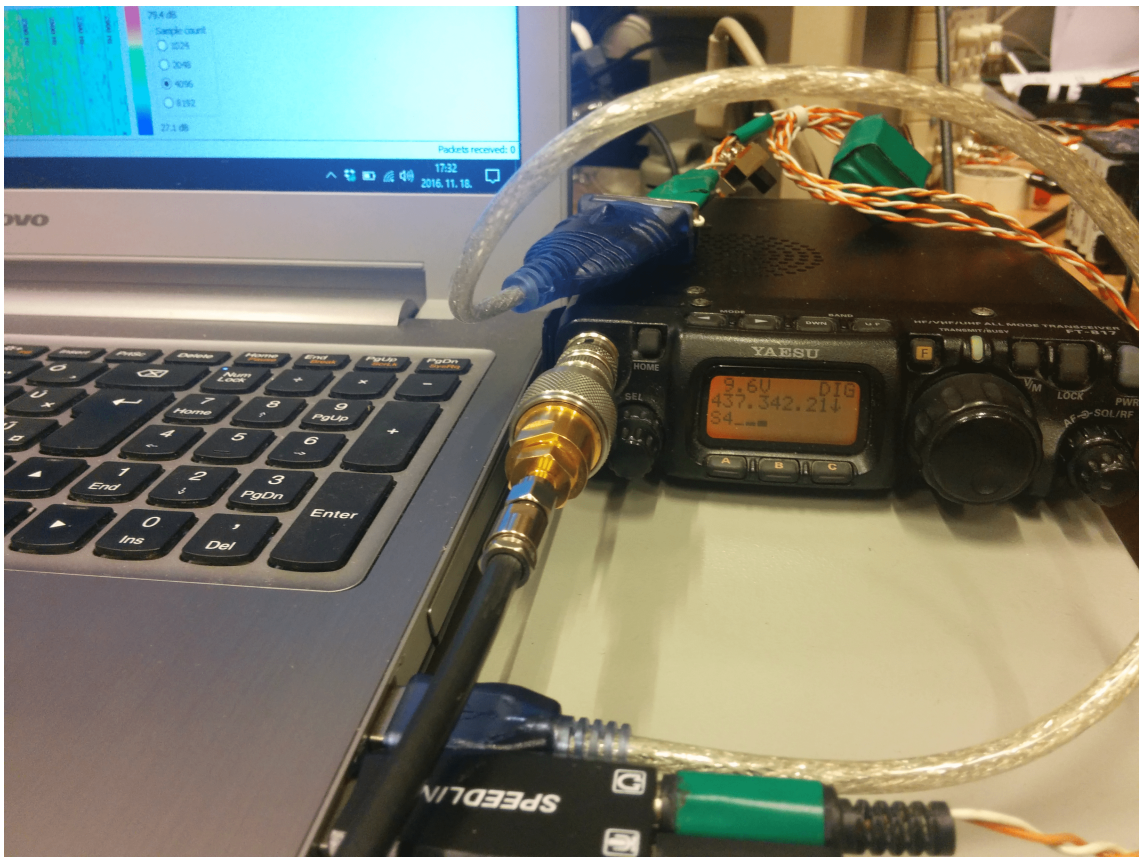
A következő pálya azimut és eleváció pontjait pedig (hiszen ezekre szükség van a manőver szükségességének eldöntéséhez) a `Predicter` osztálytól, QML-en keresztül kapja meg a forgatót vezérlő objektum.

4.4. Tesztelés

A teszteléshez több eszközre is szükségem volt, attól függően, hogy rádiót vagy forgatót akartam tesztelni.

4.4.1. Rádiók

Egy tetszőleges műhold azonosítóját a programba táplálva az képes Doppler-korrekciót számolni a beállított (437,345 MHz) frekvenciához képest. Nem volt más dolgom, mint a rádiót a géphez csatlakoztatni - például USB-s soros porton keresztül -, és bekapcsolni a vezérlést. Az alábbi képen látható, ahogy éppen követés közben a rádión 437 342 210 Hz van beállítva (a rádiót 10 Hz-enként lehet állítani).



4.6. ábra. *Yaesu FT-817 rádió vezérlése soros porton keresztül*

4.4.2. Forgató

Ez a teszt már kicsit nehezebben megvalósítható, hiszen szükség van egy forgatóra, ami pedig csak az elsődleges és másodlagos vezérlő állomáson van. Az elsődleges állomás forgatója jelenleg nem üzemel rendesen, ezért a másodlagos, Érden lévő forgatót kellett használni. Ehhez megkértem Dudás Leventét, hogy ha lehetősége van rá, akkor kövessen végig egy áthaladást a forgatóval. Ő megerősített abban, hogy a specifikáció szerint működik a forgató vezérlése.

5. fejezet

Hangkártyás demoduláció, vízesés diagram

A hardverrádiókkal vett jel egyszerű audio inputként csatlakoztatható a számítógéphez. Ezt a jelet kell demodulálni és megjeleníteni. A hangkártyás vétel és a vízesés diagram megvalósításának alapját[31] Kristóf Timur készítette el 2014-ben, én ezt alakítottam át. A demodulálást végző kódot pedig Dudás Levente írta, hiszen egy ilyen jól működő (rendkívül rossz jel-zaj viszony és néhány százalékos frekvencia-beli eltérés mellett is kiválóan funkcionáló) demodulátor megírása igen nagy szakmai kihívás. A demodulátor működése nagy vonalakban megtalálható a Függelékben.

5.1. Hangeszköz kezelése

Qt-ben hangeszközt kezelni sem túl nehéz, a `multimedia` modul kész megoldásokkal rendelkezik. El tudjuk kérni a csatlakoztatott eszközök listáját (hasonlóan, mint soros port esetén) a `QAudioDeviceInfo` osztály segítségével, beállíthatjuk az eszköz paramétereit a `QAudioFormat` osztállyal, végül az eszközhöz is csatlakozhatunk a `QAudioInput` osztály példányosításával. Ezek után nincs más dolgunk, mint készíteni egy `QIODevice` implementációt, ami feldolgozza a vett adatot, és elindítani az eszközt.

A `QIODevice` implementációnk magától meghívja a megfelelő függvényét, ha olvasott adatot. Innentől tehát csak annyi a dolgunk, hogy az adatot feldolgozzuk úgy, ahogy szeretnénk.

5.2. Hangkártyás demoduláció

A hangeszköztől kapott bájtokat signal felhasználásával továbbítjuk egy - pontosabban kettő - feldolgozó szálnak, melyek a konfigurációjuk alapján képesek különböző bitrateeken demodulálni a jelet. Azért van két ilyen szál, mert egyszerre demodulál a program 500 és 1250 bps-en, hogy biztosan minden csomagot venni tudjunk és ne legyen gond abból, hogy nem azzal az adatsebességgel ad a műhold, mint amivel demodulálunk. Ez előfordulhat, hiszen a műhold által adott jel adatsebességét adaptívan állítjuk majd, attól függően, hogy milyen a link minősége.

5.2.1. Demoduláló kód átalakítása

Ahhoz, hogy a C-ben megírt kódot gond nélkül tudjam használni C++-ban, több szálról, némileg át kellett alakítanom. Az átalakítás fő mozzanata az volt, hogy a konfigurációhoz szükséges változókat kiszerveztem egy `struct`-ba, mellyel paraméterezem a függvényhívást. Így viszonylag könnyen el tudtam azt érni, hogy különböző beállításokkal is hívható legyen egyszerre a függvény, követhető logikával. Ugyanezt az átalakítást kellett elvégeznem a szoftverrádióhoz tartozó demodulátornál is.

5.3. Vízesés diagram

A vízesés diagram egy olyan diagram, melyen a vízszintes tengely mentén helyezkednek el a frekvenciák, a függőleges tengely mentén helyezkedik el az idő, az egyes pontok színe pedig korrelál a jel amplitúdójával.

A vízesés diagram gyakorlati haszna számunkra elsősorban az, hogy amikor a műhold pályára kerül, akkor egy ideig pontatlan a NORAD-tól kapott TLE, beletelik némi időbe, mire a pálya normalizálódik. Ahhoz viszont, hogy tudjunk demodulálni, viszonylag pontosan el kell találnunk a jel frekvenciáját. A vízesés diagram egyfajta szemléltető eszköz, mely megmutatja, hogy éppen növelnünk, vagy csökkentenünk kell a rádiónkon a frekvenciát. Ha a vízesés diagramon középre (1500 Hz-hez) esik a vett jel közepe, akkor a demodulátor valószínűleg képes a feldolgozására. Mivel folyamatos Doppler-korrekcióra van szükség, ezért ha a vízesés diagramon látjuk, hogy a jel már nem középen van, akkor gyorsan korrigálhatjuk a frekvenciát.

Mint azt említettem, a vízesés diagram alapjait nem én írtam meg, így csak azokról a részeiről beszélek, melyeket én készítettem.

5.3.1. FFT

Az eredeti implementáció egy saját készítésű FFT megvalósítást tartalmazott, melyet én a teljesítmény növelésének érdekében a lehető leggyorsabbra cseréltem.

Közkedvelt és szabadon felhasználható FFT library az FFTW[32] 3-as verziója. A használat lényege az, hogy először készíteni kell egy tervet (`plan`), mely viszonylag sok ideig tart[33], viszont utána már elég mindig csak azt az egy tervet végrehajtani. A `plan` létrehozása ezért konstruktorba került, míg a végrehajtására akkor kerül sor, ha összegyűlt megfelelő (ez állítható) számú minta az FFT-hez. Az eredményt némi feldolgozás után szintén signal-lal küldöm tovább a megjelenítésért felelős `WaterfallItem` objektumnak.

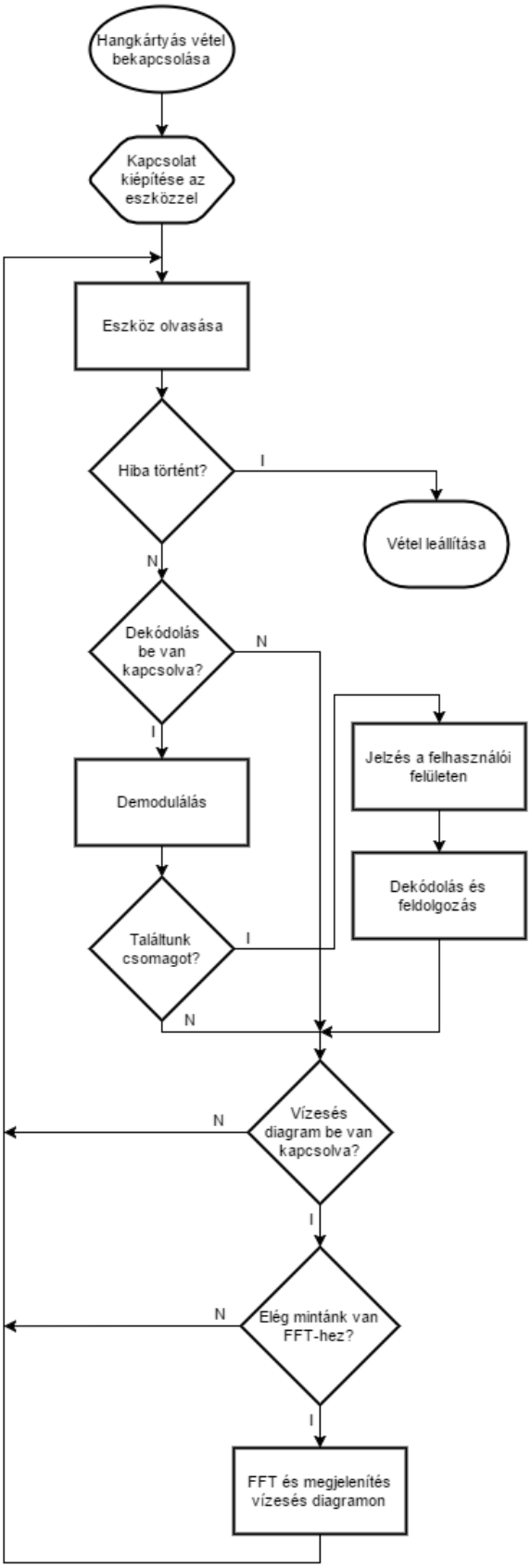
5.3.2. Adaptív színezés

Ahhoz, hogy relatív jelerősséget láthassunk az ábrán, adaptív színezésre van szükség. Ez könnyen belátható, hiszen a lehető legkisebb eltérést is látnunk kell a diagramon az amplitúdók között, ami előre beállított színezéssel nem igazán oldható meg megfelelően.

Minden, feldolgozás során keletkező sor (egyszerre elkészülő amplitúdó értékek) minimum és maximum értékét egy-egy tömbbe rakja a program. Ha a tömb betelne, akkor

természetesen a legrégebbi értéket eldobja. Minden újonnan érkezett amplitúdót elhelyez a minimális értékeket tartalmazó tömb átlagértéke és a maximális értékeket tartalmazó tömb átlagértéke közötti skálán, lineárisan. Ezek után nincs más dolgunk, mint színt rendelni a skálához, és kész is van a színezés.

A színezésnél természetesen az is fontos szempont, hogy a felhasználó meg tudja állapítani, hogy az egyes színek milyen dB értékekhez tartoznak. Ehhez egy skálát helyeztem el a diagram mellé.

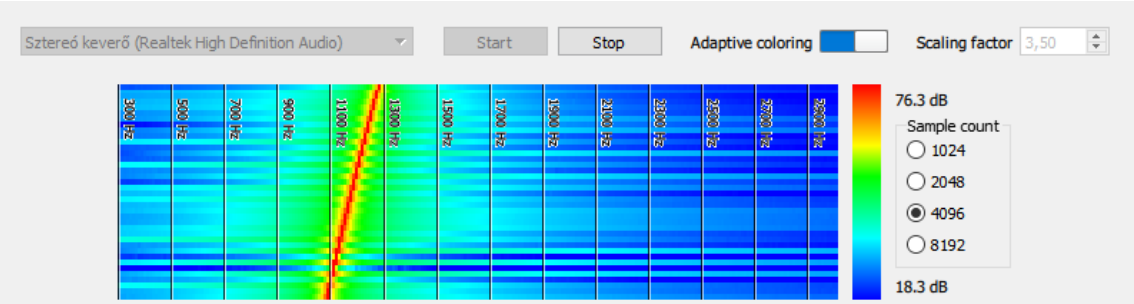


5.1. ábra. *Folyamatábra a hangkártyás vételről*

5.4. Tesztelés

5.4.1. Vízesés diagram ellenőrzése

A vízesés diagram tesztelésénél azt ellenőriztem, hogy valóban megfelelő frekvenciát rendel-e a kapott jelhez a megjelenítés. Ehhez szerencsére adott Windows-on a *Sztereó Keverő* nevű hangeszköz, mely egy audio loopback, így tetszőleges frekvencián lehet ellenőrizni, hogy jó értéket jelez-e.

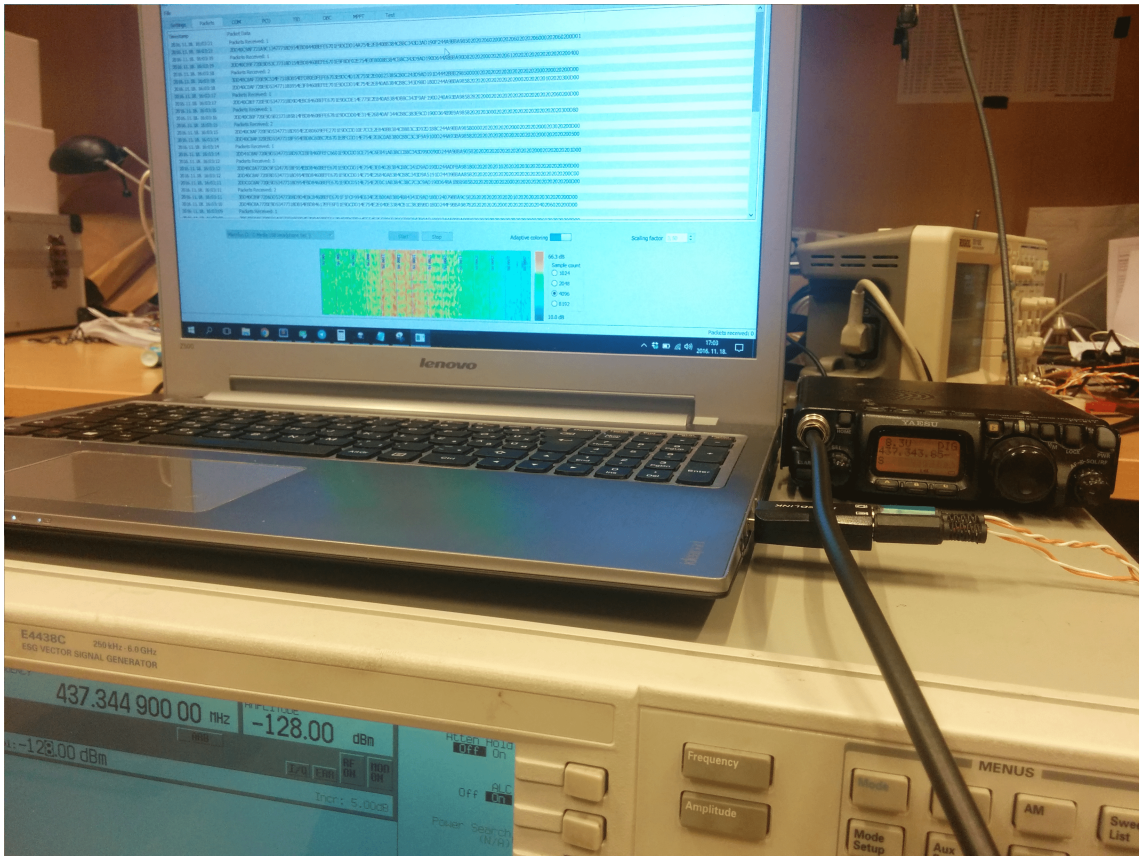


5.2. ábra. *Vízesés diagram pontosságának ellenőrzése up-chirp felhasználásával*

5.4.2. Érzékenységmérés

A demoduláció puszta működésén túl arra is kíváncsiak voltunk, hogy milyen erősségű jel mellett vagyunk még képesek csomagot demodulálni.

Ehhez a jelgenerátorba betöltötem egy csomagot, majd összekötöttem a rádióval. A rádióból kijövő 3,5 mm-es audio jack-et egy USB-s hangkártyán keresztül csatlakoztattam a laptopomhoz. A mérés során először megkerestem, hogy pontosan milyen frekvencián vesz a program csomagokat (nagyobb jelerősség mellett). Ezután fokozatosan addig csökkentettem a jelerősséget, míg a program még tudott jelet demodulálni és csomagokat találni. Olyan mérést is végeztem, ahol a jelgenerátor kimenete előbb egy előerősítőbe ment, majd csak utána a rádióba. Előerősítő nélkül -128 dBm alatt már nem tudtam csomagokat észlelni, de előerősítővel le tudtam menni a jelgenerátor képességeinek határához, azaz -136 dBm-hez (BER $<$ 1 %, PER $<$ 1 %, 1250 bit/s adatsebesség)!



5.3. ábra. *Yaesu FT-817 érzékenységmérése előerősítő nélkül*

6. fejezet

SDR (*Software Defined Radio*)

Az SDR (vagy szoftverrádió) egy olyan eszköz, melynek egyes változatai alacsony árukkal ($\sim$ \$20) igen kedvező lehetőséget nyújtanak különböző frekvenciájú jelek vételére[34][35]. Lehetővé teszik, hogy akár nagy sávszélességű jelet is fogjunk, mely nagyobb adatsebességet biztosít a kommunikációhoz. Természetesen szükség van egy megfelelő vételre alkalmas eszközre is, továbbá a kapcsolat minősége sem elhanyagolható.

Az a célunk, hogy minden áthaladás alkalmával a lehető legtöbb adatot tudjuk venni a műholdtól. Erre több okból is szükség van, ilyen például a fedélzeten rendelkezésre álló háttértár (flash memória) limitált mérete (8 megabájt)[36]. 500 bps-sel nyolc megabájtot átvinni egész pontosan $\frac{8 \cdot 10^6 \cdot 8 \ bit}{500 \ \frac{bit}{s}} = 128\ 000\ s = 35,\dot{5}\ h$ -ba telne, mely a napi 4-6 Magyarország fölötti 2-12 perces áthaladáshoz képest igen sok. És ekkor még nem számoltunk azzal, hogy nem folyamatos az adás és várhatóan nem is tudunk mindent elsőre sikeresen venni, továbbá az adatokat hibajavító kódolással és titkosítással is ellátjuk, ami tovább növeli az érkező adat mennyiségét. Ennél természetesen az 1250 bps már valamivel jobb, viszont jó kapcsolat mellett akár 12 500 bps-sel is adhat majd a műhold. Ehhez viszont nagyobb sávszélesség is kell, és itt jönnek be a szoftverrádiók a képbe.

Én egészen pontosan az RTL-SDR hardverek kezelésével foglalkoztam, melyek a korábban említett árkategóriába tartoznak. Kaphatóak jobb szoftverrádiók is, viszont a kiváló ár-érték arány miatt az RTL-SDR[37] igen elterjedt. Szerencsére ehhez sem kellett nulláról kezdenünk az implementációt, a szabadon felhasználható rtl-sdr library[38] kiváló alapot adott.

6.1. Az rtl-sdr library

Az rtl-sdr egy olyan library, mellyel szinte egyszerű függvényhívássá válik a legtöbb szoftverrádióval kapcsolatos teendő, legalábbis ezen program számára. Képes a leggyakoribb hardvereket kezelni, kellő mértékben tudjuk a segítségével konfigurálni a hardvert és a vételt elindítva jelet is tudunk venni.

Néhány példaprogramot és a library forráskódját is letölthetjük. A példaprogram haszna egyértelmű, a forráskódét is hamarosan kifejtem. Egy kicsit megnehezíti a dolgunkat az, hogy az rtl-sdr felhasználja a libusb library-t[39], melyet minden platformra külön be kell

szereznünk.

6.1.1. A library forráskódjának felhasználása

Általában a libraryk letöltésekor egy header fájlt és egy lefordított fájlt (pl. dll) kapunk. Ez megfelelő mennyiségű dokumentációval párosítva általában elég is, viszont platformfüggetlen alkalmazásnál arra is figyelnünk kell, hogy minden célplatformon beszerezhető legyen a megfelelően lefordított fájl a header mellé.

Ez már korántsem olyan könnyű feladat, a készítés során is sok óra ment csak rá arra, hogy az egyes operációs rendszerekhez sikerüljön megtalálni minden szükséges állományt. Van, ahol beépített csomagkezelő áll rendelkezésünkre, mely megkönnyíti a dolgunkat - viszont ilyen esetben is előfordulhat, hogy a keresett fájl nem szerepel az adott verzióhoz tartozó csomagtárakban. Csomagkezelő nélkül adott esetben még nehezebb dolgunk lehet, esetleg a készítő weboldalán érdemes keresnünk.

A forráskód birtokában azonban nincs más dolgunk, mint a library forrásfájlját is a programunkkal együtt fordítani, mely praktikusabb lehet egyes esetekben, mint a lefordított állományokat előkeresni, hiszen nem is biztos, hogy léteznek.

Ezen kívül természetesen arra is van lehetőségünk, hogy a forráskódhoz hozzányúljunk, ha úgy látjuk, hogy a mi esetünkben erre szükség van. Az rtl-sdr esetében én ezt meg is tettem, kis változtatásra volt csupán szükség, mellyel az eszköz lecsatlakoztatását tudom könnyebben észlelni az alkalmazásban.

6.1.2. Implementáció

A library igen könnyen használható. A segítségével le tudjuk kérdezni, hogy hány RTL-SDR van a géphez csatlakoztatva, és bármelyikkel tudunk kapcsolatot létesíteni. Mi az egyszerűség kedvéért mindig az elsőhöz csatlakozunk. Ezek után már csak frekvenciát, mintavételezési gyakoriságot, auto gain-t és PPM-et[1] kell állítani[2]. Ezt követi az olvasás megkezdése, mely lehet szinkron és aszinkron is, callback használatával.

6.2. Kapcsolat a program többi részével

A programban az SDR kezelés külön szálon kapott helyet és aszinkron módon olvassuk az eszközt. Ez gyakorlatilag azzal jár, hogy megadunk egy függvényt és opcionálisan egy pointert is (és még egyebeket, de ez a kettő a legérdekesebb), mely függvény meghívásra fog kerülni minden olvasás után. A függvény paraméterként megkapja az előbb említett pointert is, ezért felhasználhatjuk ezt arra, hogy a függvény működéséhez szükséges adatokat (*kontextust*) biztosítsuk számára.

A függvény belsejében megtörténik a demoduláció és a demodulált csomag továbbítása is - signal felhasználásával. Érdekes lehet még az a probléma, hogy a Doppler-korrekcióhoz

[1] Ebben a kontextusban ez annak a mértéke, hogy az eszközön lévő oszcillátor által előállított frekvencia mennyire térhet el a névleges értéktől

[2] Érdekes szoftveres szemmel nézve az, hogy használat közben érezhetően felmelegszik az RTL-SDR, és ha nem TCXO van benne, akkor a frekvenciája nehezen kiszámíthatóan változik

szükséges frekvenciát mégis hogyan közlöm egy olyan függvénnyel, mely gyakorlatilag állandóan fut, hogyan lehet azt az értéket egyszerre állítani és olvasni. A függvény által kapott kontextus tartalmaz egy `QMutex` pointert (többek közt), melyet minden futás elején lock-ol, kiolvassa az eredményt és elengedi. Hasonlóan, ha a frekvencia megváltozik (erről a felhasználói felület értesíti a szálat), akkor a mutex-et lock-oljuk, értéket állítunk, majd elengedjük. A demodulátor a frekvencia alapján már képes a helyes működésre. Megjegyzem, hogy az eszköz olvasásának leállítása is hasonlóan működik, csak a mutex lock-olása után más értéket állítunk és olvasunk ki.

Többféle adatsebességgel fog adni a műhold a kapcsolat minőségétől függően, mint azt már említettem. SDR-nél is megvan a lehetősége annak, hogy az összesen demoduláljunk egyszerre, hogy biztosan minden csomagot venni tudjunk. Ez relatíve erőforrásigényes, hiszen rengeteget kell a CPU-nak számolnia, viszont egy újabb gépnek nem feltétlenül okoz túl nagy problémát. Elképzelhető olyan felhasználó, aki hajlandó esetlegesen nagyobb fogyasztással kibékélni, ha cserébe minden csomagot látni fog.

Ha hiba történik olvasás közben, akkor arról a már megszokott módon értesítjük a felhasználót, hogy az esetlegesen kihúzódott eszközt minél hamarabb újracsatlakoztathassa.

6.3. Tesztelés

6.3.1. Doppler-követés

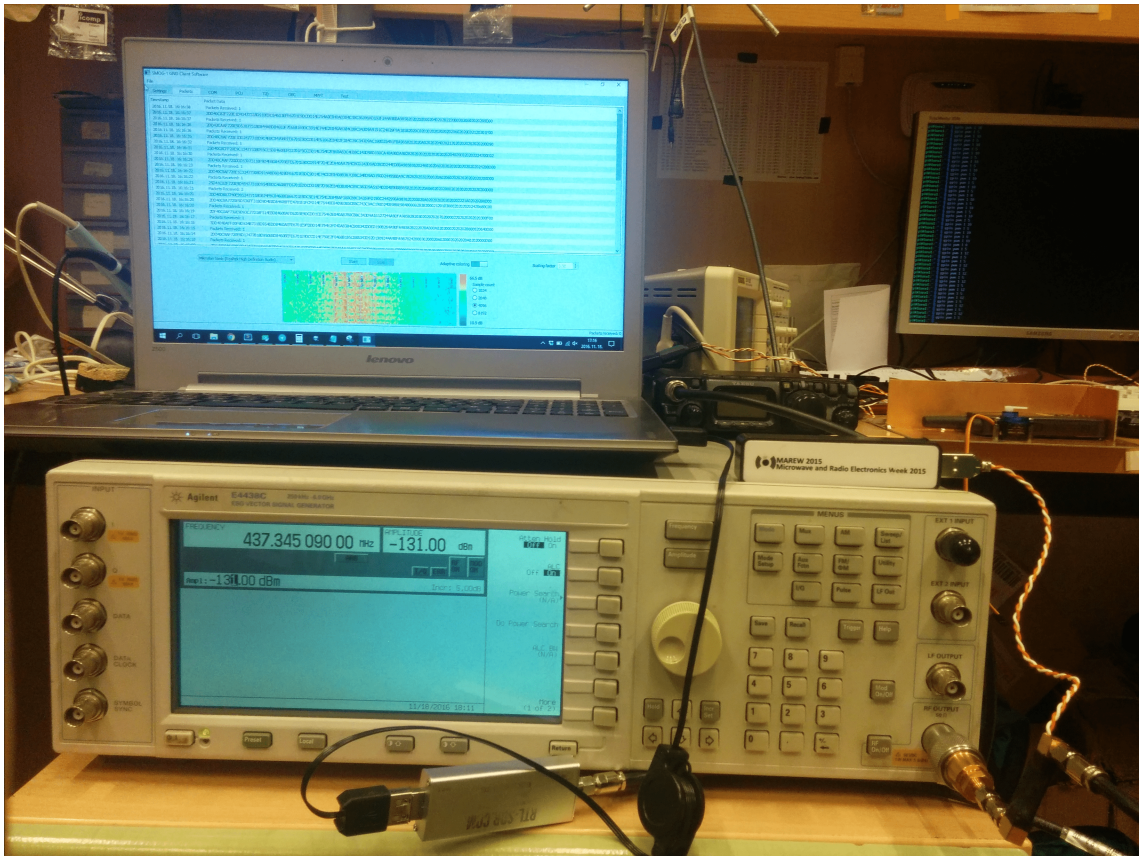
A hardverrádiós tesztnél kicsit nehezebben lebonyolítható a szoftverrádiós Doppler-követés teszt. Ennek oka, hogy míg a rádión látszik, hogy milyen frekvenciára van beállítva, SDR esetében nincsen egyszerűen megnézhető aktuális frekvencia indikátor. Erre azt a megoldást találtuk ki, hogy miközben folyamatosan egy keringő műhold alapján számítunk Doppler frekvenciát, a jelgenerátor frekvenciáját állandóan a számított érték alapján módosítjuk. Ha ilyenkor tudunk csomagot demodulálni, az azt jelenti, hogy az SDR valóban a jó frekvencián vesz és a teszt sikeres.

6.3.2. Demodulátor érzékenységmérés

Mint ahogy a hardverrádióknál, úgy az SDR-nél is kíváncsiak vagyunk arra, hogy milyen kis jelerősség mellett vagyunk képesek csomagokat demodulálni. Az összeállítás is hasonló: jelgenerátor előre betöltött csomaggal, a kimenetére kötött SDR, és végül a laptop, amihez az eszköz csatlakozik. Természetesen itt is teszteltünk előerősítővel és előerősítő nélkül is.

Hasonlóan zajlott a mérés, mint az FT-817-es rádiónál ismertettem, először a generátorral megkerestük a megfelelő frekvenciát nagyobb jelerősség mellett, majd addig csökkentettük azt, amíg még tudtunk venni csomagokat. Az ilyenkor vett csomagok természetesen már tartalmaznak hibákat, melyek javításáról majd a hibajavító kód fog gondoskodni, mely jelenleg még nincsen kész.

Előerősítő használata nélkül -123 dBm-re tudtunk lemenni, míg előerősítővel -131 dBm-ig (BER $<$ 1 %, PER $<$ 1 %, 1250 bit/s adatsebesség).



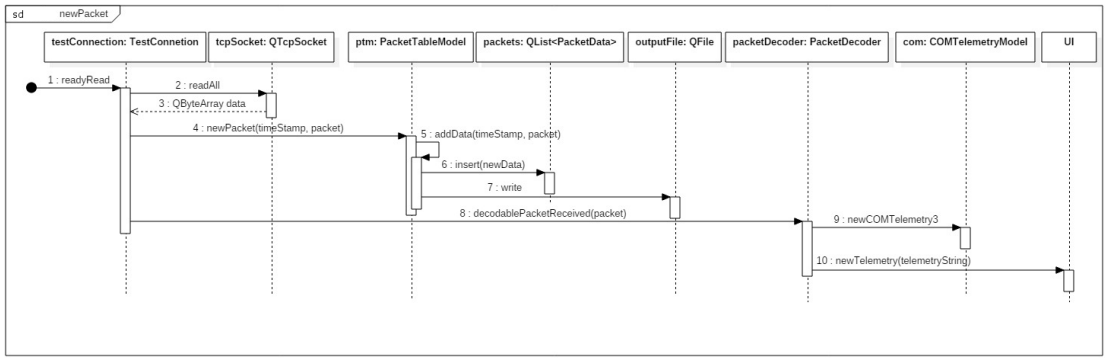
6.1. ábra. *SDR demodulátor érzékenységmérés, előerősítővel*

7. fejezet

Adatcsomagok feldolgozása

Adatcsomagot az eddigiek alapján két forrásból kaphatunk, hangkártyán keresztül és szoftverrádiótól. A műholdfejlesztés jelenlegi stádiumában azonban általában TCP-n keresztül kommunikál a program a műholddal. A forrástól függően egyelőre még más-más feldolgozási lépéseket érintenek a csomagok, a műhold elkészültével viszont ez valószínűleg egységessé fog válni. A végleges változatig még feltehetően sok implementációs részlet fog változni, ahogy a műhold fejlesztése halad.

A megkapott csomagot logolni, a szerverre küldeni és dekódolni kell. A dekódolt csomag tartalmát pedig a felhasználói felületen meg is kell jeleníteni, hogy valóban bárki számára élvezhetővé váljon a műholdkövetés. A fejezet először bemutatja a jelenleg használt TCP kapcsolatot, majd egy csomag útját beérkezéstől egészen a felhasználói felületig.



7.1. ábra. *Szekvencia diagram új OBC telemetria csomag érkezéséről, TCP-n keresztül*

7.1. Kommunikáció a földi állomással TCP-n keresztül

A műhold az elmúlt hónapokban elért abba az állapotba, hogy termovákuum kamrás teszten részt vehessen. Ahhoz, hogy ilyen környezetben az alrendszerek működését figyeljük és leteszteljünk néhány dolgot, muszáj volt már rádiós kapcsolatot használni, hiszen vezetékeket nem tudtunk volna kivezetni. Az összeállítás tartalmazza a műholdat és egy Raspberry Pi-vel összekapcsolt igen kis méretű állomást, ami képes a műhold által sugárzott jelet fogni (és demodulálni), és ő maga is tud jelet modulálni és küldeni. Ehhez az eszközhöz

például Wi-Fi-n keresztül lehet csatlakozni TCP-n keresztül. Ekkor az általa TCP-n keresztül kapott megfelelő formátumú csomagokat modulálja és kisugározza, a műholdtól kapott csomagokat pedig TCP-n továbbítja a programnak.

A `network` modul és a `QTcpSocket` Qt osztály felhasználásával ez sem egy nehéz feladat, mint azt már megszokhattuk. Egy `QTcpSocket` példánnyal könnyen tudunk tetszőleges IP-címhez és porthoz csatlakozni, adatot küldeni és fogadni, majd akár lecsatlakozni, természetesen beépített hibakezeléssel. Az adatok fogadása szintén igen egyszerű, egy signal jelzi, ha érkezett új adat. Ekkor a socket-ből kiolvashatjuk az összes beérkezett adatot és már készen is vagyunk.

7.2. Logolás és szerverre küldés

A demodulált csomagok UTC időbélyeggel együtt beérkeznek a `PacketTableModel` osztály egy slot-jába. Innen a csomagok szöveges formában kikerülnek egy felhasználói felületen lévő táblázatba, egyelőre bármiféle feldolgozás nélkül. A jövőben például azt fogom itt megjeleníteni, hogy milyen az egyes csomagok fajtája (telemetria, mérés, stb.), és ehhez hasonló egyéb információkat. Mivel nagy adatsebesség és kis csomagméret mellett másodpercenként akár húsznál több csomagról van szó, ezért a felület egy bizonyos számú csomag felett csak a csomagok számát írja ki.

Az időbélyegek és a csomagok szöveges formában logba is kerülnek a gépen, mely természetesen csupán egy egyszerű fájlba írás. A logok később feldolgozhatóak (például mint `PacketDecoder` bemeneti adat), ha valamire szükség van belőlük.

A jövőben egy szerverre is be fogja küldeni a csomagokat a program, hogy ha a világon bárki bárhol vesz valamit, arról a műhold üzemeltetői azonnal tudjanak. A backend ehhez még nem készült el, és jelenleg még nem is prioritás, viszont az adatok küldését programból nem lesz túl nehéz megvalósítani. Ha HTTP-n keresztül REST service-t használunk, akkor a TLE letöltéshez használt `QNetworkRequest` osztály jön majd jól, ha TCP-n keresztül küldjük, akkor pedig a `QTcpSocket`.

7.3. Dekódolás

A sikeresen demodulált csomag egy slot-ba érkezik, a `PacketDecoder` osztályban. A dekódoló interfésze egyszerű, ezért a későbbiekben könnyen le is cserélhető az implementáció mögötte. Egyszerű függvényhívás, melynek paraméterei a bemenő adat, annak hossza és a kimenő, már dekódolt adat helye.

Jelenleg három eltérő kategóriába tartozhatnak a Raspberry Pi-től érkező csomagok: parancs küldésének nyugtája, műholdtól vett csomag és egyéb. A kiküldött parancsok második bájtja minden esetben egy, a parancsot azonosító érték. Így tehát ha tudjuk, hogy nyugtáról van szó (az üzenet hossza alapján eldönthető), akkor annak második bájtja alapján minden parancshoz egyedi módon feldolgozhatjuk a nyugtát.

A fejlesztés ezen pontján minden, a műhold által küldött csomag azonos méretű, tartalomtól függetlenül. Ez majd változni fog, azonban jelenleg itt tartunk. A csomagok első négy bájtja időbélyeg, amit egy bájt típusú azonosító követ. Így - a nyugtához hasonlóan -

az ötödik bájt alapján el tudjuk dönteni, hogy milyen csomag érkezett. A csomagok szerkezete struktúraként van definiálva, ezért a dekódolt csomagot annak azonosítója alapján cast-olhatjuk, mely után már hozzá is férünk a benne lévő adatokhoz.

7.4. Megjelenítés a felhasználói felületen

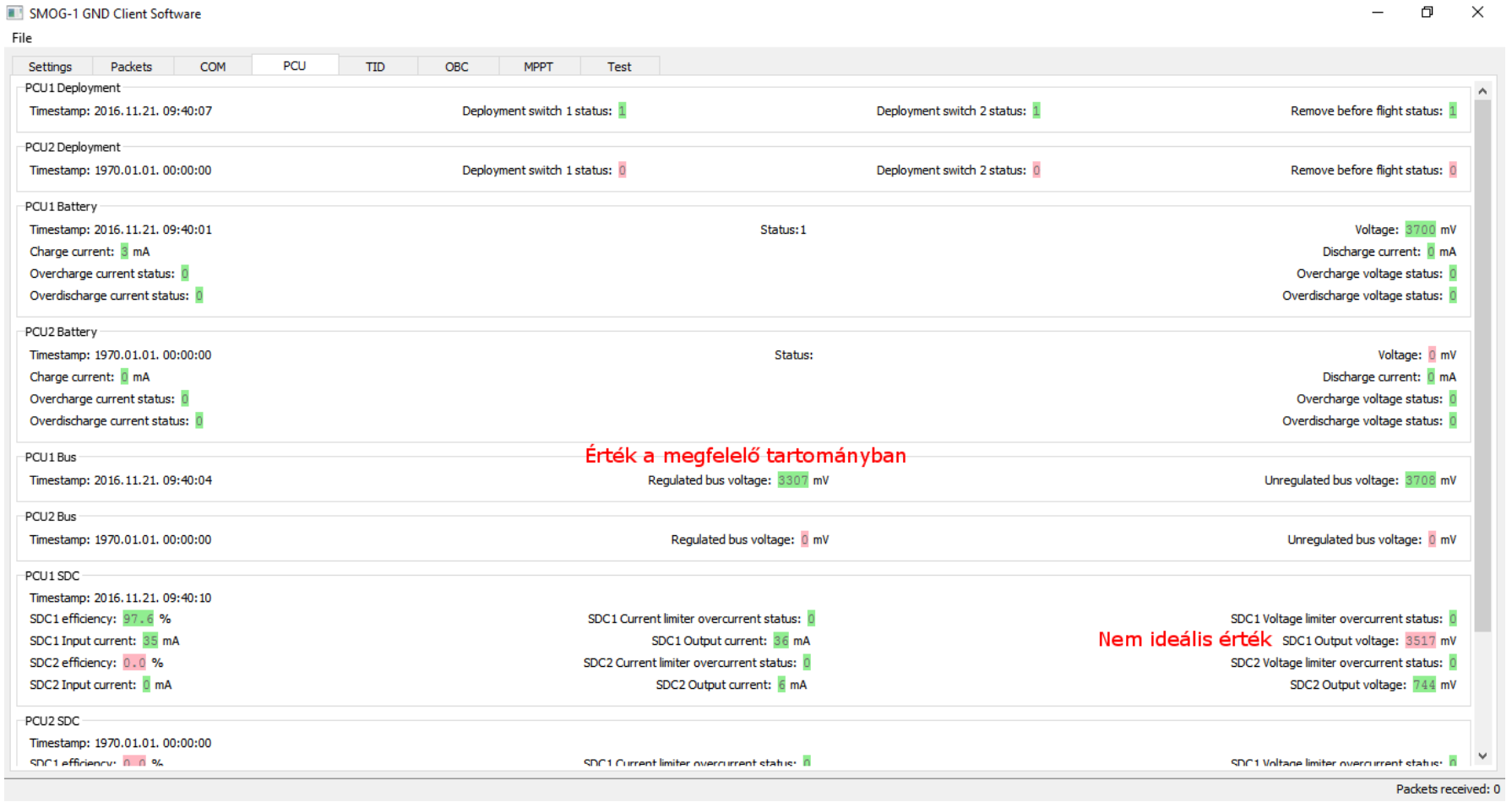
Miután cast-oltam a csomagokat, az adatok elérhetővé váltak számomra. Egy egyszerű szövegdobozba is kikerül formázva a tartalmuk a felhasználói felületen, továbbá az egyes alrendszerek telemetriaadatait számontartó osztályok (`COM-`, `EPS2A-`,`EPS2B-`, `OBC-`, `SOLARTelemetryModel`) slot-jaiba is befutnak ezek az adatok.

Ezen osztályok néhány property-ben tárolják a beérkezett adatokat, melyekhez érték alapján kötve vannak a felhasználói felületen megjelenő komponensek. A legtöbb értékhez rendelhető egy intervallum, amin ha belül van, akkor az helyes működésre utal, míg ha kilóg belőle, az rossz jel. Ezt a felhasználói felületen piros és zöld háttérszínnel szemléltetem. A felhasználói felületen jól jön, ha látjuk, hogy melyik adat frissült a közelmúltban, és melyik az, amelyik feltehetően elavult. Ehhez `QTimer`-eket használok, amik új adat érkezésekor elindulnak, és ha elérik a nullát, akkor `false`-ra állítják egy property értékét. A felhasználói felület érzékeli, ha ez a property megváltozik, és ha elavult adatot indikál, akkor beszürkíti az adott mező szövegét.

A könnyebb áttekinthetőség érdekében a jövő évre tervbe lett véve egy olyan összesítő oldal is, ahol egy szemléltető ábrán láthatjuk a műholdat és alrendszereit, továbbá az alrendszerek fontosabb adatait.

7.5. Tesztelés

A felhasználói felületen való megjelenítés rendkívül könnyen tesztelhető, ahogy a földi állomással kiépített kapcsolat is. A dekóder tesztelése már jóval bonyolultabb, viszont ez nem is az én feladatom, hiszen nem én készítem el a dekódoló (és kódoló) függvényt.

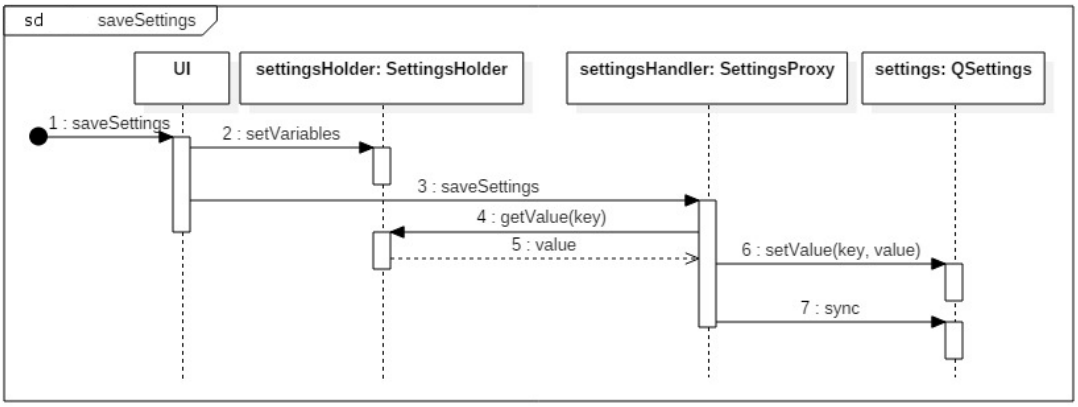


7.2. ábra. *Felhasználói felület, aktív kapcsolat közben*

8. fejezet

Automatizált működés

Ahhoz, hogy egy automatizált állomást vezérelhessen a program, arra van szükség, hogy a program indulásakor tetszőleges operációk elinduljanak, korábban beállított paraméterekkel. Ennek a megvalósítása nem volt annyira egyszerű, de szerencsére ehhez is találtam egy Qt által biztosított platformfüggetlen eszközt.



8.1. ábra. *Szekvencia diagram a beállítások mentéséről*

8.1. Új elemek a felhasználói felületen

El kellett helyeznem új elemeket a felhasználói felületen, hogy a beállítások mentését kényelmessé tegyem, továbbá hogy az egyes operációk (rádió vezérlése, SDR elindítása, stb.) automatikus indítási preferenciáit meg lehessen adni.

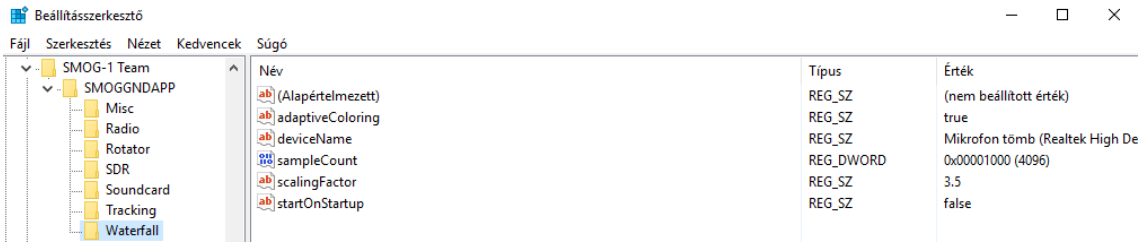
Nem nyújtana túl jó felhasználói élményt az, ha bármilyen érték módosítása automatikusan mentésre kerülne, preferenciától függetlenül. Minden felhasználónak az a legkényelmesebb, ha ki tudja választani, hogy mikor szeretné a beállításokat menteni: mindig, soha, vagy kérdezzen rá a program kilépésnél. Természetesen a beállítások aktuális állapotának mentését menüből is megtehetjük. A logika QML-ben van, mely tudja, hogy a program bezárásakor menteni is kell, felugró ablakot kell feldobni vagy egyszerűen csak kilépni.

Minden operációhoz tartozik ezen kívül egy checkbox, amelynek bepipálása biztosítja az adott operáció automatikus elindítását, a program minden indításánál. A checkbox-ok és a

hozzájuk tartozó feliratok megjelenésén a jövőben még lehet, hogy érdemes elgondolkodni, ahogy az egész felhasználói felület elrendezésén is. Egyelőre azonban a funkcionalitás a prioritás, nem pedig a megjelenés.

8.2. Beállítások mentése

A `QSettings` osztály beállítások perzisztálására igen jól használható. A használatához szükséges kód platformfüggetlen, ugyanakkor lehetőségünk van arra is, hogy az egyes platformokon natív módon kerüljenek a beállítások mentésre - például Windows-on a registry-be kerülnek. Így azzal sem kell foglalkoznunk, hogy hova is kerülnek ezek a beállítások, ezt elfedi előlünk a keretrendszer.



8.2. ábra. *Beállítások Windows registry-ben, hierarchikusan*

Mint az a képen is látható, lehetőségünk van a beállításokat hierarchikusan szervezni, ami megkönnyíti az életünket programozás közben és még a forráskód is átláthatóbb tőle. A `QSettings` tehát egyszerűen csak értékek mentését és betöltését teszi lehetővé, kulcs-érték párokban. A kulcs a beállítás csoportjának és a beállításnak a neve, az értéket pedig a program futása során változhat, így egy változóban kell tárolnunk mentések között.

Erre szolgál a `SettingsHolder` osztály. Ebben az osztályban minden beállításhoz tartozik egy-egy kétirányú adatkötést lehetővé tevő property. Harmincnál is több beállítás van, ezért egy Kristóf Timur által korábban készített makró átalakításával hoztam létre a property-ket. Ez létrehoz privát tagváltozót, inline getter és setter függvényeket, továbbá egy megfelelően felparaméterezett `Q_PROPERTY` makrót is.

A harmadik szükséges osztály a `SettingsProxy`, mely rendelkezik egy `SettingsHolder` pointerrel és egy `QSettings` objektummal. Ezt a kettőt felhasználva képes menteni és betölteni is, méghozzá QML-ből hívható módon. Mentésnél nincs más dolga, mint a pointer által mutatott objektum tagváltozóinak értékeit lementeni, a megfelelő névvel.

8.3. Beállítások betöltése

A beállítások betöltése tehát a `SettingsHolder` osztály megfelelő függvényének QML-ből történő hívásával kezdődik. Ez a függvény kiolvassa az értékeket és signal-ok felhasználásával eljuttatja őket a felhasználói felületre, csoportosítva. Azt is meg lehet adni, hogy milyen alapértelmezett értéket használjunk olvasásnál, ha olyan beállítást próbálunk meg elérni, ami nem létezik. Ez kapóra jön a legelső indításnál, hiszen olyankor még semmi sincsen mentve.

QML-ben JavaScript segítségével feldolgozom ezeket a signal-okat, és inicializálom a felhasználói felület komponenseit. Ez némi odafigyelést igényelt, hiszen például ahhoz, hogy SDR vételt be tudjuk kapcsolni, annak minden beállítását már az indítás előtt meg kell adni.

8.4. Tesztelés

A tesztelés könnyen kivitelezhető volt, csupán meg kellett adnom a kívánt beállításokat, menteni, és megfigyelni, hogy azok helyesen betöltődnek-e.

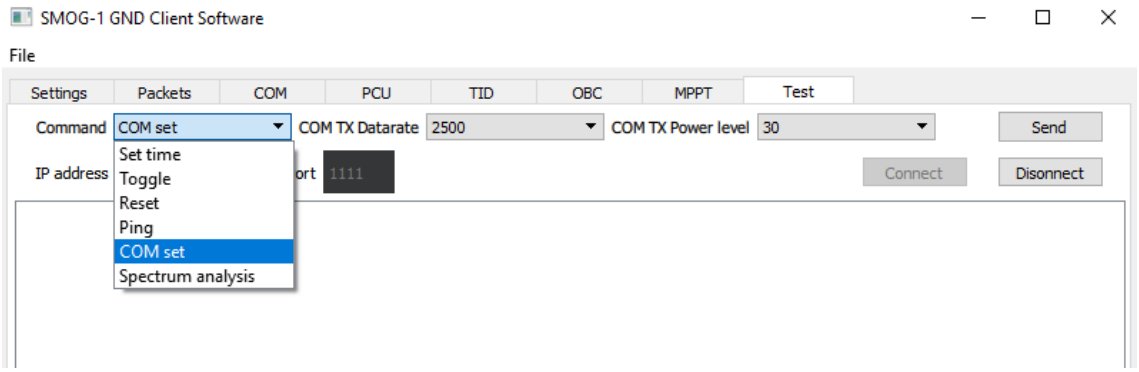
9. fejezet

Műhold vezérlése és spektrumanalízis

Bár a feladatkiírásomban leírva nem szerepelt, mégis szükség volt néhány plusz funkcióra, hogy a műholdfejlesztés és a tesztelés is gördülékenyebben haladhasson, és tudjuk tartani a határidőket. Két dolgot említenék itt meg, a műhold vezérlését és a spektrumanalízis eredményeinek megjelenítését.

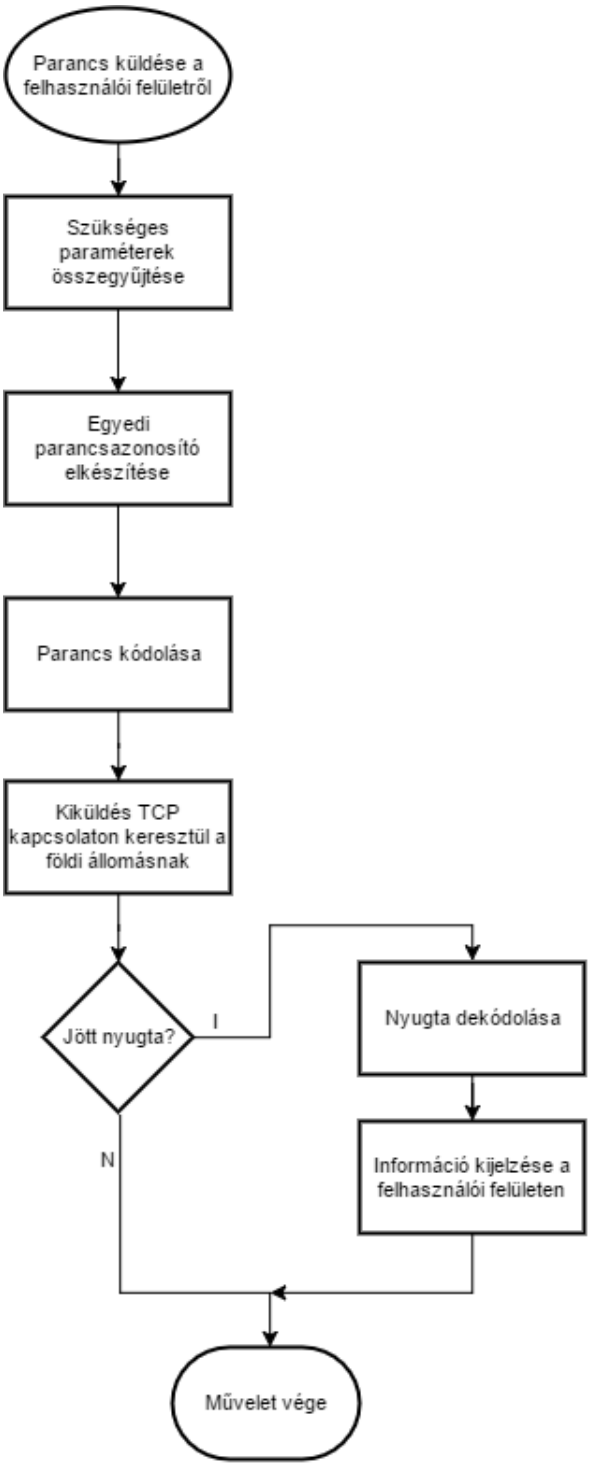
9.1. Műhold vezérlése

A műhold vezérlése jelenleg a már korábban említett módon, egy Raspberry Pi-n keresztül történik, TCP összeköttetéssel. A felhasználói felületen megadhatjuk az IP-címet és a portot, majd csatlakozhatunk. Miután sikeresen csatlakoztunk, lehetőségünk van különféle parancsokat kiadni a műholdnak. Amennyiben a parancsnak van paramétere is, azt automatikusan megjelenő vezérlővel tudjuk megadni.



9.1. ábra. *Parancs küldése a műholdnak*

Miután a felületen rákattintottunk a *Send* gombra, JavaScript felhasználásával felépül a string, melyet a `TestConnection` osztály egy függvénye megkap paraméterként. Némi feldolgozás után, ha a parancs jónak tűnik, akkor egy azonosítót (1 bájtos szám) fűz elé a program, ami alapján tudja majd, ha a műhold nyugtázta a parancsot. Utolsó lépésben kódolja a teljes parancsot, átalakítja a földi állomás által emészthetővé, és kiküldi.



9.2. ábra. *Folyamatábra parancs küldéséről*

9.2. Spektrumanalízis

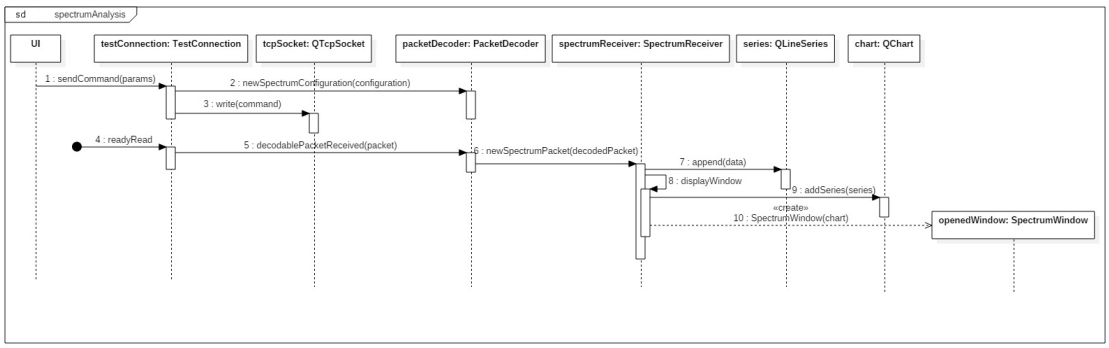
Ha a kiküldött parancs spektrumanalízisre utasította a műholdat, akkor néhány extra lépést is beiktattam. A `TestConnection` parancs küldő függvényéből signal-lal jelzi a `SpectrumReceiver` osztálynak, hogy spektrumanalízis fog történni. Ez a signal tartalmazza a parancs azonosítóját, a kezdő- és végfrekvenciát, valamint a lépésközt. Az utolsó három számra szüksége van a programnak ahhoz, hogy meg tudja majd jeleníteni a mérési eredményeket.

Ha a műhold nyugtázza a parancsunkat, akkor erről is signal fut be a `SpectrumReceiver` osztályba, hiszen innentől már jogosan várjuk adatok beérkezését.

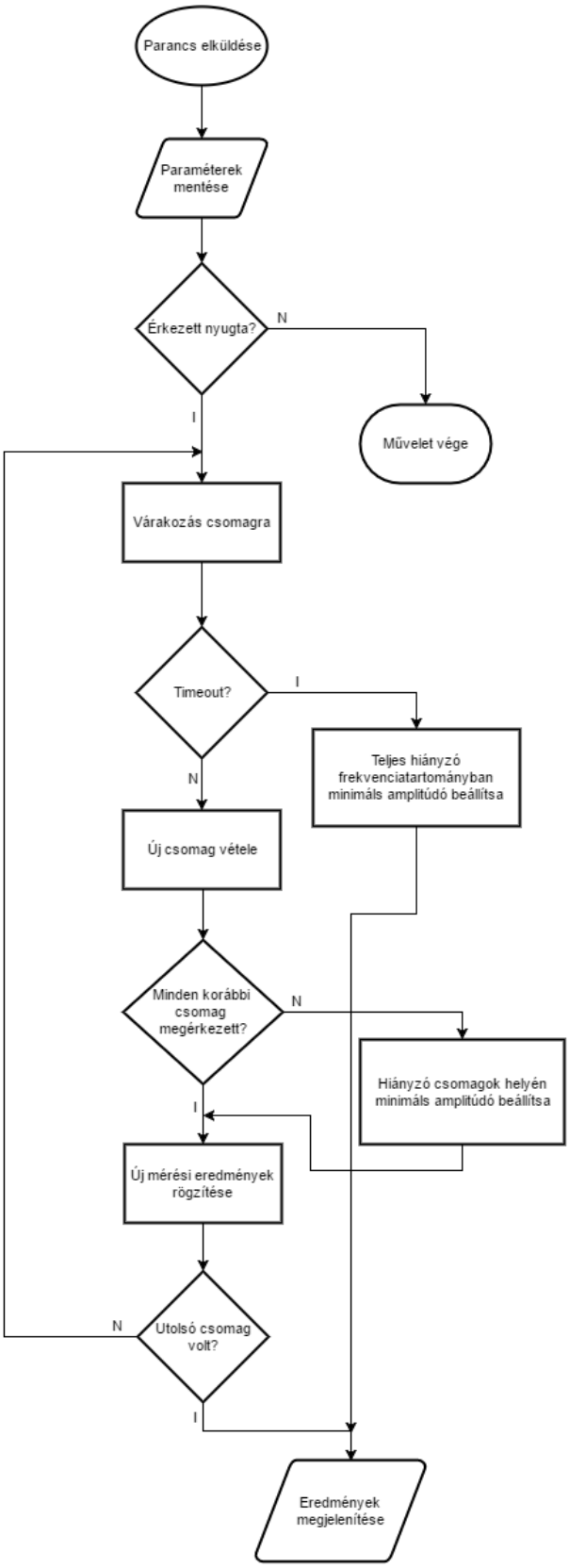
Az ábrázoláshoz és az adatok nyilvántartásához a `charts` modult használom. A `QChart` osztályt példányosítva viszonylag szabadon formázható diagramokat tudunk készíteni, megadhatjuk az adatsort, a tengelyeket és különféle megjelenítési beállításokat. Az adatokat egy `QLineSeries` objektumban tartja a program számon, melybe folyamatosan tölti az értékeket egymás után, végül pedig a `QChart` objektumhoz rendeli.

Mérési eredmény érkezésekor ellenőrizni kell, hogy az azt megelőző csomagok megérkeztek-e, mert ha nem, akkor a kimaradt frekvenciatartományban minden frekvenciához minimális amplitúdót rendelünk. Miután felvittük az új eredményeket, indítunk egy `QTimer`-t, ha még várható csomag. Ha ez a timer lejár azelőtt, hogy érkezne új csomag, akkor úgy vesszük, hogy már nem fog több csomag érkezni, a hátralévő tartományban szintén minimális amplitúdót rendelünk minden frekvenciához. Miután az adatgyűjtés befejeződött (mert minden csomag megérkezett, vagy timeout történt), jöhet a megjelenítés.

A megjelenítéshez példányosítjuk a `SpectrumWindow` osztályt, konstruktorában átadva neki a megfelelő megjelenésűvé konfigurált `QChart` objektumra mutató pointert. Az új objektum létrejöttével együtt megjelenik egy ablak is, ez a szülőosztálytól, a `QMainWindow`-tól örökölt viselkedés. A konstruktorban beállítjuk az ablak néhány paraméterét, majd megjelenítjük a spektrumanalízis eredményét ábrázoló diagramot.



9.3. ábra. *Spektrumanalízis parancs és a válasz feldolgzása, leegyszerűsítve*



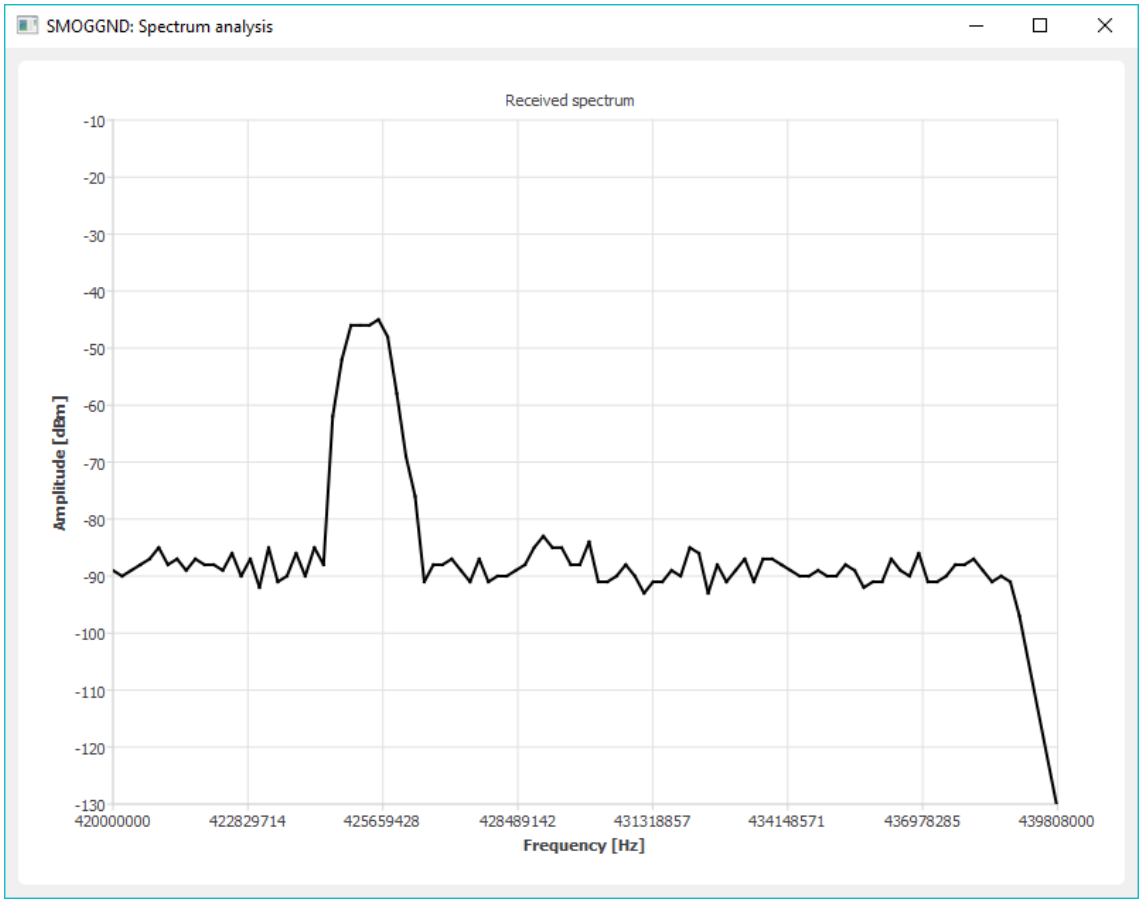
9.4. ábra. *Folyamatábra spektrumanalízis parancsról és az eredmények feldolgozásáról*

9.3. Tesztelés

Minden megvalósított parancsot leteszteltünk, hogy a műhold helyesen veszi-e azt, és hogy válaszol-e rá, amit jól fel is dolgoz a program. Ezeket szerintem nem érdemes egyesével bemutatni, a spektrumanalízis teszt során úgyis adtam ki parancsot és fel is dolgozzuk a választ.

9.3.1. Spektrumanalízis

A spektrumanalízis teszteléséhez beállítottam a jelgenerátort, hogy 425 MHz-en sugározzon jelet, akkora jelszinttel, hogy azt könnyedén látható legyen a mérés eredményén. Nem volt más teendőm, mint jól felparaméterezni a parancsot a műhold számára, kiküldeni azt és várni az eredményeket.



9.5. ábra. *Spektrumanalízis eredménye, melyen jól látható a 425 MHz-en adott jel*

Összefoglalás és a jövő

A szakdolgozat készítése során elkészítettem a leendő SMOG-1 műhold követésére, jelének vételére és feldolgozására alkalmas platformfüggetlen, felhasználói felülettel rendelkező alkalmazás első változatát.

A feladathoz egy korábban általam ismeretlen keretrendszerrel, a Qt-val kellett megismerkednem. Úgy gondolom, hogy ez egész jól sikerült és már én is bátran ajánlom másoknak, ha cross-platform megoldásra van szükségük.

Ahogy a műhold fejlesztése halad tovább, ez a program is a műhold szoftverével együtt fog fejlődni, hogy a remélhetőleg jövő év folyamán pályára álló műholdat minél többen követhessék.

Az előbb említett teendőkön túl valószínűleg nem ártana némi refaktorálás és code review sem. Sok munkát fog igényelni az is, hogy az egyes platformokon könnyen futtatható állományok álljanak rendelkezésre.

Bízom benne, hogy a műhold pályára állásakor kiadott változat már késznek lesz mondható és minél több rádióamatőrnek fog örömöt okozni a megléte.

Köszönetnyilvánítás

Szeretnék köszönetet mondani konzulensemnek, Dudás Leventének, amiért mindig készségesen segítségemre volt és nem csak betanított a feladatot elvégezni, hanem azt is fontosnak tartotta, hogy belelássak a rendszer működésébe, ezáltal tényleg tanuljak valami olyat, amit máshol nem tudtam volna.

Köszönöm Kristóf Timurnak, amiért mindig jó irányba fordított, amikor elakadtam a feladatommal. Ezenfelül köszönöm Géczy Gábornak és Herman Tibornak, hogy minden kérdésemet megválaszolták a műholddal kapcsolatosan, akkor is, ha némelyik a szakdolgozatomon túlmutatott.

Köszönöm dr. Gschwindt Andrásnak, hogy a SMOG-1 projekt vezetésével lehetővé teszi, hogy hallgatók ilyen érdekes témákon dolgozhassanak és köszönöm dr. Seller Rudolfnak, hogy a Mikrohullámú Távérzékelés Laboratóriumban helyet kaptam, hogy dolgozhassak a feladatomon.

Végül, de nem utolsó sorban szeretném megköszönni bátyámnak, Kálmán Attilának, hogy napokon keresztül a szabadidejét annak szentelte, hogy a szakdolgozatomat lektorálja és tanácsokkal lásson el.

Irodalomjegyzék

[1] Dudás Levente, Gschwindt András, The Communication and Spectrum Monitoring System of Smog-1 PocketQube Class Satellite, Microwave and Radar Week, MIKON, Krakkó, Lengyelország, 2016, ISBN: 978-1-5090-2214-4

[2] Dudás Levente, Szűcs László, Gschwindt András, A Smog-1 kisműhold spektrummonitorozó rendszere, Repüléstudományi Közlemények XXVII:(1) pp. 85-105. (2015), Repüléstudományi Konferencia, Szolnok
http://www.repulestudomany.hu/folyoirat/2015_1/2015-1-08-0194-Dudas_L_Szucs_L_Gschwindt_A.pdf

[3] Dudás Levente, Szűcs László, Gschwindt András, The Spectrum Monitoring System of Smog-1 Satellite, 14th Conference on Microwave Techniques, COMITE 2015, Pardubice, pp. 143-146, ISBN:978-1-4799-8121-2
http://dx.doi.org/10.1109/COMITE.2015.7120316

[4] Ayman N. Mohi, Jabir S. Aziz, Lubab A. Salman, *CubeSat Communication System, a New Design Approach*
https://www.researchgate.net/publication/286060825_CubeSat_Communication_System_a_New_Design_Approach

[5] Levente Dudás, Lajos Varga, Rudolf Seller, The Communication Subsystem of Masat-1, the First Hungarian Satellite, Signal Processing Symposium, Jachranka, Lengyelország, 2009
http://dx.doi.org/10.1117/12.837484

[6] Császár János, Dudás Levente, Rádiós kapcsolat a Nemzetközi Űrállomással és a Masat-1 - az első magyar műhold - kommunikációs alrendszerének tesztje, Kommunikáció 2009, Budapest, Zrínyi Miklós Nemzetvédelmi Egyetem, 2009, ISBN:978 963 7060 70 0
https://opac.uni-nke.hu/webview?infile=details.glu&oid=231546

[7] Levente Dudás, Lajos Varga, Masat-1 COM, Antenna Systems & Sensors for Information Society Technologies COST Action IC0603, Dubrovnik, Horvátország, 2010

[8] Dudás Levente, Varga Lajos, Masat-1 - az első magyar műhold kommunikációs alrendszere - pályára állás, műhold vétel és vezérlés, üzemszerű működés, REPÜLÉSTUDOMÁNYI KÖZLEMÉNYEK 2012:(2) pp. 652-673. (2012)

[9] Levente Dudás, Masat-1 - Case Study, ITU Symposium and Workshop on small satellite regulation and communication systems, Prága, 2015
http://www.itu.int/en/ITU-R/space/workshops/2015-prague-small-sat/Pages/default.aspx

[10] Levente Dudás, Levente Pápay, Rudolf Seller, Automated and Remote Controlled Ground Station of Masat-1, the First Hungarian Satellite, Radioelektronika, 24th International Conference, Bratislava, 2014, ISBN:978-1-4799-3715-8
http://dx.doi.org/10.1109/Radioelek.2014.6828410

[11] Dudás Levente, Pápay Levente, Gschwindt András, Seller Rudolf, A MASAT-1 AUTOMATIZÁLT ÉS TÁVVEZÉRELT FÖLDI VEZÉRLŐ ÁLLOMÁSAI, Repüléstudományi Konferencia 2014, Szolnok, Repüléstudományi Közlemények XXVI. évfolyam, 2014. 2. szám, pp 426-442
http://epa.oszk.hu/02600/02694/00065/pdf/EPA02694_rtk_2014_2_426-442.pdf

[12] Holli Riebeek, *Catalog of Earth Satellite Orbits*, 2009
(Hozzáférés: 2016. december 03.)
http://earthobservatory.nasa.gov/Features/OrbitsCatalog

[13] GPREDICT, *About Gpredict*
(Hozzáférés: 2016. december 03.)
http://gpredict.oz9aec.net

[14] PREDICT
(Hozzáférés: 2016. december 03.)
http://www.qsl.net/kd2bd/predict.html

[15] BME-GND, *Földi állomás - műholdkövetés, vezérlés*
(Hozzáférés: 2016. december 03.)
http://gnd.bme.hu

[16] Qt
(Hozzáférés: 2016. december 03.)
https://www.qt.io

[17] Qt, *Qt Reference Pages*
(Hozzáférés: 2016. december 03.)
http://doc.qt.io/qt-5/reference-overview.html

[18] Qt, *The QML Reference*
(Hozzáférés: 2016. december 03.)
http://doc.qt.io/qt-5/qmlreference.html

[19] Qt, *Signals & Slots*
(Hozzáférés: 2016. december 03.)
http://doc.qt.io/qt-5/signalsandslots.html

[20] CelesTrak
(Hozzáférés: 2016. december 03.)
https://www.celestrak.com

[21] CelesTrak, *NORAD Two-Line Element Set Format*
(Hozzáférés: 2016. december 03.)
https://www.celestrak.com/NORAD/documentation/tle-fmt.asp

[22] Hoots, Felix R., and Ronald L. Roehrich, *Models for propagation of NORAD element sets*, AEROSPACE DEFENSE COMMAND PETERSON AFB CO OFFICE OF ASTRODYNAMICS, 1980.
http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA093554

[23] JE9PEL, *All Satellites Frequency List Update*
(Hozzáférés: 2016. december 03.)
http://www.ne.jp/asahi/hamradio/je9pel/satslist.htm

[24] Microsoft, *WM_DEVICECHANGE message*
(Hozzáférés: 2016. december 03.)
https://msdn.microsoft.com/en-us/library/windows/desktop/aa363480(v=vs.85).aspx

[25] Yaesu, *FT-817ND*
(Hozzáférés: 2016. december 03.)
http://www.yaesu.com/indexVS.cfm?cmd=DisplayProducts&ProdCatID=102&encProdID=06014CD0AFA0702B25B12AB4DC9C0D27

[26] Yaesu, *FT-897D*
(Hozzáférés: 2016. december 03.)
http://www.yaesu.com/indexVS.cfm?cmd=DisplayProducts&ProdCatID=102&encProdID=0372FA803B7BBADBF3076C94ACA7A8C5&DivisionID

[27] Kenwood, *TS-2000/X*
(Hozzáférés: 2016. december 03.)
http://www.kenwood.com/au/com/amateur/ts-2000

[28] Yaesu, *FT-817 Operating Manual*
(Hozzáférés: 2016. december 03.)
http://www.anico.sk/manualy/YAESU/FT-817_EN.pdf

[29] Kenwood, *Instruction Manual - TS-2000, TS-2000X, TS-B2000*
(Hozzáférés: 2016. december 03.)
http://www.nvadg.org/images/pdfs/radio_manual_kenwood_ts-2000.pdf

[30] Yaesu, *GS-232B Computer Control Interface for Antenna Rotators* (Hozzáférés: 2016. december 03.)
http://gatorradio.org/Manuals/Yaesu_GS-232B_Manual.pdf

[31] Kristóf Timur, *Audio frequency analyzer app*
(Hozzáférés: 2016. december 03.)
`https://github.com/Venemo/frequency-analyzer`

[32] FFTW
(Hozzáférés: 2016. december 03.)
`http://www.fftw.org`

[33] FFTW, *FAQ - Question 3.3. FFTW seems really slow.*
(Hozzáférés: 2016. december 03.)
`http://www.fftw.org/faq/section3.html#slow`

[34] Maheshwarappa, M.R., Bowyer, M., Bridges, C.P., *Software Defined Radio (SDR) architecture to support multi-satellite communications*
`http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7119186`

[35] Dascal, V., Dolea, P., Cristea, O., Palade, T., *Low-cost SDR-based ground receiving station for LEO satellite operations*
`http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6704456`

[36] Kristóf Timur, *A SMOG-1 PocketQube műhold redundáns fedélzeti számítógépének hardver és szoftver fejlesztése*, TDK dolgozat, 2015 ősz

[37] RTL-SDR
(Hozzáférés: 2016. december 03.)
`http://www.rtl-sdr.com`

[38] Osmocom, *rtl-sdr*
(Hozzáférés: 2016. december 03.)
`http://sdr.osmocom.org/trac/wiki/rtl-sdr`

[39] libusb
(Hozzáférés: 2016. december 03.)
`http://libusb.info/`

[40] Proakis, *Digital Communications*, McGraw-Hill Higher Education, 2005

[41] Roddy, D, *Satellite Communications*, Fourth Edition, McGraw Hill Higher Education, 2006

Függelék

F.1. Demodulátor működése

F.1.1. Hangkártyás demodulátor

Az általános frekvencia modulált jel időtartományban a következő: (F.1)[40][41].

$$S_{FM}(t) = U_v cos[2\pi f_v t + 2\pi k_{FM} \int_0^t S_m(\alpha) d\alpha] \tag{F.1}$$

Ahol:

- U_v a vivő amplitúdója,
- f_v a vivőfrekvencia,
- k_{FM} a frekvencia modulációs tényező,
- $S_m(t)$ a moduláló jel.

A moduláló jel digitális esetben egy, az adatbiteknek és az adatsebességnek megfelelő +/- 1 amplitúdójú T_{bit} hosszúságú négyszögimpulzus sorozat.

Ahhoz, hogy a SMOG-1-gyel történő rádiókommunikáció spektrálisan a lehető leghatékonyabb legyen, az FSK átvitelnek egy speciális esetét használjuk: a GMSK-t (*Gaussian Minimal Shift Keying*).

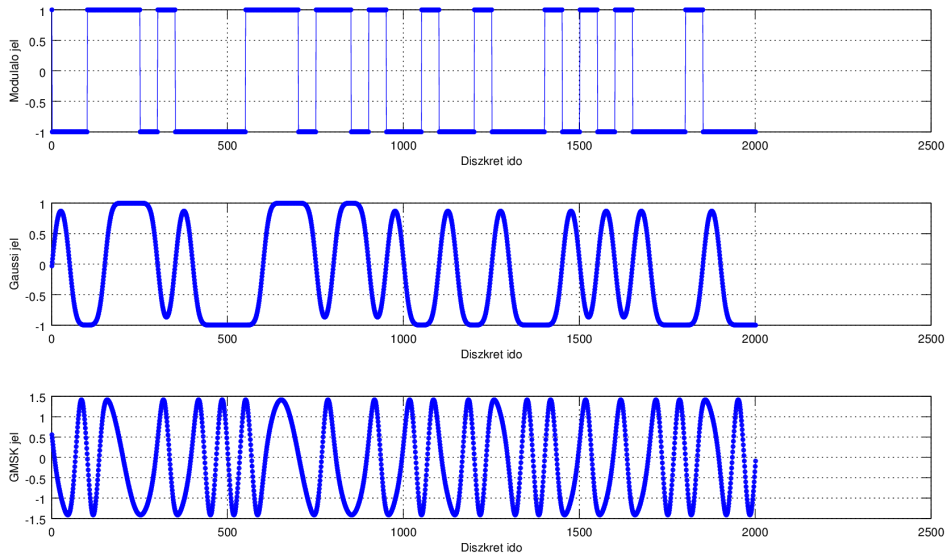
A diszkrét minták segítségével előállított FM jel (F.2) szerint alakul.

$$S_{FM}[k] = U_v cos\{2\pi f_v / f_s k + 2\pi f_{dev} / f_s \sum_{i=0}^{k} S_m[i]\} \tag{F.2}$$

Ahol:

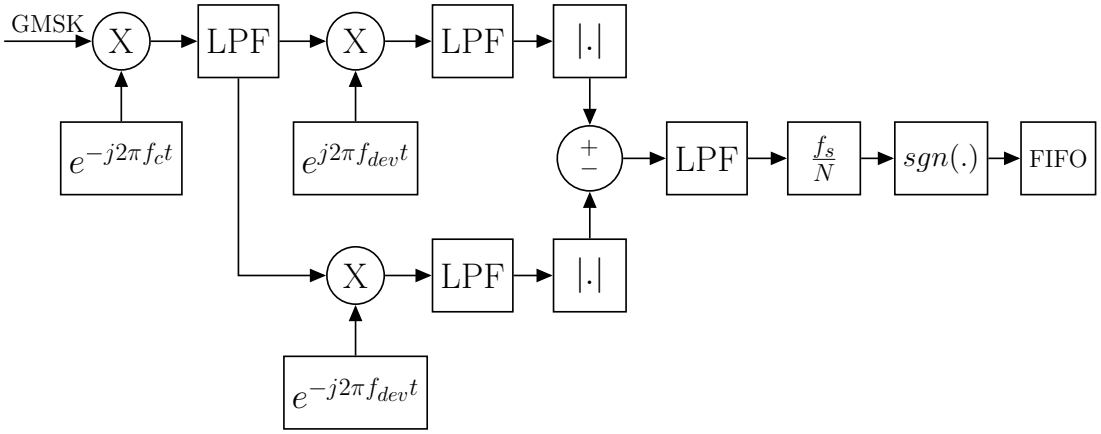
- k a diszkrét idő,
- f_s a mintavételi frekvencia,
- f_{dev} a frekvencialöket, amely (G)MSK esetén az adatsebesség negyede,
- $S_m[i]$ az adatsebességnek megfelelő hosszúságú négyszög-impulzus sorozat (adatbitektől függően +/- 1 amplitúdóval).

GMSK esetben a négyszög impulzus sorozatot egy megfelelő Gaussi ablakolású FIR szűrőn keresztül átengedve kerül az összegzőre: F.1 ábra.



F.1. ábra. *A GMSK jel előállításának lépései*

A demodulátor feladata a hardver rádióról jövő digitalizált hangfrekvenciás GMSK jel átalakítása telemetria adatcsomagokká. A funkcionális blokkdiagram F.2 ábrán látható.



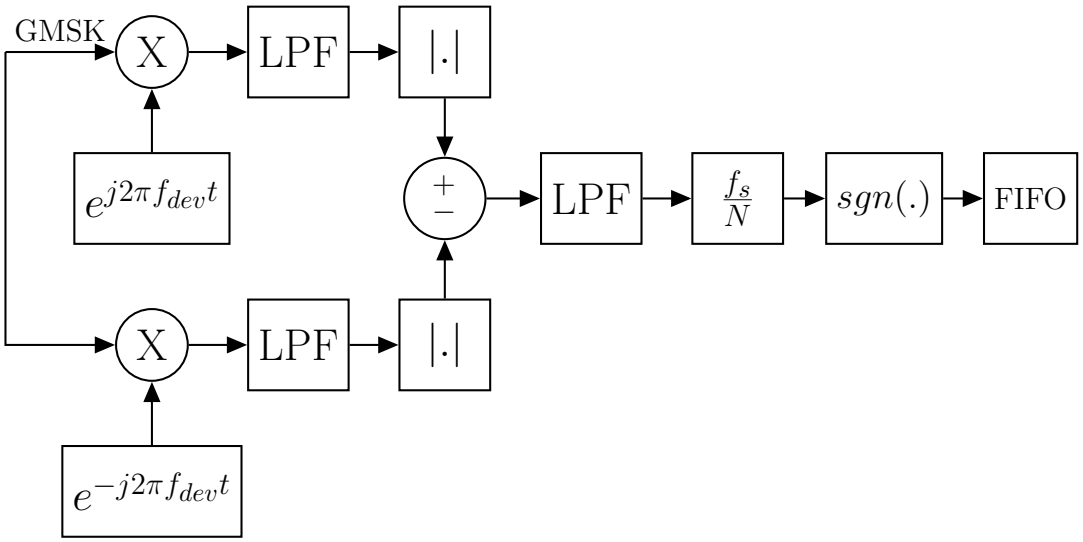
F.2. ábra. *A hangkártyás GMSK demodulátor szerkezete*

A demodulátor kód a digitalizált hangfrekvenciás jelet I-Q-ban keverve egy segédoszcillátor jellel (mert a hangkártya csak 20 Hz - 20 kHz-ig visz át jelet) leteszi I-Q alapsávra, ahol ismételten komplexen keveri a löket illetve negatív löket frekvenciának megfelelő szinuszos jellel, amellyel lényegében egy illesztett szűrős vevő valósul meg. A +/- löket frekvenciás korrelátor kimenetén a jelek amplitúdóját hasonlítja össze, amely az adatbiteknek megfelelő mintavételezett zajos négyszögjelhez hasonló jelet jelent. Ezt a továbbiakban megfelelően újra mintavételezi úgy, hogy 3 minta jusson egy adatbitre (többségi kemény döntés). A komparált mintasorozatot egy megfelelő méretű FIFO-ba tölti, és teszteli az adatcsomagok elején levő csomagszinkront biztosító 2 (vagy 4) bájt adatot. Ha a csomagszinkront megtalálta a vett jelben, akkor a FIFO tartalma alapján létrehozza a demodulált telemetria

adat csomagot.

F.1.2. SDR demodulátor

A szoftverrádióval digitalizált I-Q alapsávi jel demodulációjánál a hangkártyás demodulátorhoz képest kicsit egyszerűbb demodulátort használunk: F.3.



F.3. ábra. *Az RTL-SDR-nél használt GMSK demodulátor felépítése*