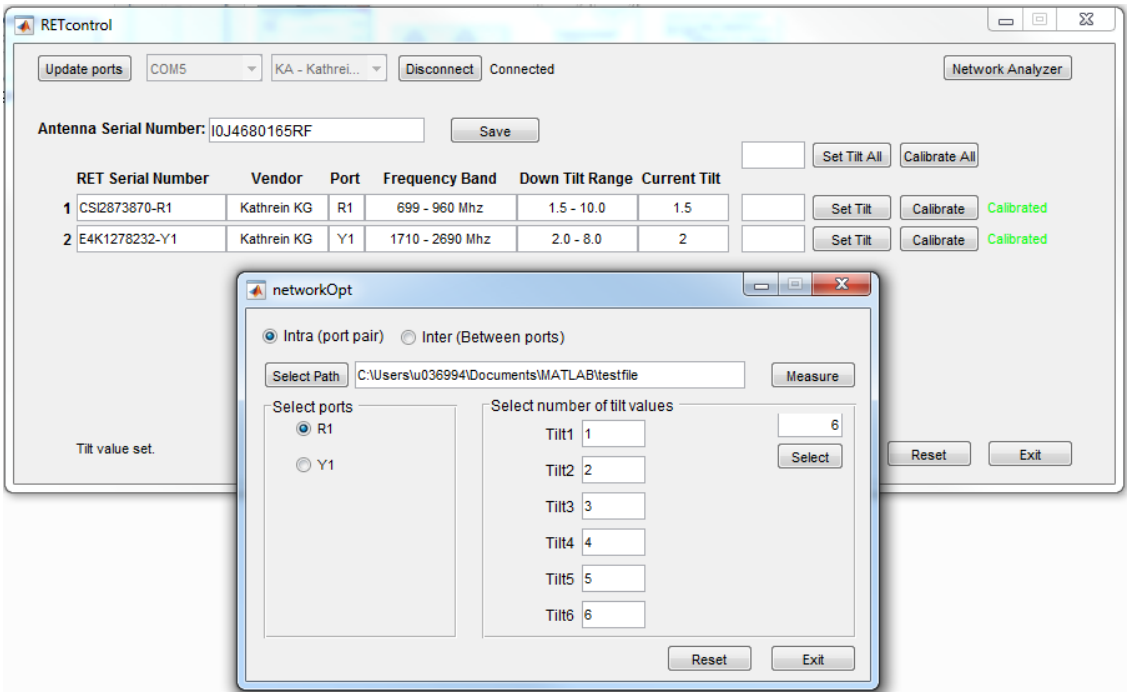
CHALMERS



Fjärrstyrning av elektrisk tilt

Styrning av basstationsantenner vid antennmätning

Examensarbete Elektroingenjör vid institutionen för Elektroteknik

Oscar Johansson

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sweden 2017

Examensarbete 2017

Fjärrstyrning av elektrisk tilt

Styrning av basstationsantenner vid antennmätning

Oscar Johansson

CHALMERS

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sweden 2017

Fjärrstyrning av elektrisk tilt
Styrning av basstationsantenner vid antennmätning
Oscar Johansson

Handledare: Jesper Pedersen, Institutionen för Elektroteknik
Examinator: Thomas Eriksson, Institutionen för Elektroteknik

Examensarbete 2017
Institutionen för Elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Telephone +46 31 772 1000

Titelsida: GUI för stryning utav Remote Electrical Tilt enheter skapat i MATLABs GUIDE.

Typeset in LaTeX

Remote Electrical Tilt
Oscar Johansson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Saab Surveillance's control and measurement of antennas is mainly done manually, which is ineffective and takes time. Saab is looking for an automated solution, handled by the programming language MATLAB, which can manage the control of the antennas and perform measurements with a network analyzer. The solution was a program in MATLAB that controls the antenna Remote Electrical Tilt (RET) and Remote Control of the network analyzer, where communication was performed according to the Antenna Interface Standard Group (AISG) standard. The program controls the antennas tilt values, gets information about the antenna and manages calibration. This program can then perform two automated measurements, intra (one port pair) and inter (between port pairs), and the measurements can be saved via Remote Control of the network analyzer.

Fjärrstyrning av elektrisk tilt
Oscar Johansson
Institutionen för Elektroteknik
Chalmers tekniska högskola

Sammanfattning

Styrning samt mätningar av antenner på Saab Surveillance görs idag huvudsakligen manuellt. Detta tar tid och är ineffektivt. Saab söker en lösning genom ett automatiserat program, konstruerat i programmeringsspråket MATLAB, som kan utföra både styrning av antenner samt mätning med nätverksanalysator. Uppgiften blev att utveckla ett program som styr antennens Remote Electrical Tilt (RET) samt Remote Control utav nätverksanalysatorn via MATLAB. Kommunikationen utfördes enligt Antenna Interface Standard Group (AISG) standard. Programmet styr antennens tilt-vinklar, hämtar information samt sköter kalibrering. Med detta program kan två automatiserade mätmetoder utföras, intra-mätning (portpar) och inter-mätning (mellan portpar). Mätningarna sparas ner genom fjärrstyrning utav nätverksanalysatorn.

Förord

Examensarbetet ingår som ett moment i elektroingenjörsprogrammet, 180 HP, vid Chalmers tekniska högskola och omfattar 15 HP. Projektet har utförs vid Saab Surveillance's antennmätsträcka A15 i Lackarebäck, Mölndal.

Jag vill förutom de anställda på A15 rikta ett särskilt stort tack till:

- Christian Augustsson, Saab Surveillance, A15.
- Jesper Pedersen, Institutionen för Elektroteknik på Chalmers Tekniska Högskola.
- Bo Granstam, Ericsson AB, Lindholmen.

Oscar Johansson, Göteborg, 2017

Innehåll

Beteckningar

3GPP 3rd Generation Partnership Project
AISG Antenna Interface Standard Group
CRC cyclic redundancy check
DISC disconnect
GUI graphical user interface
GUIDE GUI development environment
HDLC High-Level Data Link Control
OSI Open Systems Interconnection
RCU Remote Control Unit
RNR receive not ready
RR receive ready
REJ reject
RET Remote Electrical Tilt
RSET reset
SREJ selective reject
SNRM Set Normal Response Mode
TSG Technical Specification Groups
TMA Tower Mounted Amplifier
UA Unnumbered Acknowledgment
XID Exchange Identification

Figurer

1
Inledning

Detta examensarbete syftar till att konstruera ett MATLAB-program som kan styra Remote Electrical Tilt (RET) enheter från olika tillverkare på marknaden.

1.1 Bakgrund och syfte

Saab Surveillance erbjuder mätning av basstationsantenner för mobilsystem till externa kunder. Antennerna blir allt mer komplicerade och därför ställs det höga krav på snabb och effektiv mätning. Antennerna har elektriskt styrda lober som man enkelt och smidigt vill kunna ändra tilt-värde på när man gör mätningar. Denna styrning görs idag antingen manuellt eller med styrenheter som kunderna själva skickar med. Problem uppstår då styrenheterna inte alltid är med vid leveransen av antennen. Mätningarna utförs med en nätverksanalysator som även den sköts manuellt. Skulle man kunna integrera styrningen utav de elektriska loberna och mätningen med nätverksanalysatorn i samma system skulle det göra arbetet effektivare och därmed kan fler mätningar utföras. Saab vill framför allt ha en lösning som är kompatibel med utvecklingsverktyget MATLAB då detta är den främsta programvaran de redan arbetar i för beräkningar med mera.

1.2 Mål

Målet är att i MATLAB konstruera ett program som kan kalibrera antennerna, sköta lobstyrningen samt spara ner viktig data som serienummer, frekvensband etc. Programmets funktioner ska kunna köras separat men även ett grafiskt användargränssnitt ska skapas för att kunna integreras med nätverksanalysatorn.

1.3 Kravspecifikation

Saab har specificerat följande krav för projektet: Programmet(s)

- ska skrivas i MATLAB.
- ska kunna spara ner information om antennen och Remote Control Unit (RCU), som serienummer, tillverkare, frekvensband och tilt-spann.
- ska kunna kalibrera antennen och sätta ett specifikt tilt-värde.
- ska kunna styra kalibrering och tilt-värde på samtliga antennportar, separat och simultant.
- ska implementeras med en nätverksanalysator av modell Rohde-Schwarz ZNB40 Vector Network Analyzer.
- funktioner ska vara skrivna så att de även ska kunna integreras i andra MATLAB-program.

Slutligen ska ett grafiskt användargränssnitt för programmets funktioner konstrueras.

1.4 Metod

Projektet inledes med att undersöka eventuella lösningar. Det resulterade i att kommunikationen mellan MATLAB och RET-enheterna skulle skrivas från grunden via Antenna Interface Standard Group (AISG) standarden. Därför genomfördes efterforskning på hur just denna standard är utformad och hur man använder den. Utifrån detta och loggarna i Appendix A så kunde High-Level Data Link Control (HDLC) ramar med kommandon för RET-enheterna skapas i MATLAB.

Den fysiska kopplingen mellan PC och RET-enhet etablerades med USB-till-RS485 adapter, 24V DC adapter och en Kathrein RET kabel. Detta kopplades till Comba RCU-003(IV) RCU-enhet som i sin tur satt monterad på en Comba dummy-antenn.

Funktionerna konstruerades för att kunna kontrollera och hämta information om RET-enheterna och antennen till programmet.

När alla kommandon fungerade skapades ett graphical user interface (GUI). Efter att dummy-antennen kunde kontrolleras i det grafiska användargränssnittet så seriekopplades två Kathrein RCU-enheter som monterades på Kathrein-antennen. Detta gjordes för att konstruera ett Exchange Identification (XID) kommando som upptäckte och anslöt flera enheter. Utifrån det uppdaterades GUI och huvuddelen av programmet färdigställdes.

Sista delen av projektet innefattade att lägga till mätningar av S-parametrar med nätverksanalysator. För detta skapades ett nytt GUI för att kunna välja mellan två mätningar, mellan olika portpar eller enskilda portpar. Med funktioner skapade av Saab så öppnades kommunikation med nätverksanalysator och mätningar kunde utföras med fjärrstyrning.

2
Hårdvara, protokoll och mjukvara

I detta kapitel beskrivs den teori, hårdvara samt mjukvara som används i projektet.

2.1 Hårdvara

De hårdvarukomponenter omnämnda i kravspecifikationen samt hårdvara som tillkommit beskrivs här i närmre detalj. Figur 2.1 visar kopplingsschema för hårdvaran.

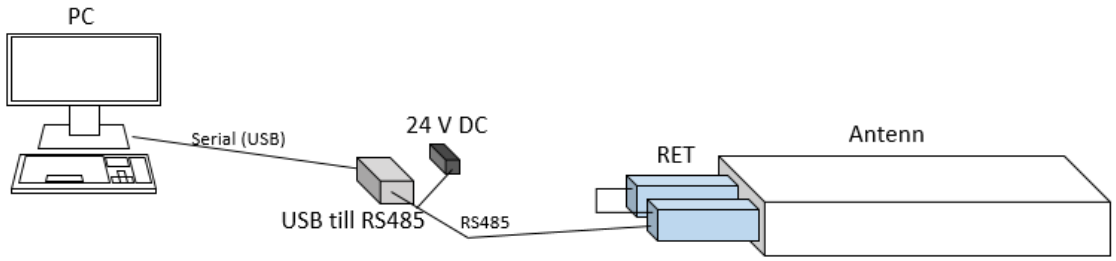


Figure 2.1: Kopplingsschema för koppling mellan PC och RET.

2.1.1 Antenn

För att kunna verifiera att koden var korrekt skriven gjordes tester på en antenn. Den antenn som användes var en Kathrein 4-port antenn 80010666V01. Denna har två portpar, R1 och Y1. R1 har ett frekvensband mellan 698-960 MHz och tilt-vinklar mellan 1.5-10°. Y1 har ett frekvensband mellan 1710-2690 MHz och tilt-vinklar mellan 2-8°[1].

2.1.2 Dummy-antenn

Vid tester av funktioner i programmet så användes ett Comba Dummy Antenna Kit av typen ODU (Dual-band) kit, se figur 2.2. Det är en simulering av en antenn där man enkelt kan se dess tilt-vinkel, genom att en sticka med tilt-vinkelnummer rör sig. Det är dessutom betydlig mer hanterbart än en basstationsantenn eftersom dummy antennen är mindre. Dummy-antennen har två portpar, B1 och B2. Båda portparen har ett frekvensband mellan 1710-2170 MHz. Tilt-vinklarna på båda portarna är mellan 0-10°.

Figure 2.2: Comba ODU (Dual-band) Dummy antenna kit.

2.1.3 RET

För att kontrollera en antenns RET så används en RCU-enhet. Dessa gör det möjligt att bland annat ändra det elektriska tilt-värdet utan att fysiskt behöva vara på plats. Elektrisk tilt innebär att man ändrar signalfasen för vare element i antennen, vilket resulterar i en ändring av antennens strålningsriktning, se figur 2.3. I projektet användes tre stycken RCU-enheter. En Comba RCU-003(IV) RCU-enhet, figur 2.4 som var kopplad till dummy-antennen, och två stycken Kathrein 86010148V01 RCU-enheter. Kathrein RCU-enheterna var kopplade till Kathrein-antennen.

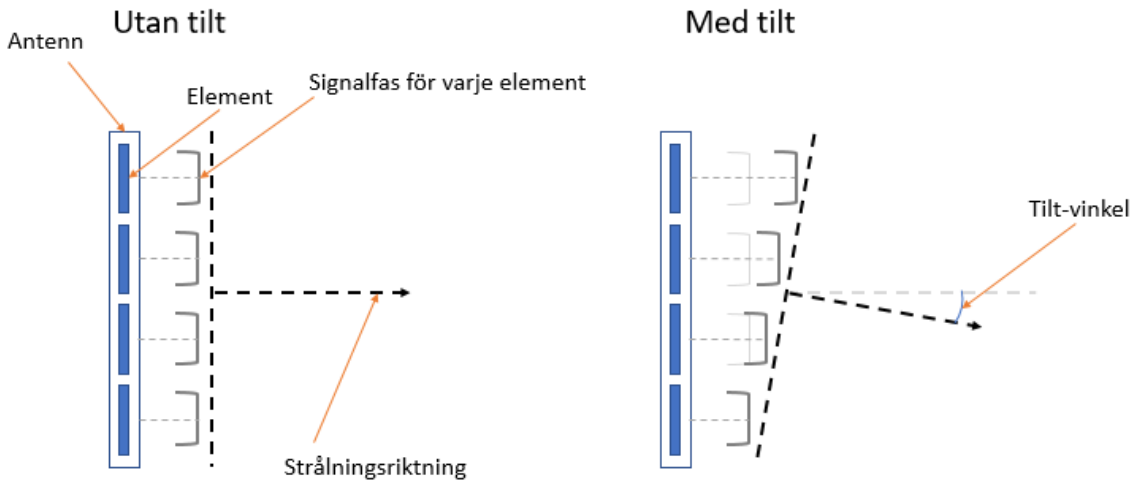


Figure 2.3: Illustration för hur elektrisk tilt fungerar.

Figure 2.4: Comba RCU-003(VI) Remote Control Unit.

2.1.4 USB till RS-485

För konvertering från USB till RS-485 används en Titan USB-COMi-TB USB-to-Industrial Single RS-422/485 Adapter [2], se figur 2.5. Den används i RS-485 Half duplex (2 wire) mode enligt AISG-standarden. För att kunna kommunicera med RET-enheten krävs även en RET-kabel. Till detta användes en Kathrein 86010054 Control Cable [3].

2.1.5 Nätverksanalysator

Den nätverksanalysator som ska användas är en Rohde-Schwarz ZNB40 Vector Network Analyzer, 2 Port, 40 GHz. Nätverksanalysatorn kommer att fjärrstyras.

2.2 RS-485

RS-485 är en av flera standarder för seriell kommunikation. Den kommunicerar via partvinnade kablar, för att motverka störningar [4]. RS-485 har två olika inställningar, tvåkablad och fyrkablad. Med tvåkablad RS-485 delar sändare och mottagare på en partvinnad kabel. Detta är en så kallad halv-duplex kommunikation där bara en enhet kan sända åt gången. Fyrkablad RS-485 möjliggör full-duplex vilket innebär att både sändare och mottagare kan sända och ta emot samtidigt. I detta projekt används halv-duplex som tidigare nämnt i 2.1.4.

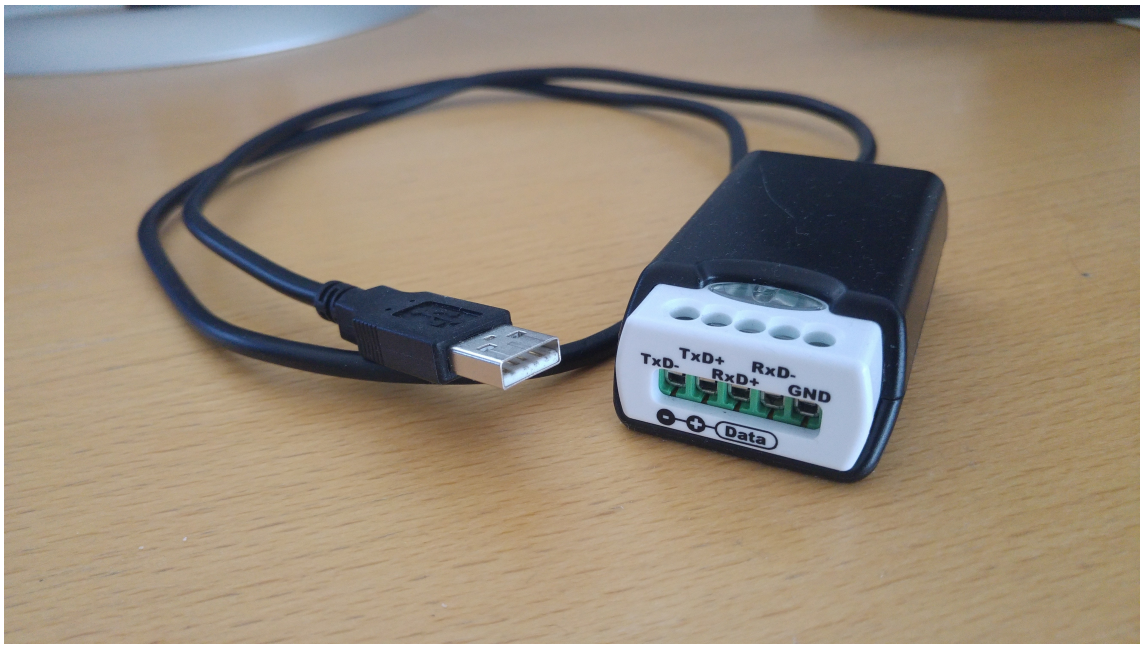
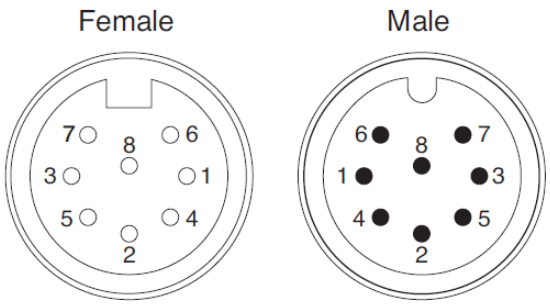


Figure 2.5: Titan USB-COMi-TB USB-to-Industrial Single RS-422/485 Adapter.

2.3 AISG

AISG är en öppen standard för digital kontroll och övervakning av olika antenntyper [5]. Deras medlemmar är teknikföretag inom trådlös kommunikation. AISG:s senaste version [6] av standarden utgår från 3rd Generation Partnership Project (3GPP) dokumentation [7]-[9]. Där presenteras tillägg för lager 1, 2 och 7 av Open Systems Interconnection (OSI) modellen som är specifikt för AISG standarden. För lager 1 (fysiska lagret) så behandlar standarden specifikationer för termineringsimpedans, ström- och effektkonsumtion, brus och rippel, DC-matning och kontaktens gränssnitt. Standardformatet för kontakten är RS-485, vilket visas i figur 2.6.



PIN assignment according AISG:
1 +13 V DC (+12 V DC nominal)
2 not connected
3 RS485 B
4 not connected
5 RS485 A
6 +29 V DC (+24 V DC nominal)
7 DC Return
8 not connected

Figure 2.6: RS-485 kontakt med PIN-tilldelning [3]. Används med utgivarens tillstånd.

Standarden presenterar tillägg till lager 2 (datalänklagret) för enhetstyper och protokollversionskommandon. Specifikationer för lager 7 (applikationslagret) är kommandon tillhörande Tower Mounted Amplifier (TMA) enheter. I AISG v2.0 Annex A [6] så är alla företag med tillhörande tillverkarkoder listade.

2.4 HDLC

Enligt AISG v2.0 [6] så är Lager 2 (datalänklagret) i OSI-modellen baserat på en anpassad version av High-Level Data Link Control (HDLC) standarden. Figur 2.7 visar hur denna version är uppbyggd. All information skickas som en lång sträng av oktetter, vilket representeras av 8-bitar binärt eller som två tecken hexadecimalt. Exempel på hur en sträng kan se ut: 7E 01 54 33 02 00 0A 00 73 4E 7E.

Flagga 1 oktett	Adress 1 oktett	Kontroll 1 oktett	Procedur-ID 1 oktett	Antal dataoktetter 2 oktetter		Dataoktetter	CRC 2 oktetter		Flagga 1 oktett
0x7E	Enhets-address	Kontrollbitar, I-,S- eller U-ramar	Kommando-kod	Låg oktett	Hög oktett	Varierande längd (måste ha underlag för maximal längd på 71 oktetter)	CRC1 Låg oktett	CRC2 Hög oktett	0x7E

Figure 2.7: HDLC-protokoll enligt Annex D i AISG v2.0 [6]. Visar i vilken ordning de olika oktetterna ska vara sammansatta för ett enskilt kommando.

Det finns tre typer av HDLC-protokoll: Information-, Supervisory- och Unnumbered-protokoll [10].

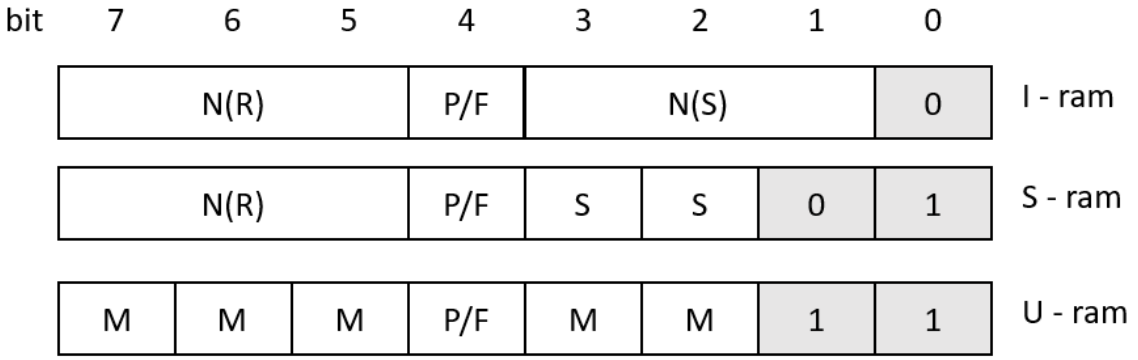


Figure 2.8: Generellt format på kontrollbitarna i informations- (I), övervaknings- (S) och onumrerade ramar (U) så som de dikteras enligt HDLC-standarden [10].

Informationsramen (Information frame) används vid transport av data från nätverkslagret. N(S)-fältet, se figur 2.8, innehåller sekvensnumret och N(R)-fältet används för att skicka bekräftelser (acknowledgements) samt visar vilken ram som enheten förväntar sig att erhålla nästa gång. P/F-biten (Poll/Final) är etta när sändaren förväntar sig ett svar av mottagaren (Poll). Den är även en etta när sändaren indikerar på att det är slutet av överföringen (Final). Annars är P/F-biten en nolla.

Övervakningsramen (Supervisory frame) används för flödes- och felhantering. Det finns fyra olika typer: receive ready (RR), reject (REJ), receive not ready (RNR) och selective reject (SREJ). RR indikerar att sändaren nu kan ta emot mer data. SS bitarna i S-ramen har värdet 00. REJ indikerar att det har inträffat ett fel och att sändaren måste skicka ramar igen, med start vid N(R), SS = 10. RNR informerar sändaren att mottagaren inte kommer att acceptera några mer inkommande ramar, SS = 10. SREJ indikerar att sändaren endast ska skicka ram N(R) igen, SS = 11.

Den onumrerade ramen (Unnumbered frame) används för ett antal kontrollfunktioner, specificerade av M-bitarna. Exempel på kontrollfunktioner är Set Normal Response Mode (SNRM), vilket innebär att en normal uppkoppling sätts upp där 3-bitars sekvensnummer används. Disconnect (DISC) innebär att en RET-enhet önskar att avbryta uppkopplingen. Reset (RSET) återställer mottagarenhetes N(R)-fält men inte N(S)-fältet. Unnumbered Acknowledgment (UA) är det mottagarenheten svarar med.

Procedur-ID är den kommandokod som specificeras i 3GPP TS 25.466 [7]. Antal dataoktetter refererar till längden av ramens informationsdel i oktetter. Vid information av typen nummer eller liknande, till exempel tilt-värde i antal grader, representeras värdena av typen Little-Endian, vilket betyder att den låga oktetten kommer före den höga av ett 16-bitars tal. Cyclic redundancy check (CRC), är en 16-bitars felsökningskod som man utför på alla bitar innanför flaggorna. Det vill säga man tar inte med flaggorna i CRC-beräkningen. När mottagarenheten tar emot ramen gör den en egen CRC-beräkning och jämför den egna CRC-koden med den mottagna. Stämmer de överens så accepteras ramen, annars så kasseras den. CRC beräknas genom binär polynomdivision. Varje sort av CRC-kod har ett eget polynom som används som nämnare i polynomdivisonen.

2.5 3GPP

3GPP är ett projekt som förenar sju stycken organisationer som utvecklar standarder inom telekommunikation. Projektet innehåller tre tekniska specifikationsgrupper (Technical Specification Groups (TSG)) vilka är Radio Access Networks, Services & Systems Aspects och Core Network & Terminals [11]. För detta examensarbete används 3GPP TS 25.466, TS 25.461 och TS 25.462 [7]-[9] vilket är inom gruppen Radio Access Networks. Där ligger intresset i den del av standarden som behandlar lager 1 och lager 7 i OSI-modellen samt signaltransport.

2.5.1 3GPP TS 25.466

3GPP TS 25.466 [7] beskriver hur procedur-ID och tillhörande dataoktetter ska konstrueras. De kommandon som kommer att användas är det generella *Get Information* [7, Kap. 6.5.3] och Single-antenna kommandon som *Calibrate*, *Set Tilt*, *Get Tilt* och *Get Device Data* [7, Kap. 6.6]. Enligt Bo Granstam på Ericsson beter sig seriekopplade RET-enheter som Single-antenna och inte som Multi-antenna, därför används endast kommandon för Single-antenna. Alla kommandon resulterar i minst en returkod som svar. Returkoden innehåller ett OK-svar eller en specifik felkod, lista för felkoder finns i [7, Annex A].

2.5.1.1 Hämta information från RET-enhet

Kommandot *Get Information* har procedur-ID 0x05 och returnerar RET-enhetens produktnummer, serienummer, hårdvaruversion och mjukvaruversion. Om hårdvaruversion eller mjukvaruversion inte hittas eller inte existerar så returneras bara en tom textsträng. Innan varje parameter kommer ett nummer med det antal oktetter den parametern innehåller.

2.5.1.2 Utföra kalibrering av antennport

För kalibrering används kommandot *Calibrate*. Det har procedur-ID 0x31 och genomför kalibrering av antennen, det vill säga att ställdonet går igenom hela tiltspannet för antennen. Svar från RET-enheten ska komma inom 4 minuter. Svar

innehåller endast returkod.

2.5.1.3 Ändra elektriskt tilt-värde på antennport

Set Tilt har procedur-ID 0x33 och sätter den elektriska tilt-vinkeln till ett specificerat värde med 0.1° inkrement. Värdet ska bestå av två oktetter och är av typen Little-Endian. Den maximala responstiden är 2 minuter.

2.5.1.4 Hämta nuvarande elektsiska tilt-värde på antennport

Get Tilt har procedur-ID 0x34 och returnerar RET:ens nuvarande tilt-värde. Detta tilt-värde representeras av två oktetter och är också av typen Little-Endian.

2.5.1.5 Hämta information från antenn

Kommandot *Get Device Data* har procedur-ID 0x0F och returnerar den information som specificeras med den kod som skickas med i kommandot. Koder finns specificerade i [7, Annex B]. De som används för detta examensarbete är:

- 0x02 *Antenna serial number*.
- 0x03 *Antenna operating band*.
- 0x06 *Maximum supported electrical tilt*.
- 0x07 *Minimum supported electrical tilt*.

Antenna serial number returnerar serienumret för antennen. *Antenna operating band* returnerar frekvensbanden för antennen vilket man kan läsa av i 3GPP TS 25.461 [7, Table B.2] där varje satt bit representerar frekvenserna i [8, Kap. 4.3.7]. Om svarets bit 15 och/eller 14 är en etta så måste man även skicka efter 0x08 Antenna operating band och/eller 0x09 Antenna operating band för att få resterande frekvensband. *Maximum supported electrical tilt* och *Minimum supported electrical tilt* returnerar högsta och lägsta tilt-vinkel gånger 10. Dessa värden är av typen Little-Endian.

2.5.2 3GPP TS 25.462

3GPP TS 25.462 [9] går igenom datalänklagret med fokus på XID, vilket innebär att mottagarenheten ska identifiera sig samt ta emot sändarens identifieringsegenskaper. TS 25.462 tar även upp upprättande av kommunikationslänken och djupare beskrivning av HDLC-strukturen.

2.5.2.1 Hitta och tilldela adresser till RET-enheter

För att hitta och adressera RET-enheter utförs XID, som består av två delar, *Device Scan* och *Address Assignment*.

Med kommandot *Device Scan* undersöker man vilka enheter som finns inkopplade. Detta gör man först genom att broadcasta med Unique ID (0x01) och Bitmask (0x03), båda med längd 0, det vill säga tomma textsträngar. Notera att längden på Unique ID och Bitmasken alltid måste vara densamma. Finns det bara en enhet svarar den och man kan gå vidare till Address Assignment. Är det flera enheter så

kan man inte urskilja något i svaret och man måste därför göra ytterligare en XID. Då skickar man Unique ID = Vendor Code hexadecimalt och Bitmask = 0xFFFF. Enheterna maskar de två första oktetterna i sitt egna serienummer med Bitmasken och jämför det med det Unique ID som skickades med. Stämmer detta överens så maskar mottagarenheten den sista oktetten i sitt serienummer med den sista oktetten i Bitmasken och jämför detta med den sista oktetten i Unique ID. Det resulterar i att om man har flera mottagarenheter från samma tillverkare så ska man även skicka med sista siffran i serienumret. När mottagarenheten svarar skickar den med hela sitt serienummer, enhetstyp och tillverkarkod [9, Kap. 4.8.4].

Kommandot *Address Assignment* skickar XID med enhetens serienummer (Unique ID 0x01) och den HDLC-adress (0x02) som ska tilldelas enheten. Mottagarenheten sätter sin HDLC-adress till den som skickades och svarar med sitt serienummer och enhetstyp [9, Kap. 4.8.3].

2.5.2.2 Upprätta anslutning till RET-enheter

När XID har genomförts kan man upprätta länken. Detta gör man genom att skicka SNRM-kommando, som är en U-ram. SNRM-kommandot innehåller inte någon data utan är endast flaggor, adress, kontrollbitar 0x97 och CRC. SNRM gör mottagarenheten redo för kommunikation och återställer sekvensnummer. När mottagarenheten svarar med en UA-ram, kontrollbitar 0x63, är länken upprättad. Får inte mottagarenheten en HDLC-ram som är specifikt adresserad till den, det vill säga ej broadcast, återställer sig enheten efter 3 minuter och uppkopplingen avbryts. För att manuellt koppla ifrån enheten används kommandot DISC, kontrollbitar 0x53.

2.6 Nätverksanalysator

En antenns prestanda karakteriseras av dess S-parametrar. I detta projekt är det 2-ports S-parametrar (Scattering parameters) som är intressanta då mätningarna som görs med nätverksanalysator sker mellan två av antennens portar.

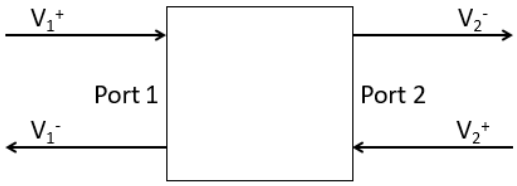


Figure 2.9: Infallande och reflekterade vågor för ett 2-ports-objekt.

S-parametrar för en 2-port beskriver hur infallande och reflekterade vågor påverkar ett objekts portar, se figur 2.9. Detta representeras av fyra stycken parametrar, S_{11}, S_{12}, S_{21} och S_{22} [12]. Termineras port 2 med referensimpedansen Z_0 så fås sambanden,

$$S_{11} = \frac{V_1^-}{V_1^+} \quad \text{och} \quad S_{21} = \frac{V_2^-}{V_1^+}.$$

På liknande sätt om port 1 termineras fås,

$$S_{12} = \frac{V_1^-}{V_2^+} \quad \text{och} \quad S_{22} = \frac{V_2^-}{V_2^+}.$$

S_{11} är reflektionskoefficienten för port 1 medan S_{22} är reflektionskoefficienten för port 2. S_{12} och S_{21} är spänningsförstärkningen på port 1 respektive port 2.

För att mäta S-parametrarna fjärrstyrs nätverksanalysatorn. Detta sker via Saabs egna nätverk där anslutning upprättas med nätverksanalysatorns IP-nummer genom MATLAB. Kommandon skickas för mätning av 2-ports S-parametrar, S_{11}, S_{12}, S_{21} och S_{22}. Dessa kommandon för Remote Control finns specificerade i nätverksanalysatorns användarmanual [13] och är:

- *CALCulate<Ch>:PARameter:SDEFine <TraceName>, <Result>.*
- *MMEMory:STORe:TRACe:CHANnel <Channel>, <TraceFile>.*

CALCulate<Ch>:PARameter:SDEFine <TraceName>, <Result> skapar ett spår (Trace) som den sparar en parameter till, parametern specificeras i <Result>. Vill man mäta S_{11} och spara den i spåret Trc1 på channel 1 så blir kommandot: CALC1:PAR:SDEF 'Trc1','S11'.

MMEMory:STORe:TRACe:CHANnel <Channel>, <TraceFile> exporterar komplexa värden på alla spår i den specificerade kanalen (Channel) till den plats specificerad i TraceFile. Spåren skrivs enligt kataloglistan. Det innebär att den array som returneras innehåller n antal par av reella och imaginära värden för S_{11} sen n antal par av S_{12}, S_{21} och S_{22}.

2.7 MATLAB

Enligt kravspecifikationen i kapitel 1.3 används utvecklingsmiljön MATLAB i detta projekt. MATLAB är ett högnivåspråk optimerat för beräkning och problemlösning inom vetenskap och ingenjörskonst. Språket är matrisbaserat vilket gör det utmärkt för matematiska beräkningar. Det finns ett brett utbud av tillägg (toolboxes) för olika tekniska applikationer. MATLAB har även verktyg för att skapa egna användargränssnitt för sina projekt [14].

2.7.1 GUIDE

För att skapa ett GUI användes MATLAB GUIDE (GUI development enviroment). Den har en Layout Editor, så att man grafiskt kan se alla element i användargränssnittet. Detta gör det lätt att få en bild av hur programmet kommer att se ut samt enkelt tilldela värden och flytta runt element.

3
Programmet

I detta kapitel presenteras resultatet av projektet samt hur programmet fungerar.

3.1 Kommunikation mellan RET och PC

För kommunikation mellan dator och RET-enheter användes en USB till RS-485 adapter [2] och en RET-kontrollkabel [3]. Inställningarna på USB-till-RS485 adaptern var följande: Operation mode: Half Duplex (2 wire) - without Echo, S1 = ON, S2 = OFF och S3 = ON. På 6x3(18-pin) Header Block var endast jumper 1-2 aktiverade, vilket gav Tx+/- med en 120 Ω termision, se figur 3.1.

Figure 3.1: Inställningar för USB-till-RS485 adaptern enligt [2]. Header Block överst och Operation mode S1, S2, S3 underst.

Med hjälp av en multimeter utreddes vilka kablar som tillhörde pinnarna i figur 2.6. RS485A och RS485B kopplades till USB-till-RS485 adapterns Pin 1 (Data-) samt Pin 2 (Data+), se figur 3.2. För 24 V DC-matning användes en Mouser GS25A 24W AC-DC Industrial Adaptor [15]. Figur 2.1 visar hela kopplingen mellan PC och RET-enheter.

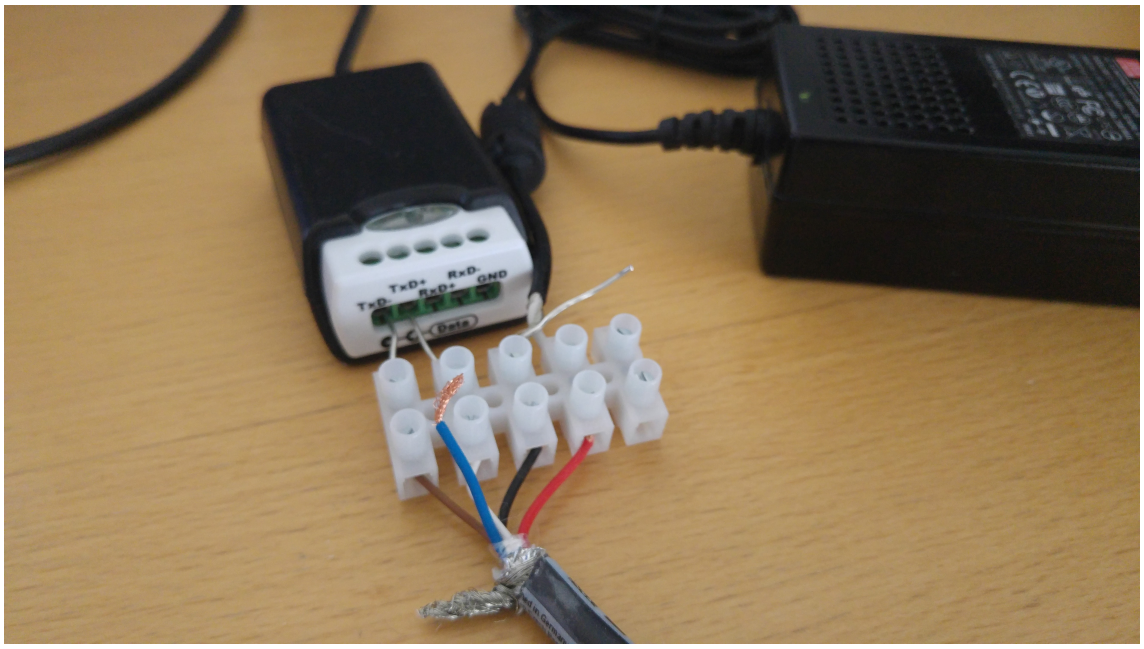


Figure 3.2: Ihopkoppling av USB-till-RS485 adaptern, 24V DC adapter och RET-kabel.

3.2 Konstruktion av HDLC-ramar

Bo Granstam på Ericsson tillhandahöll loggarna i Appendix A samt dummy-antennen. Genom att studera loggarna och med hjälp av en tråd på ett diskussionsforum [16][17] visade det sig att CRC-koden var av typen CRC-CCITT (0xFFFF). Funktion skrevs för att beräkna CRC-koden. Notera att för korrekt beräkning av CRC så måste alla oktetter bitvis reflekterats innan CRC-CCITT (0xFFFF) utförs [17]. Därefter måste 16-bitars koden inverteras och slutligen bitvis reflekterats återigen [16].

Beräkning av kontrollbitarna skrevs så att man väljer vilken sorts protokoll det är som ska skapas och sen beräknas det enligt figur 2.8.

HDLC-ramen sätts ihop genom att sammanfoga alla oktetter, HDLC-adress, kontrollbitar, procedur-ID, datalängd och data, se figur 2.7. CRC beräknades av de tidigare oktetterna och slutligen placerades flaggorna (0x7E) i början och slutet.

Ramarna som skickas konstrueras som en hexadecimal sträng, se Appendix A. Dessa konverteras sedan till en lodrät array av decimaltal, där varje oktett har en plats i arrayen. Det lästa svaret är av samma form, det vill säga en lodrät array av decimaltal. Svaret konverteras sedan om till en hexadecimal sträng för att kunna hanteras på ett enklare sätt.

För att testa kommunikation med en RET-enhet kopplades Comba RCU-003(IV) till Comba dummy-antennen. Alla tester utfördes först på dummy-antennen. För

att sedan kontrollera att kommandona var skrivna enligt standarden gjordes tester på en 4-ports Kathrein-antenn. Där kunde även två RET-enheter anslutas för att undersöka hur XID fungerar med flera enheter.

3.3 Programmets funktioner

Programmets struktur består främst av kommandon och svarshanterare. Kommandon skrivs till ett serieobjekt med hjälp av MATLAB-kommandot `<fwrite>`. Serieobjektet är kopplat till den COM-port som USB-till-RS485 adaptern är kopplad till. För att hitta alla tillgängliga COM-portar körs kommandot `<system('mode')>` som returnerar en sträng med status för alla COM-portar. Med kommandot `<regexp>` och dess indata 'outkey' satt i läge 'mode' extraheras endast namnen på COM-portarna som sedan läggs i en lista.

Svaret läses sedan med `<fread>`, som läser ett specificerat antal värden, eller om inget värde anges läser `<fread>` tills dess att timeout-tiden går ut. Enligt beskrivningen i [9, Kap. 4.5] anges det att en mottagarenhet ska svara inom 10 millisekunder efter att den har fått ett kommando. Därför valdes timeout tiden till 20 millisekunder, som marginal för överförings- samt processtid.

När `<fread>` har lästs så kontrolleras det att svaret ej är en tom sträng. Om så är fallet skickas ett RR-kommando och serieobjektet läses igen. Är svaret inte en tom sträng kontrolleras CRC, samt att det lästa svaret är svaret på det kommando som skickas. Är svaret korrekt skickas det till motsvarande svarshanterare, annars kasseras det. Efter genomfört kommando uppdateras kontrollbitarna.

3.3.1 Anslutning

För att ansluta till RET-enheten används kommandot XID Device Scan, kapitel 2.5.2.1. För att göra enhetsidentifieringen något snabbare så kallas XID med tillverkarkod direkt. Detta väljes ur en lista med alla tillverkarkoder. När en enhet hittas tilldelas den en HDLC-adress med XID Address Assignment, kapitel 2.5.2.1. Första enheten tilldelas 0x01, andra 0x02 etc. Address Assignment gav HDLC-adress, serienummer, tillverkarkod och enhetstyp för varje RET-enhet. Dess värden sparades i cellobjekt med HDLC-adressnummer som index. För att sätta mottagarenheten i anslutningsläge så skickades ett SNRM-kommando. Sista delen i anslutningen var att initiera kontrollbitarna för varje enhet till 0x11, S-ram med P/F-bit satt. Kapitel 2.5.2.2 beskriver att en enhet bryter anslutningen om den inte har fått ett kommando skickat till sin specifika HDLC-adress inom 3 minuter. För att undvika detta skapades ett MATLAB timer-objekt. Det är satt till att efter var 120:e sekund (2 minuter) skicka ett RR-kommando till varje enhet. För att de inte ska kalla samma Callback-funktion samtidigt är det en fördröjning mellan enheternas timer-objekt på 10 sekunder.

3.3.2 Information

För att få information om RET-enheten användes fem funktioner. Alla funktioner körs för varje RET-enhet.

`<getInformation>` använder sig av *Get Information*, kapitel 2.5.1.1, och ger RET-enhetens produktnummer, serienummer, portnummer, hårdvaruversion och mjukvaruversion. Innan varje informationsdel kommer längden av den delen i oktetter. Detta gör det enkelt att få ut rätt information från svaret genom att läsa längden på t.ex. produktnumret, gå fram så många oktetter och sen läsa längden på serienumret etc. Portnumret är de två sista oktetterna i RET-enhetens serienummer.

`<getOperatingBand>` kör kommandot *Get Device Data*: 0x03 *Antenna operating band*, kapitel 2.5.1.5. Svaret är två oktetter (16-bitar) där varje satt bit representerar ett frekvensband. Enligt [7, Table B.2], om bit 14 och/eller 15 är satt så körs kommandot *Get Device Data*: 0x08 *Antenna operating band* och/eller *Get Device Data*: 0x09 *Antenna operating band*. Är bit 15 satt i svaret från *Get Device Data*: 0x09 *Antenna operating band* körs kommandot *Get Device Data*: 0x0A *Antenna operating band*. När alla kommandon är körda sammanställs frekvensbanden och den lägsta och högsta frekvensen blir frekvensbandet för den porten. Detta ska överensstämma med det som står på antennen.

`<getMinMaxTilt>` kör två kommandon, *Get Device Data*: 0x06 *Maximum supported electrical tilt* och *Get Device Data*: 0x07 *Minimum supported electrical tilt*, kapitel 2.5.1.5. Dessa returnerar två oktetter med tilt-värdet. För att värdet ska bli i korrekt format konverteras det från hexadecimalt till decimalt och divideras med 10 [7, Annex B].

`<getTilt>` kör kommandot *Get Tilt*, kapitel 2.5.1.4, och returnerar antennportens nuvarande tilt-värde. Detta värde representeras också av två oktetter och hanteras på samma sätt som för `<getMinMaxTilt>`.

`<getAserial>` kallar på *Get Device Data*: 0x02 *Antenna serial number*, kapitel 2.5.1.5. Svaret är serienumret till antennen som RET-enheten är kopplad till.

3.3.3 Kontroll

Programmet kan kontrollera antennens tilt-värde samt utföra en kalibrering. För detta finns det två funktioner: `<exeCalibrate>` och `<setTilt>`. `<exeCalibrate>` utför *Calibrate*-kommandot, kapitel 2.5.1.2. Efter att kommandot körs så startas en timer på fyra minuter. Funktionen läser serieobjektet kontinuerligt och skickar ett RR-kommando mellan varje läsning. Läsningen avbryts då RET-enheten meddelar att kalibreringen är utförd eller när det har gått fyra minuter. Funktionen returnerar en bekräftelse på att kalibreringen är utförd. Efter kalibreringen körs även *Get Tilt*-kommandot för att en del RET-enheter, i detta fall Kathrein 86010148V01 RCU, går ner till sitt lägsta värde efter att ha utfört en kalibrering. Därefter uppdaterar *Get Tilt* det nuvarande tilt-värdet.

`<setTilt>` använder två kommandon, *Set Tilt*, kapitel 2.5.1.3, och *Get Tilt*. *Set Tilt* tar det tilt-värde som ska skrivas och multiplicerar det med 10 och konverterar det från decimalform till hexadecimal. Liknande `<exeCalibrate>`, startas en timer på två minuter när kommandot körs. RR-kommandon skickas medan tilt-värdet sätts. Då svar returneras om att tilt-värdet är satt, körs *Get Tilt* för att kontrollera att det är det önskade tilt-värdet är satt. Stämmer det önskade värdet överens med det nuvarande värdet returneras en bekräftelse om att tilt-värdet är satt.

3.4 GUI

För att smidigt kunna använda funktionerna skapades ett GUI i MATLAB GUIDE. Fönstret RETcontrol, se figur 3.3, används för hitta, ansluta och visa information om de olika RET-enheterna. Det används även för att ändring av tilt-värde och kalibrering. Fönstret NetworkOpt, se figur 3.6, utför mätning av S-parametrar genom fjärrstyrning av nätverksanalysatorn. Utseendet på gränssnitten har utvecklats efter Saabs önskemål.

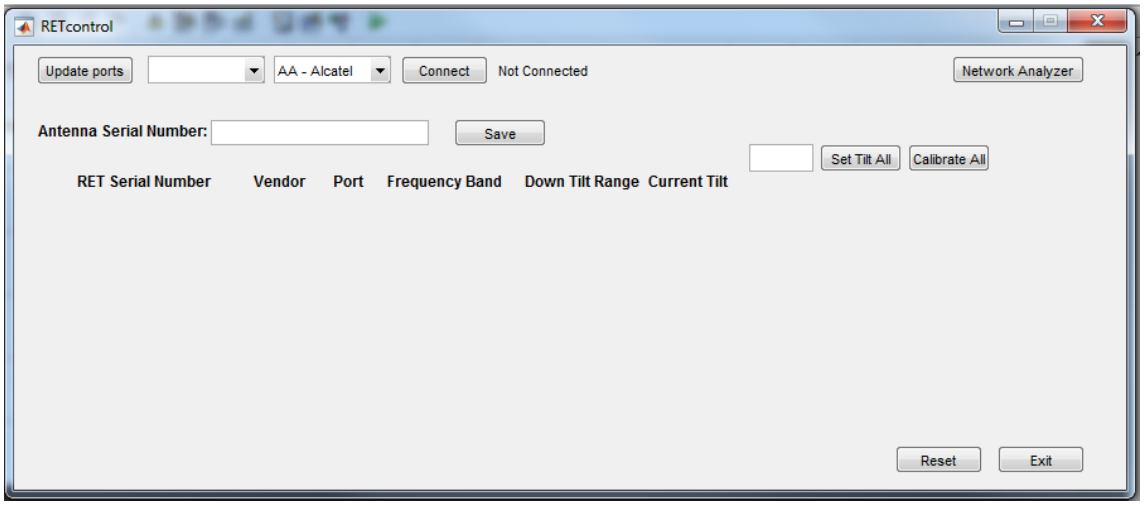


Figure 3.3: Programmets GUI, RETcontrol vid uppstart.

3.4.1 RETcontrol

För att enkelt kunna ansluta till RET-enheten, lades de tillgängliga COM-portarna samt tillverkarkoderna med tillhörande namn i varsitt popupmenu-objekt. En knapp lades till för att kunna uppdatera listan för COM-portar. Anslutningen startades sedan genom att trycka på en knapp med texten ”Connect”. Skulle en enhet redan vara ansluten står det ”Disconnect” och vid knapptryck raderas timer-, serie- och informationsobjekt samt att kommandot DISC skickas till RET-enheten. När anslutning påbörjas körs, som tidigare beskrivet, XID och all information om de olika enheterna samlas. Informationen visas i en rad för varje enhet där RET-enheternas serienummer, tillverkare, portnamn, frekvensband, tilt-spann samt nuvarande tilt-värde visas. Antennens serienummer visas som en rubrik. Det går att markera och

ändra värdet på antennens serienummer, RET-enhetens serienummer och portnamnet. Anslutningstid för en enhet är ca 25 sekunder. För två enheter är den ca 2 minuter.

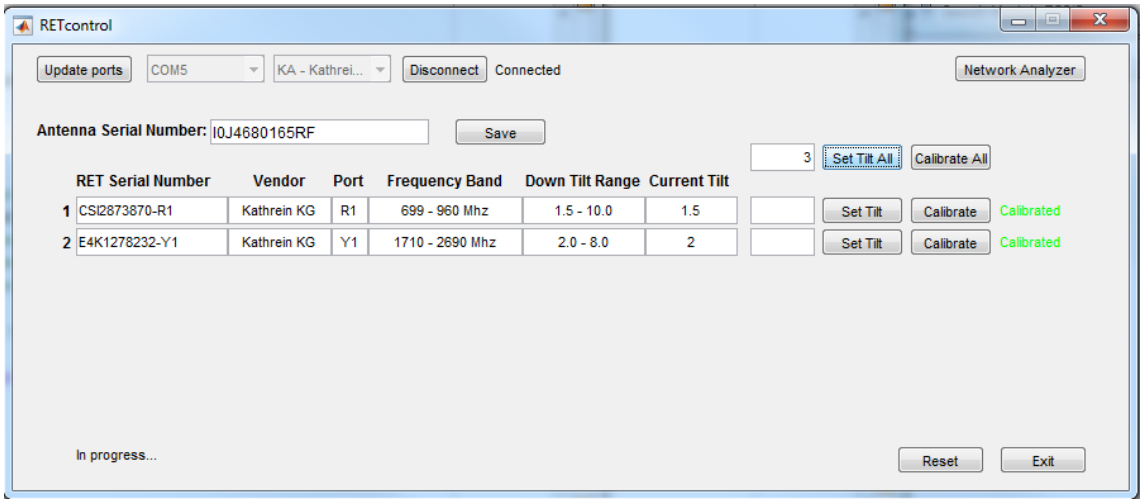


Figure 3.4: Fönstret RETcontrol med två enheter anslutna.

För varje enhet skapas även en knapp för att sätta tilt-värde och en för kalibrering, se figur 3.4. Ett edit-objekt skapas för att kunna ange tilt-värde. Vid nedtryckning av ”Set Tilt”-knappen kontrolleras det att det finns ett värde angivet och att det värdet är inom tilt-spannet. Om det stämmer utför den `<setTilt>` och uppdaterar ”Current Tilt”-objektet.

Till höger om knappen för kalibrering står det vid anslutning av RET-enhet ”Not Calibrated”. När ”Calibrate”-knappen trycks ned utför RET-enheten `<exeCalibrate>` och uppdaterar det nuvarande tilt-värdet. Medan kalibreringen pågår ändras texten till ”In progress...” i svart och när den är klar ändras den till ”Calibrated” och har en grön färg.

Knappar skapades även för att sätta tilt-värden för alla enheter samt kalibrering av alla enheter. ”Set Tilt All” har två valmöjligheter. Sätter man ett värde i rutan för ”Set Tilt All” och då utförs `<setTilt>` av det värdet för alla enheter efter varandra. Om rutan för ”Set Tilt All” lämnas tom och värden för varje enskild enhet specificeras så kommer den att genomföra `<setTilt>` för varje enhet till just den enhetens specifika tilt-värde. Knappen ”Calibrate All” utför kalibrering av alla enheter i ordning.

Knappen ”Save” låter användaren välja en specifik plats och namnet på en fil att spara informationen på. Den utgår från där MATLAB-biblioteket ligger. Filen är av typen .txt och sparar ner antennens serienummer och för varje RET-enheten sparar serienummer, enhetstyp, portnamn, frekvensband och tilt-spann. Se figur 3.5 för ett exempel på en sådan .txt-fil.

```
Antenna serial number: IOJ4680165RF.

RET serial number: CSI2873870-R1
Device type: Single-Antenna RET Device.
Port name: R1.
        Frequency band: 699 - 960 MHz.
        Tilt values: 1.5 - 10.0 degrees.

RET serial number: E4K1278232-Y1
Device type: Single-Antenna RET Device.
Port name: Y1.
        Frequency band: 1710 - 2690 MHz.
        Tilt values: 2.0 - 8.0 degrees.
```

Figure 3.5: Exempel på textdokument som skapas av programmet.

För avslutning av programmet finns tryckknappen ”Exit”. När ”Exit”-knappen trycks ner öppnas en dialogruta som frågar om man vill lämna programmet. Trycker man ”No” återgår man till programmet. Trycker man ”Yes” avslutas alla anslutningar, alla timer- och serie-objektet raderas, DISC-kommando skrivs till RET-enheten och slutligen stängs programmet.

Ifall programmet skulle fastna i en funktion och måste återställas skapades en ”Reset”-knapp. Vid knapptryck skickar sändaren först ett RSET-kommando och sen ett SNRM-kommando till varje ansluten enhet för att återställa sekvensnummer. Sekvensnumret uppdateras till det ursprungliga (0x11) för alla enheter.

För att göra mätningar med nätverksanalysatorn skapades knappen ”Network Analyzer” som kontrollerar att alla anslutna enheter är kalibrerade och att alla enheter har ett portnamn. Sen startar den ett nytt GUI för kommunikation med nätverksanalysatorn. Detta beskrivs vidare i 3.4.2. När det nya GUI:t öppnas uppdateras en variabel som gör att det inte går att öppna ett nytt fönster om man inte först stänger det gamla.

För att få information om när funktioner körs finns det ett flertal textrutor. När funktionen för att söka efter tillgängliga COM-portar körs, visas det en text ”Searching...” under knappen ”Update ports”. Vid anslutning visas texten ”Connecting...” till höger om Connect-knappen. Denna text ändras sedan till ”Connected” när alla enheter är anslutna. Längst ner i vänstra hörnet av programmet finns en textruta för fel-, OK- och framstegmeddelanden. Under anslutning, kalibrering och ändring av tilt-värde ändras även muspekaren till ”watch”-läge som visar att programmet kör en funktion. Fönstret expanderar nedåt när fler än 7 enheter ansluts.

För att kunna använda och uppdatera sekvensnumret i båda användargränssnitten, varje gång en I-ram ska skickas, hämtas data från en variabel som sparas med

`<setappdata>` och hämtas med `<getappdata>`. Denna variabel hanteras som global, det vill säga att man kommer åt den i båda GUI-fönstren.

För att undvika problem med att sändare och mottagare får osynkroniserade sekvensnummer, som resulterar i att programmet kommer att vänta på ett svar i oändlig tid, används ”Reset”-knappens ”Userdata”-variabel. Vid start av programmet sattes ”Userdata” värde till 1 och när ”Reset”-knappen trycks ned ändras värdet på ”Userdata” till 0. I varje funktion som väntar på svar från en RCU-enhet, där programmet skulle kunna fastna, kontrolleras ”Reset”-knappens ”Userdata”. Skulle denna vara 0 kommer funktionen att stoppas. För att programmet ska kunna uppdatera värdet på ”Userdata” medan en funktion körs, skrivs MATLAB-funktionen `<drawnow>` som uppdaterar figurer och bearbetar callbacks. Vid nedtryck av ”Reset”-knappen körs dess callback och därmed avbryts den funktion som körs när ”Reset”-knappen trycks ned.

3.4.2 NetworkOpt

För att ansluta till nätverksanalysatorn trycker man på knappen ”Network Analyzer”. Då öppnas ett nytt GUI, ”networkOpt.fig”, se figur 3.6.

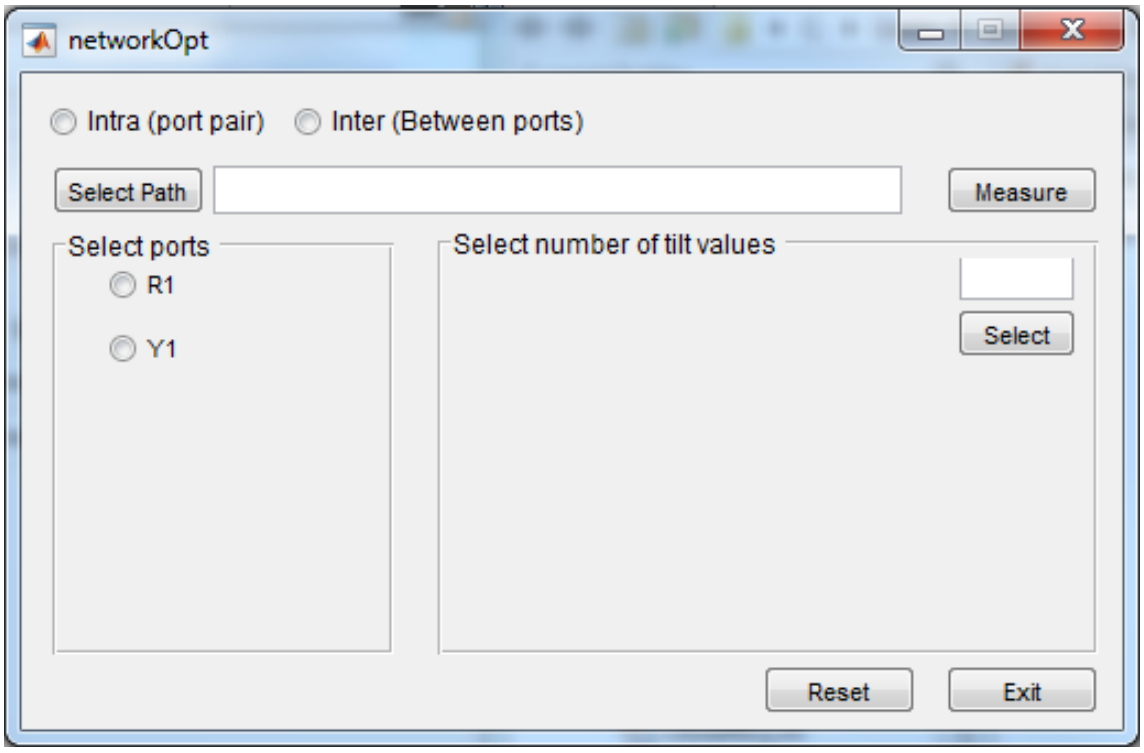


Figure 3.6: Utseende på fönstret ”networkOpt” när det öppnas.

Med radioknappar väljes en av två olika mätmetoder. ”Intra (port pair)” utför mätning av alla S-parametrar för ett enskilt portpar, t.ex. R1+ och R1- eller Y1+ och Y1-. I den vänstra rutan väljes den port som mätningen ska utföras på via radioknappar. Portparsnamnen laddas in från RETcontrol-GUI. För att få tillgång till alla objekt i RETcontrol används MATLAB-funktionerna `<findobj>` och `<guidata>`. Radioknapparna för portparen fungerar enligt följande: trycks en radioknapp i trycks resterade knappar ur.

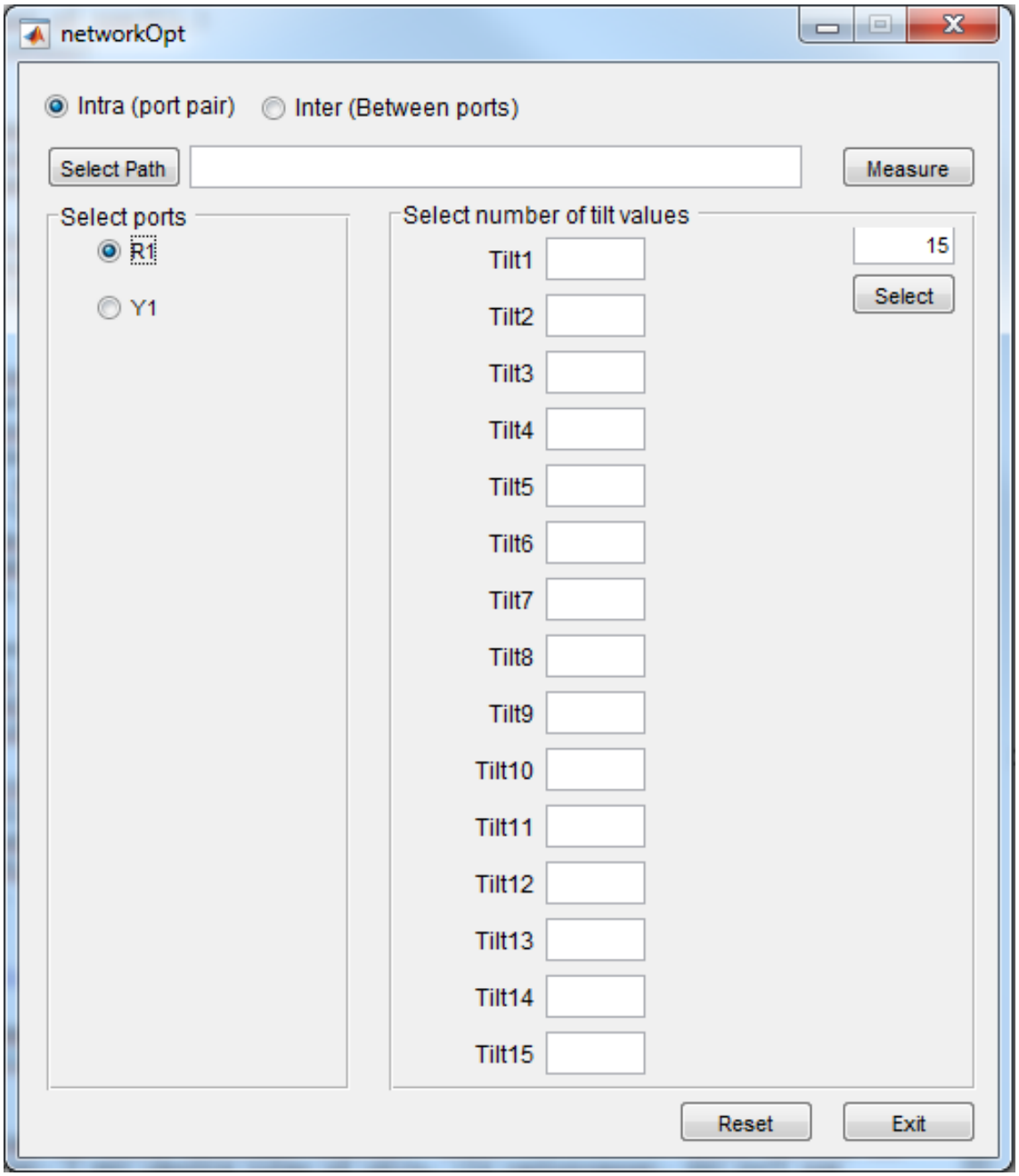


Figure 3.7: Visar val för intra-mätning. Fönstret expanderar då antalet tilt-värden inte får plats i originalfönstret.

I den högra rutan finns ett edit-element och en ”Select”-knapp. Vid knapptryck kontrolleras det att ett tal har angetts. Är det ett tal som är 30 eller lägre skapas så många edit-element, annars visas ett felmeddelande. I dessa edit-element kan man mata in de tilt-vinklar som man vill mäta vid. Då antalet tilt-vinklar blir mer än som får plats expanderar fönstret, se figur 3.7.

”Inter (Between ports)” mäter alla S-parametrar mellan alla portar. Eftersom varje portpar består av en positiv port (+) och en negativ port (-), innebär att vid två anslutna portpar utförs fyra mätningar. En som mäter mellan R1+ och Y1+, en mellan R1- och Y1-, en mellan R1+ och Y1- och slutligen en mellan R1- och Y1+. Tilt-värdet som mäts är endast portparets lägsta tilt-värde, därför avaktiveras radioknapparna för portparen, tilt-vinkelobjekten samt val av antal tilt-vinklar.

För att kunna starta mätning behöver en mapp väljas, till vilken .dat filer för S-parameterna sparas. Då trycker man på ”Select Path” och ett fönster öppnas för att välja mapp. Den har utgångspunkt ifrån MATLAB-biblioteket. När filplatsen är vald visas den i textrutan bredvid.

När knappen ”Measure” trycks ner kontrolleras det att en filplats är vald. Är en intra-mätning vald kontrolleras det att alla tilt-värden har blivit tilldelat ett tal som är inom just det portparets tilt-spann. Om alla värden är giltiga börjar programmet med att sätta det första tilt-värdet med funktionen `<setTilt>`, väntar 10 sekunder för att nätverksanalysatorn ska hinna anpassa sig, och kör sedan funktionen `<getSparam>`. `<getSparam>`-funktionen använder sig av MATLAB-funktioner som Henrik Lindén på Saab har konstruerat för att kommunicera med nätverksanalysatorer via MATLAB inom Saabs nätverk. Kommandona är `<tcpip\_open>` för att öppna kommunikation, `<tcpip\_write>` för att skriva enskilda kommandon, `<tcpip\_read>` för att ta emot svar och `<tcpip\_close>` för att stänga kommunikationen. I detta projekt används de för uppkoppling mot nätverksanalysatorn med ett serieobjekt. Till serieobjektet skrivs ”CALC1:PAR:SDEF” kommandon med `<tcpip\_write>` för varje S-parameter.

Kommandot *MMEMory:STORe:TRACe:CHANnel <Channel>, <TraceFile>* exporterar sedan S-parametrarna till den specificerade platsen. Filnamnet skapas enligt Saabs specifikation: ”AUTIntraSWR_a_b_cT.dat”. Där ”a” är antennens serienummer, ”b” är portparets namn och ”c” är tilt-vinkeln gånger 10 representerat av tre siffror, t.ex. 5° är 050, 10.3° är 103 osv. När första tilt-vinkelns S-parametrar är mätta och sparade, mäter och sparar programmet S-parametrar för resterande tilt-vinklar.

Är det istället en inter-mätning, se figur 3.8, så sker en av två händelser: antingen sätts alla portar till sitt lägsta tilt-värde med funktionen `<setTilt>`, eller om ett värde anges i rutan vid ett portpar i ”Select ports” sätts det angivna tilt-värdet istället för det lägsta. Därefter uppmanas användaren genom ett popupfönster att koppla in två portar enligt exempelmallen, se figur 3.9.

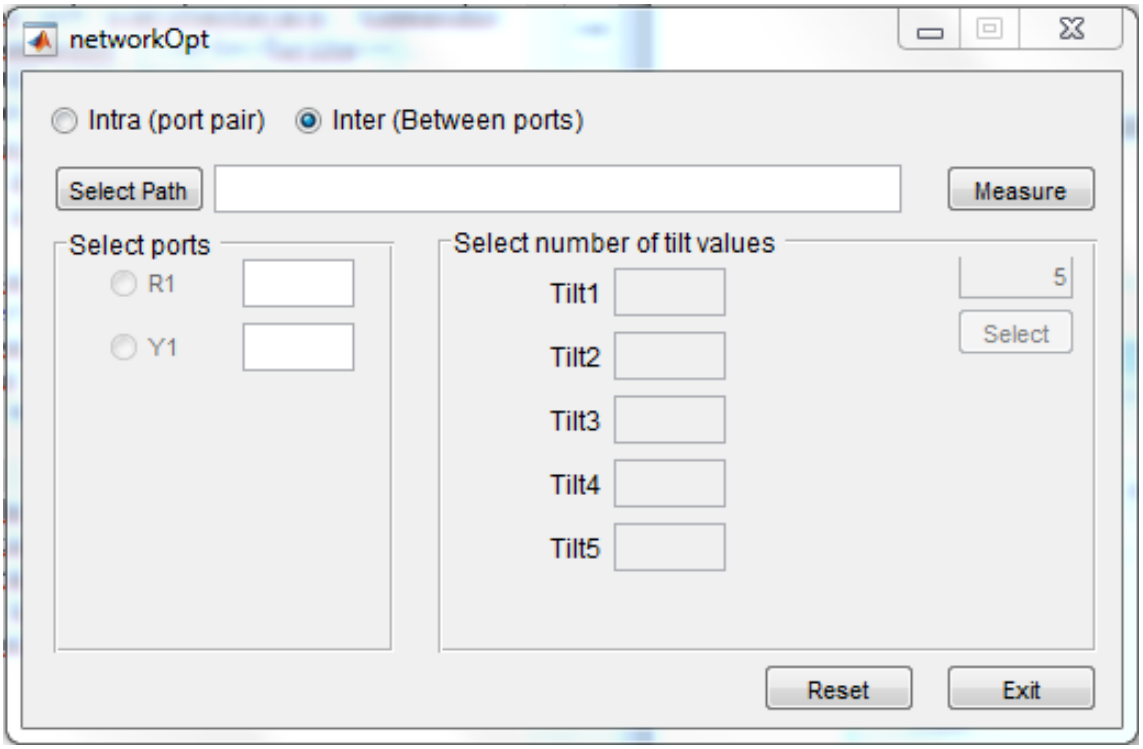


Figure 3.8: Visar val för Intermätning. De andra inställningarna vid intra-mätning är inaktiverade.

	R1+	R1-	R2+	R2-	B1+	B1-	B2+	B2-	Y1+	Y1-	Y2+	Y2-
R1+	NAN	NAN	x	x	x	x	x	x	x	x	x	x
R1-	NAN	NAN	x	x	x	x	x	x	x	x	x	x
R2+	x	x	NAN	NAN	x	x	x	x	x	x	x	x
R2-	x	x	NAN	NAN	x	x	x	x	x	x	x	x
B1+	x	x	x	x	NAN	NAN	x	x	x	x	x	x
B1-	x	x	x	x	NAN	NAN	x	x	x	x	x	x
B2+	x	x	x	x	x	x	NAN	NAN	x	x	x	x
B2-	x	x	x	x	x	x	NAN	NAN	x	x	x	x
Y1+	x	x	x	x	x	x	x	x	NAN	NAN	x	x
Y1-	x	x	x	x	x	x	x	x	NAN	NAN	x	x
Y2+	x	x	x	x	x	x	x	x	x	x	NAN	NAN
Y2-	x	x	x	x	x	x	x	x	x	x	NAN	NAN

Figure 3.9: Figur för hur mätning mellan portpar ska utföras. Endast det gröna fältet är intressant för att undvika att behöva göra dubbla mätningar.

Programmet är pausat med MATLAB `<uiwait>`-funktionen till dess att knappen ”OK” trycks ned. När användaren trycker ”OK”, väntar programmet 10 sekunder som för intra, och sen utför programmet samma S-parametermätning som för intra-mätningen. Enda skillnaden är att filnamnet istället blir:
”AUTInterSWR_a_b1c1T_b2c2T.dat”. Där ”a” är antennens serienummer, ”b1” är första porten t.ex. R1+, ”c1” är första portens tilt-vinkel. ”b2c2” är motsvarande ”b1c1” fast för den andra porten.

”Reset”-knappen i ”networkOpt”-fönstret fungerar på samma sätt som ”RETcontrol”-fönstrets ”Reset”-knapp. Det vill säga att den avbryter funktioner som körs och åter-

ställer sekvensnummer för både sändare och mottagare. ”Exit”-knappen stänger fönstret och uppdaterar variabeln som gör det möjligt att öppna ”networkOpt”-fönstret igen.

4

Diskussion och slutsats

Det tar relativt lång tid att upptäcka enheter och programmet behöver optimeras för att öka användarvänligheten. Inledningsvis krävdes det till exempel kommandon som nu skulle kunna rationaliseras bort. Anslutningstiden för en enhet är cirka 25 sekunder och för två enheter är den cirka 2 minuter. När en enhet ansluts skickas ca 24 HDLC-ramar med timeout-tiden t_{out} emellan, vilket resulterar i en anslutningstid på ungefär 25 sekunder. Med $t_{out} = 1$ sekund betyder detta att cirka 1 sekund av anslutningstiden är beräkningar. När flera enheter ansluts skickas cirka $66 + 23 * n$ ramar, där n är antal enheter. Anledningen till att antalet ramar ökar med en bas på 66 stycken är att XID:s algoritm går igenom hela ASCII-tabellen för att hitta sista tecknet i enheternas serienummer. För två enheter skickas då 112 ramar, med $t_{out} = 1$ sekund vilket blir en tid på 1 min och 52 sekunder. Där blir resterande tid, 8 sekunder, beräkningstid. Har man fler enheter kommer tiden att öka linjärt.

Eftersom man inte alltid vet hur mycket data som ska läsas, beror kommunikationstiden väldigt mycket av serieobjektets timeout-tid. Vid tester med en timeout-tid under 20 millisekunder måste programmet skicka om en hel del information eftersom det går för snabbt för att låta mottagarenheten svara. En timeout-tid på 20 millisekunder är något väl tilltagen men garanterar att flertalet ramar når fram. Dock skickades ramar inte snabbare med en timeout-tid under 1 sekund, vilket innebar att anslutningstiden inte förbättrades.

Projektet har varit en utmaning då majoriteten av dokumentationen om AISG-standarden är det 3GPP har skrivit. Det finns få diskussionstrådar om ämnet och därför varit svårare att skapa sig en förståelse för hur kommunikationsprocessen fungerar. Det har resulterat i att det tagit tid att färdigställa vissa delar, så som beräkning av rätt sorts CRC-kod och Device Scan. Bo Granstam från Ericsson har varit till stor hjälp med detta då han har expertis inom området.

AISG standarden har varit lätt att arbeta med när man väl förstått hur den fungerar. När kommandon väl är skapade är det inga större problem med att skriva funktioner som utför den styrning man är intresserad av. Det man måste tänka på vid kommunikationen är sekvensnummer, som håller reda på i vilken ordning information mellan sändare och mottagare skickas. De gånger programmet fastnat i en funktion beror på att rätt svar inte har erhållits. Detta på grund av att fel sekvensnummer har skickats. Lösningen på problemet har varit funktionerna RSET och SNRM.

Programmets funktioner har skapats med tanken att de ska kunna integreras i andra MATLAB-program. På grund av tidsbrist för att få RESET-knapparna att fungera, konstruerades en lösning med GUI-specifika objekt i funktionerna. Detta resulterade i att funktionerna inte kan användas direkt i andra program. För att underlätta vidareutveckling konstruerades en användarmanual för hur man snabbt kan få funktionerna att fungera utan GUI-objekt.

För att vidareutveckla programmet kan man till exempel försöka optimera Device Scan för att reducera anslutningstiden. Man kan även kunna utveckla GUI för att få det mer användarvänligt då programmet i sin nuvarande version är specifikt konstruerat för de anställda på A15. Trots programmets brister finns det nu en automatiserad lösning för Saab samt funktioner för kommunikation med RET-enheter att bygga vidare på.

Referenser

[1] *Datasheet 80010066V01*, Tyskland: Kathrein. [Online]. Tillgänglig: https://www.kathrein.com/en/solutions/mobile-communication/products/antennas-accessories/outdoor-antennas/?tx_solr%5Bq%5D=80010666v01&id=194&L=0 . Hämtad: 2017-05-04.

[2] *USB-to-RS-422/485 Adapter User's Manual*, Titan Electronics Inc. [Online]. Tillgänglig: http://www.titan.tw/product/manual/TITAN_USB-COMi-TB_USB_to_RS-422_RS-485_with_Terminal_Block_manual.pdf. Hämtad: 2017-04-18.

[3] *Connecting Cable For Remote Electrical Tilt (RET) System*, Tyskland: Kathrein. [Online]. Tillgänglig: https://www.kathrein.de/svg/download/9364533.pdf. Hämtad: 2017-04-18.

[4] B+B SmartWorx, 'Basics of the RS-485 standard,' *bb-elec.com*, 2016. [Online]. Tillgänglig: http://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/Basics-of-the-RS-485-Standard.aspx. Hämtad: 2017-04-07.

[5] AISG, 'Welcome to the AISG Website,' *aisg.org.uk*, 2016. [Online]. Tillgänglig: http://www.aisg.org.uk/. Hämtad: 2017-04-07.

[6] *Antenna Interface Standards Group Standard No. AISG v2.0: Control interface for antenna line devices*, AISG Ltd. 2006. [Online]. Tillgänglig: http://www.aisg.org.uk/IndexDocs/Specifications/AISG%20v2.0%20.pdf. Hämtad: 2017-03-29.

[7] *3GPP TS 25.466 V14.1.0 (2017-01): 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; UTRAN Iuant Interface: Application Part (Release 14)*, 3GPP, 2017. [Online]. Tillgänglig: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1219. Hämtad: 2017-04-10.

[8] *3GPP TS 25.461 V14.1.0 (2017-01): 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; UTRAN Iuant interface: Layer 1 (Release 14),* 3GPP, 2017. [Online]. Tillgänglig: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails

.aspx?specificationId=1216. Hämtad: 2017-04-26.

[9] *3GPP TS 25.462 V14.0.0 (2017-03): 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; UTRAN Iuant interface: Signalling transport (Release 14),* 3GPP, 2017. [Online]. Tillgänglig: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails .aspx?specificationId=1217. Hämtad: 2017-04-26.

[10] A. Leon-Gracia, I. Windjaja, *Communication Networks: Fundamental Concepts and Key Architectures*, Second Edition, McGraw-Hill, 2004.

[11] 3GPP, 'About 3GPP,' *3gpp.org*, 2017. [Online]. Tillgänglig: http://www.3gpp.org/about-3gpp. Hämtad: 2017-04-26.

[12] 'S-paramters,' *Mircowaves101.com*, 2017. [Online]. Tillgänglig: https://www.microwaves101.com/encyclopedias/438-s-parameters-microwave-encyclopedia-microwaves101-com. Hämtad: 2017-06-03.

[13] *R&S® ZNB/ZNBT Vector Network Analyzers: User Manual,* Rohde & Scwarz, 2017. [Online]. Tillgänglig: https://www.rohde-schwarz.com/us/manual/r-s-znb-znbt-user-manual-manuals-gb1_78701-29151.html. Hämtad: 2017-05-12.

[14] The MathWorks, Inc., 'MATLAB,' *mathworks.com*, [Online]. Tillgänglig: https://se.mathworks.com/products/matlab.html. Hämtad: 2017-04-18.

[15] *GS25A series: 25W AC-DC Industrial Adaptor,* Mouser Eletronics, 2015. [Online]. Tillgänglig: http://www.mouser.com/ds/2/260/gs25a-spec-767496.pdf. Hämtad: 2017-04-26.

[16] 'Inquiry About HDLC CRC,' *lammertbies.nl*, [Online]. Tillgänglig: https://www.lammertbies.nl/forum/viewtopic.php?t=607. Hämtad: 2017-04-28.

[17] 'On-line CRC calculation and free library,' *lammertbies.nl*, [Online]. Tillgänglig: https://www.lammertbies.nl/comm/info/crc-calculation.html?crc=00+80+80+FC&method=hex. Hämtad: 2017-04-28.

A

Uppkopplingsloggar

A.1 Uppkopplingslogg Kathrein RET

uppkopplingslogg_RET.txt

```
07:59:28.303  7E FF BF 81 F0 04 01 00 03 00 A6 58 7E
07:59:28.316  U-Frame  XID_CMD
07:59:28.330  7E 00 BF 81 F0 15 01 0C 4B 41 53 41 4D 50 4C 45 36 2D 58 58 04 01 01
06 02 4B 41 4B 0B 7E
07:59:28.361  U-Frame  XID_RESP
07:59:37.146  7E FF BF 81 F0 11 01 0C 4B 41 53 41 4D 50 4C 45 36 2D 58 58 02 01 01
F7 47 7E  07:59:37.173  U-Frame  XID_CMD
07:59:37.176  7E 01 BF 81 F0 11 01 0C 4B 41 53 41 4D 50 4C 45 36 2D 58 58 04 01 01
9C 4D 7E  07:59:37.203  U-Frame  XID_RESP
07:59:45.370  7E 01 93 8D B0 7E  07:59:45.376  U-Frame  SNRM_CMD
07:59:45.386  7E 01 73 83 57 7E  07:59:45.392  U-Frame  UA_RESP
07:59:52.871  7E 01 11 97 17 7E  07:59:52.877  S-Frame NR=0  RR_CMD
07:59:52.890  7E 01 11 97 17 7E  07:59:52.896  S-Frame NR=0  RR_RESP
08:00:00.311  7E 01 10 33 02 00 00 00 7E A1 7E
08:00:00.323  I-Frame   NS=0 NR=0  Set Tilt command
08:00:00.341  7E 01 31 95 36 7E  08:00:00.347  S-Frame NR=1  RR_RESP
08:00:02.961  7E 01 11 97 17 7E  08:00:02.967  S-Frame NR=0  RR_CMD
08:00:02.984  7E 01 30 33 01 00 00 96 14 7E
08:00:02.994  I-Frame   NS=0 NR=1  Set Tilt response  OK
08:00:03.055  7E 01 31 95 36 7E  08:00:03.061  S-Frame NR=1  RR_CMD
08:00:03.086  7E 01 31 95 36 7E  08:00:03.092  S-Frame NR=1  RR_RESP
08:00:13.042  7E 01 31 95 36 7E  08:00:13.048  S-Frame NR=1  RR_CMD
08:00:13.059  7E 01 31 95 36 7E  08:00:13.065  S-Frame NR=1  RR_RESP
08:00:13.304  7E 01 32 31 00 00 C5 6D 7E
08:00:13.313  I-Frame   NS=1 NR=1  Calibrate command
08:00:13.329  7E 01 51 93 55 7E  08:00:13.335  S-Frame NR=2  RR_RESP
08:00:23.135  7E 01 31 95 36 7E  08:00:23.141  S-Frame NR=1  RR_CMD
08:00:23.152  7E 01 52 31 01 00 00 DB 9A 7E
08:00:23.162  I-Frame   NS=1 NR=2  Calibrate response  OK
08:00:23.226  7E 01 51 93 55 7E  08:00:23.232  S-Frame NR=2  RR_CMD
08:00:23.262  7E 01 51 93 55 7E  08:00:23.268  S-Frame NR=2  RR_RESP
08:00:33.228  7E 01 51 93 55 7E  08:00:33.234  S-Frame NR=2  RR_CMD
08:00:33.250  7E 01 51 93 55 7E  08:00:33.256  S-Frame NR=2  RR_RESP
08:00:43.305  7E 01 51 93 55 7E  08:00:43.311  S-Frame NR=2  RR_CMD
08:00:43.313  7E 01 51 93 55 7E  08:00:43.319  S-Frame NR=2  RR_RESP
08:00:53.414  7E 01 51 93 55 7E  08:00:53.420  S-Frame NR=2  RR_CMD
08:00:53.451  7E 01 51 93 55 7E  08:00:53.457  S-Frame NR=2  RR_RESP
08:01:03.492  7E 01 51 93 55 7E  08:01:03.498  S-Frame NR=2  RR_CMD
08:01:03.515  7E 01 51 93 55 7E  08:01:03.521  S-Frame NR=2  RR_RESP
```

```
08:01:13.585  7E 01 51 93 55 7E  08:01:13.591  S-Frame NR=2  RR_CMD
08:01:13.605  7E 01 51 93 55 7E  08:01:13.611  S-Frame NR=2  RR_RESP
08:01:23.695  7E 01 51 93 55 7E  08:01:23.701  S-Frame NR=2  RR_CMD
08:01:23.702  7E 01 51 93 55 7E  08:01:23.708  S-Frame NR=2  RR_RESP
```

A.2 Uppkopplingslogg Comba RET

uppkopplingslogg_Comba_RET.txt

```
09:34:58.269  7E FF BF 81 F0 04 01 00 03 00 A6 58 7E  09:34:58.282  U-Frame  XID_CMD
09:34:58.272  7E 00 BF 81 F0 1C 01 13 43 42 30 30 30 30 30 43 41 31 36 33 30 30 31
33 36 34 36 06 02 43 42 04 01 01 B6 A2 7E  09:34:58.310  U-Frame  XID_RESP
09:35:04.611  7E FF BF 81 F0 18 01 13 43 42 30 30 30 30 30 43 41 31 36 33 30 30 31
33 36 34 36 02 01 01 20 E8 7E  09:35:04.646  U-Frame  XID_CMD
09:35:04.651  7E 01 BF 81 F0 18 01 13 43 42 30 30 30 30 30 43 41 31 36 33 30 30 31
33 36 34 36 04 01 01 22 2D 7E  09:35:04.685  U-Frame  XID_RESP
09:35:07.352  7E 01 93 8D B0 7E  09:35:07.358  U-Frame  SNRM_CMD
09:35:07.362  7E 01 73 83 57 7E  09:35:07.368  U-Frame  UA_RESP
09:35:16.379  7E 01 11 97 17 7E  09:35:16.386  S-Frame NR=0  RR_CMD
09:35:16.383  7E 01 11 97 17 7E  09:35:16.389  S-Frame NR=0  RR_RESP
09:35:26.441  7E 01 11 97 17 7E  09:35:26.447  S-Frame NR=0  RR_CMD
09:35:26.457  7E 01 11 97 17 7E  09:35:26.463  S-Frame NR=0  RR_RESP
09:35:34.981  7E 01 10 05 00 00 2F 3E 7E
09:35:34.990  I-Frame   NS=0, NR=0  Get Information command
09:35:34.996  7E 01 30 05 29 00 00 0B 52 43 55 2D 30 30 33 28 49 56 29 11 30 30
30 30 30 43 41 31 36 33 30 30 31 33 36 34 36 04 56 35 2E 30 04 56 35 2E 33 A1 21 7E
09:35:35.048  I-Frame   NS=0, NR=1  Get Information response  OK
Prod nr : RCU-003IV  SeNo : 00000CA1630013646  HW-Version : V5.0  SW-Version : V5.3
09:35:35.098  7E 01 31 95 36 7E  09:35:35.104  S-Frame NR=1  RR_CMD
09:35:35.119  7E 01 31 95 36 7E  09:35:35.125  S-Frame NR=1  RR_RESP
09:35:36.498  7E 01 31 95 36 7E  09:35:36.504  S-Frame NR=1  RR_CMD
09:35:36.516  7E 01 31 95 36 7E  09:35:36.522  S-Frame NR=1  RR_RESP
09:35:38.369  7E 01 32 31 00 00 C5 6D 7E
09:35:38.378  I-Frame   NS=1, NR=1  Calibrate command
09:35:38.383  7E 01 51 93 55 7E  09:35:38.389  S-Frame NR=2  RR_RESP
09:35:46.560  7E 01 31 95 36 7E  09:35:46.566  S-Frame NR=1  RR_CMD
09:35:46.566  7E 01 51 93 55 7E  09:35:46.572  S-Frame NR=2  RR_RESP
09:35:56.630  7E 01 31 95 36 7E  09:35:56.636  S-Frame NR=1  RR_CMD
09:35:56.636  7E 01 51 93 55 7E  09:35:56.642  S-Frame NR=2  RR_RESP
09:36:06.690  7E 01 31 95 36 7E  09:36:06.696  S-Frame NR=1  RR_CMD
09:36:06.701  7E 01 51 93 55 7E  09:36:06.707  S-Frame NR=2  RR_RESP
09:36:16.749  7E 01 31 95 36 7E  09:36:16.755  S-Frame NR=1  RR_CMD
09:36:16.782  7E 01 51 93 55 7E  09:36:16.788  S-Frame NR=2  RR_RESP
09:36:26.862  7E 01 31 95 36 7E  09:36:26.868  S-Frame NR=1  RR_CMD
09:36:26.878  7E 01 51 93 55 7E  09:36:26.884  S-Frame NR=2  RR_RESP
09:36:36.949  7E 01 31 95 36 7E  09:36:36.955  S-Frame NR=1  RR_CMD
09:36:37.178  7E 01 51 93 55 7E  09:36:37.184  S-Frame NR=2  RR_RESP
09:36:47.230  7E 01 31 95 36 7E  09:36:47.236  S-Frame NR=1  RR_CMD
09:36:47.461  7E 01 51 93 55 7E  09:36:47.467  S-Frame NR=2  RR_RESP
09:36:57.512  7E 01 31 95 36 7E  09:36:57.518  S-Frame NR=1  RR_CMD
09:36:57.518  7E 01 51 93 55 7E  09:36:57.524  S-Frame NR=2  RR_RESP
09:37:07.570  7E 01 31 95 36 7E  09:37:07.576  S-Frame NR=1  RR_CMD
```

```
09:37:07.584  7E 01 51 93 55 7E  09:37:07.590  S-Frame NR=2  RR_RESP
09:37:17.640  7E 01 31 95 36 7E  09:37:17.646  S-Frame NR=1  RR_CMD
09:37:17.658  7E 01 51 93 55 7E  09:37:17.664  S-Frame NR=2  RR_RESP
09:37:27.712  7E 01 31 95 36 7E  09:37:27.718  S-Frame NR=1  RR_CMD
09:37:27.712  7E 01 52 31 01 00 00 DB 9A 7E
09:37:27.722  I-Frame   NS=1, NR=2  Calibrate response  OK
09:37:27.778  7E 01 51 93 55 7E  09:37:27.784  S-Frame NR=2  RR_CMD
09:37:27.818  7E 01 51 93 55 7E  09:37:27.824  S-Frame NR=2  RR_RESP
09:37:36.201  7E 01 54 33 02 00 0A 00 73 4E 7E
09:37:36.212  I-Frame   NS=2, NR=2  Set Tilt command
09:37:36.222  7E 01 71 91 74 7E  09:37:36.228  S-Frame NR=3  RR_RESP
09:37:37.779  7E 01 51 93 55 7E  09:37:37.785  S-Frame NR=2  RR_CMD
09:37:37.799  7E 01 71 91 74 7E  09:37:37.805  S-Frame NR=3  RR_RESP
09:37:47.851  7E 01 51 93 55 7E  09:37:47.858  S-Frame NR=2  RR_CMD
09:37:47.863  7E 01 74 33 01 00 00 A4 F8 7E
09:37:47.873  I-Frame   NS=2, NR=3  Set Tilt response  OK
09:37:47.929  7E 01 71 91 74 7E  09:37:47.935  S-Frame NR=3  RR_CMD
09:37:47.970  7E 01 71 91 74 7E  09:37:47.976  S-Frame NR=3  RR_RESP
09:37:57.932  7E 01 71 91 74 7E  09:37:57.938  S-Frame NR=3  RR_CMD
09:37:57.953  7E 01 71 91 74 7E  09:37:57.959  S-Frame NR=3  RR_RESP
09:38:07.992  7E 01 71 91 74 7E  09:38:07.998  S-Frame NR=3  RR_CMD
09:38:08.005  7E 01 71 91 74 7E  09:38:08.011  S-Frame NR=3  RR_RESP
09:38:18.042  7E 01 71 91 74 7E  09:38:18.048  S-Frame NR=3  RR_CMD
09:38:18.053  7E 01 71 91 74 7E  09:38:18.059  S-Frame NR=3  RR_RESP
```

B

MATLAB-kod

Listing B.1: RETcontrol

```
function varargout = RETcontrol(varargin)
% untitled MATLAB code for untitled.fig
%      untitled, by itself, creates a new untitled or raises the existing
%      singleton*.
%
%      H = untitled returns the handle to a new untitled or the handle to
%      the existing singleton*.
%
%      untitled('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in untitled.M with the given input arguments.
%
%      untitled('Property','Value',...) creates a new untitled or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before untitled_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to untitled_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help untitled

% Last Modified by GUIDE v2.5 16-May-2017 10:06:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @RETcontrol_OpeningFcn, ...
    'gui_OutputFcn',  @RETcontrol_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before RETcontrol is made visible.
function RETcontrol_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RETcontrol (see VARARGIN)

% Choose default command line output for RETcontrol
handles.output = hObject;
set(handles.connectedtxt,'string',' Not Connected');
% Vendor list.
vnlist = {'AA' 'AC' 'AD' 'AE' 'AL' 'AI' 'AM' 'AN' 'AT' 'AR' 'AV' 'BW' 'CB' 'CC' 'CM' 'CT' 'CS' '
    CX' 'DA' 'DB' 'EB' 'EM' 'ET' 'ER' 'EY' 'FI' 'FO' 'FR' 'GN' 'GR' 'HI' 'HW' 'HS' 'JB' 'JQ' 'KA
    ' 'KL' 'KM' 'LA' 'LG' 'LU' 'MA' 'MI' 'MO' 'MT' 'MY' 'NK' 'NN' 'PO' 'PW' 'QU' 'RA' 'RC' 'RE'
    'RF' 'RY' 'SE' 'SH' 'SI' 'SM' 'SP' 'SU' 'TH' 'UW' 'VX' 'XH'};
a = cell(1,length(vnlist));
% Adds vendor name.
for i = 1:length(vnlist)
    k = getVendor(vnlist{i});
    a{i} = strcat(vnlist{i},32,'-',32,k);
end
set(handles.vendorlist,'String',a);
set(handles.rsetBtn,'userdata',1)
setappdata(0,'fig2',1);
handles.conOK = 0;
```

```
70  % Update handles structure
71  guidata(hObject, handles);
72
73  % UIWAIT makes RETcontrol wait for user response (see UIRESUME)
74  % uiwait(handles.Gui1);
75
76  % --- Outputs from this function are returned to the command line.
77  function varargout = RETcontrol_OutputFcn(hObject, eventdata, handles)
78  % varargout  cell array for returning output args (see VARARGOUT);
79  % hObject    handle to figure
80  % eventdata  reserved - to be defined in a future version of MATLAB
81  % handles    structure with handles and user data (see GUIDATA)
82
83  % Get default command line output from handles structure
84  varargout{1} = handles.output;
85
86  % --- Executes on selection change in comList.
87  function comList_Callback(hObject, eventdata, handles)
88  % hObject    handle to comList (see GCBO)
89  % eventdata  reserved - to be defined in a future version of MATLAB
90  % handles    structure with handles and user data (see GUIDATA)
91
92  % Hints: contents = cellstr(get(hObject,'String')) returns comList contents as cell array
93  %        contents{get(hObject,'Value')} returns selected item from comList
94
95
96  % --- Executes during object creation, after setting all properties.
97  function comList_CreateFcn(hObject, eventdata, handles)
98  % hObject    handle to comList (see GCBO)
99  % eventdata  reserved - to be defined in a future version of MATLAB
100 % handles    empty - handles not created until after all CreateFcns called
101
102 % Hint: popupmenu controls usually have a white background on Windows.
103 %       See ISPC and COMPUTER.
104 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
105     set(hObject,'BackgroundColor','white');
106 end
107
108
109
110 % --- Executes on button press in connectBtn.
111 function connectBtn_Callback(hObject, eventdata, handles)
112 % hObject    handle to connectBtn (see GCBO)
113 % eventdata  reserved - to be defined in a future version of MATLAB
114 % handles    structure with handles and user data (see GUIDATA)
115 handles = guidata(hObject);
116 % Checks if the RETs are allready connected.
117 if ~handles.conOK
118     set(handles.connectedtxt,'string','Connecting... Takes some time (+2 min).');
119     set(handles.tiltinfo,'string','');
120     % Waiting animation of the pointer.
121     oldpointer = get(handles.Gui1, 'pointer');
122     set(handles.Gui1, 'pointer', 'watch')
123     drawnow;
124
125     % Extracts the selected COM port.
126     allItems = get(handles.comList,'string');
127     selectedIndex = get(handles.comList,'Value');
128     selectedItem = allItems{selectedIndex};
129     % Exctracts the selected Vendor code
130     aI = get(handles.vendorlist,'string');
131     sIndex = get(handles.vendorlist,'Value');
132     Vnfull = aI{sIndex};
133     Vn = Vnfull(1:2);
134     % Tries to connect to the RET.
135     try
136         set(handles.rsetBtn,'userdata',1);
137         % Creates a serial Object and executes the XID command.
138         set(handles.tiltinfo,'string','Scaning for devices...');
139         [Addr,SN,VnCd,DeTy,slast,RET] = openConnection(selectedItem,Vn,handles.rsetBtn);
140         % Gets the Serial number of the antenna connected to the RET.
141         set(handles.tiltinfo,'string','Devices found.');
142         % Preallocation
143         ProdNr = cell(1,length(Addr));
144         SerNrRET = cell(1,length(Addr));
145         HwVer = cell(1,length(Addr));
146         SwVer = cell(1,length(Addr));
147         port = cell(1,length(Addr));
148         min = cell(1,length(Addr));
149         max = cell(1,length(Addr));
150         minT = cell(1,length(Addr));
151         maxT = cell(1,length(Addr));
152         cval = cell(1,length(Addr));
153
154         for i = 1:length(Addr)
155             % Checks if reset button is pressed.
156             if get(handles.rsetBtn,'userdata')
157                 % Calls the GetInformation command.
158                 s = sprintf('Collecting information for device %d.',i);
159                 set(handles.tiltinfo,'string',s);
160                 [ProdNr{i},SerNrRET{i},HwVer{i},SwVer{i},port{i},slast{i}] = getInformation(Addr
                        {i},slast{i},RET,handles.rsetBtn);
161                 % Calls the GetOperatingBands command.
162                 s = sprintf('Collecting operrating bands for device %d.',i);
163                 set(handles.tiltinfo,'string',s);
```

```
164                 [min{i},max{i},slast{i}] = getOperatingBand(Addr{i},slast{i},RET,handles.rsetBtn
                        );
165                 % Calls the Min/Max Tilt value command.
166                 s = sprintf('Collecting min/max Tilt for device %d.',i);
167                 set(handles.tiltinfo,'string',s);
168                 [minT{i},maxT{i},slast{i}] = getMinMaxTilt(Addr{i},slast{i},RET,handles.rsetBtn)
                        ;
169                 % Gets the current tilt value.
170                 s = sprintf('Collecting current Tilt for device %d.',i);
171                 set(handles.tiltinfo,'string',s);
172                 [cval{i},slast{i}] = getTiltGui(Addr{i},slast{i},RET,handles.rsetBtn);
173                 % Gets the Antenna Serial number.
174                 s = sprintf('Collecting Antenna serial number for device %d.',i);
175                 set(handles.tiltinfo,'string',s);
176                 [SerNrAn,slast{i}] = getAserial(Addr{i},slast{i},RET,handles.rsetBtn);
177             end
178             set(handles.tiltinfo,'string','Finishing connection.');
179         end
180         k = 236;
181         % Checks if reset button is pressed.
182         if get(handles.rsetBtn,'userdata')
183             for i = 1:length(Addr)
184                 % Controls if row is outside of the window.
185                 if k-25*i <= 65
186                     k = k + 25;
187                     % Move all obejcts
188                     a = get(handles.Gui1,'position');
189                     set(handles.Gui1,'Units','pixels','position',[a(1) a(2)-25 a(3) a(4)+25]);
190                     changepos(handles.getCombtn,25);
191                     changepos(handles.comList,25);
192                     changepos(handles.vendorlist,25);
193                     changepos(handles.connectBtn,25);
194                     changepos(handles.portTx,25);
195                     changepos(handles.connectedtxt,25);
196                     changepos(handles.text33,25);
197                     changepos(handles.AnSer,25);
198                     changepos(handles.saveBtn,25);
199                     changepos(handles.text34,25);
200                     changepos(handles.text35,25);
201                     changepos(handles.text36,25);
202                     changepos(handles.text37,25);
203                     changepos(handles.text38,25);
204                     changepos(handles.text39,25);
205                     changepos(handles.tiltAlltxt,25);
206                     changepos(handles.tiltall,25);
207                     changepos(handles.Cal_all,25);
208                     for j = 1:i-1
209                         % Move each row.
210                         changepos(handles.nr(j),25);
211                         changepos(handles.retSer(j),25);
212                         changepos(handles.Vendor(j),25);
213                         changepos(handles.portNa(j),25);
214                         changepos(handles.freqBand(j),25);
215                         changepos(handles.tiltRange(j),25);
216                         changepos(handles.cTilt(j),25);
217                         changepos(handles.setTilttxt(j),25);
218                         changepos(handles.stiltbtn(j),25);
219                         changepos(handles.calibtn(j),25);
220                         changepos(handles.calitxt(j),25);
221                     end
222                 end
223                 % Creates a row of information and tilt/calibrate button for each RET.
224                 handles.nr(i) = uicontrol('Style','text','HorizontalAlignment','right','FontSize
                        ',9.0,'FontWeight','bold',...
225                     'String',sprintf('%d',i),'Position',[32 (k+4)-25*i 16 14]);
226                 handles.retSer(i) = uicontrol('Style','edit','String',SerNrRET{i},'
                        HorizontalAlignment','left','Position',[51 k-25*i 120 22]);
227                 handles.Vendor(i) = uicontrol('Style','edit','String',getVendor(VnCd{i}),'Enable
                        ','inactive','Position',[171 k-25*i 82 22]);
228                 handles.portNa(i) = uicontrol('Style','edit','String',port{i},'Position',[253 k
                        -25*i 29 22]);
229                 handles.freqBand(i) = uicontrol('Style','edit','String',strcat(sprintf('%d',min{
                        i}),32,'-',32,sprintf('%d',max{i}),32,'Mhz'),'Enable','inactive','Position'
                        ,[282 k-25*i 122 22]);
230                 handles.tiltRange(i) = uicontrol('Style','edit','String',strcat(sprintf('%1.1f',
                        minT{i}),32,'-',32,sprintf('%1.1f',maxT{i})),'Enable','inactive','Position'
                        ,[404 k-25*i 97 22]);
231                 handles.cTilt(i) = uicontrol('Style','edit','String',cval{i},'Enable','inactive'
                        ,'Position',[501 k-25*i 73 22]);
232
233                 handles.setTilttxt(i) = uicontrol('Style','edit','String','','
                        HorizontalAlignment','right','Position',[584 k-25*i 51 22]);
234                 handles.stiltbtn(i) = uicontrol('Style','pushbutton','String','Set Tilt','Tag',
                        sprintf('stiltbtn%d',i),'Position',[640 k-25*i 65 22],...
235                     'Callback',@(hObject,eventdata)RETcontrol('setTiltBtn_Callback',hObject,
                            eventdata,guidata(hObject)));
236                 handles.calibtn(i) = uicontrol('Style','pushbutton','String','Calibrate','Tag',
                        sprintf('calibtn%d',i),'Position',[710 k-25*i 65 22],...
237                     'Callback',@(hObject,eventdata)RETcontrol('calibrateBtn_Callback',hObject,
                            eventdata,guidata(hObject)));
238                 handles.calitxt(i) = uicontrol('Style','text','String','Not Calibrated','
                        HorizontalAlignment','left','ForegroundColor','red','Position',[780 (k+4)
                        -25*i 70 14]);
239             end
240
```

```
241             % Disable the comlist and resets the pointer.
242             set(handles.vendorlist,'Enable','off');
243             set(handles.comList,'Enable','off');
244             set(handles.connectedtxt,'string','Connected');
245             set(handles.Gui1, 'pointer', oldpointer)
246             set(handles.AnSer,'string',SerNrAn);
247
248             % Creates timer objects to keep the link established longer than 3
249             % min.
250
251             for i = 1:length(Addr)
252                 t(i) = timer;
253                 t(i).ExecutionMode = 'fixedRate';
254                 % Calls the polling function every 120s (2 min).
255                 t(i).Period = 90+i*10;
256                 t(i).TimerFcn = {@callbackDelay,Addr{i},slast{i},RET};
257                 handles.t(i) = t(i);
258                 start(t(i))
259             end
260             % Creates appdata var with sequence number.
261             setappdata(0,'MyStruct',slast);
262             % Updates the hObjects.
263             handles.portN1 = port;
264             handles.SerNrAn = SerNrAn;
265             handles.SerNrRET = SerNrRET;
266             handles.DeTy = DeTy;
267             handles.min = min;
268             handles.max = max;
269             handles.minT = minT;
270             handles.maxT = maxT;
271             handles.Addr = Addr;
272             handles.RET = RET;
273             handles.slast = slast;
274             handles.conOK = 1;
275             set(hObject,'string','Disconnect')
276             set(handles.tiltinfo,'string','All devices connected.');
277         else
278             set(handles.connectedtxt,'string','Not connected');
279             set(handles.tiltinfo,'string','Program stopped.');
280         end
281     catch
282         %If not availble to connect, error message.
283         msgbox('Wrong COM port or Vendor .','Error')
284         set(handles.connectedtxt,'string','Not connected')
285         set(handles.Gui1, 'pointer', oldpointer)
286     end
287
288 else
289     % If disconnect
290     for i = 1:length(handles.Addr)
291         % Disconnect each device.
292         DISC(handles.Addr{i},handles.RET);
293         stop(handles.t(i))
294         delete(handles.t(i))
295     end
296     % Delete all rows of information.
297     closeConnection(handles.RET);
298     delete(handles.t);
299     delete(handles.nr);
300     delete(handles.retSer);
301     delete(handles.Vendor);
302     delete(handles.portNa);
303     delete(handles.freqBand);
304     delete(handles.tiltRange);
305     delete(handles.cTilt);
306     delete(handles.setTilttxt);
307     delete(handles.stiltbtn);
308     delete(handles.calibtn);
309     delete(handles.calitxt);
310     set(hObject,'string','Connect')
311     set(handles.vendorlist,'Enable','on');
312     set(handles.comList,'Enable','on');
313     handles.conOK = 0;
314 end
315
316 guidata(hObject,handles);
317
318
319 % --- Executes on button press in ExitBtn.
320 function ExitBtn_Callback(hObject, eventdata, handles)
321 % hObject    handle to ExitBtn (see GCBO)
322 % eventdata  reserved - to be defined in a future version of MATLAB
323 % handles    structure with handles and user data (see GUIDATA)
324 handles = guidata(hObject);
325 % Question idalog for exiting program
326 selection = questdlg('Exit program?',...
327     'Exit',...
328     'Yes','No','Yes');
329 switch selection
330     case 'Yes'
331         try
332             % If connected: disconnect.
333             if handles.conOK == 1
334                 for i = 1:length(handles.Addr)
335                     DISC(handles.Addr{i},handles.RET);
```

```
                    stop(handles.t(i))
                    delete(handles.t(i))
                end
                closeConnection(handles.RET);
                handles.conOk = 0;
                delete(handles.t)
            end
            guidata(hObject,handles);
            close all;
        catch
            guidata(hObject,handles);
            close all;
        end
    case 'No'
        return
end


% --- Executes on button press in saveBtn.
function saveBtn_Callback(hObject, eventdata, handles)
% hObject    handle to saveBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);
% Gets the serial number.
SerNrAn = get(handles.AnSer,'string');
string = sprintf('Antenna serial number: %s.\n',SerNrAn);
% Gets the information of each RET.
for i = 1:length(handles.Addr)
    portNr = get(handles.portNa(i),'string');
    SerNrRET = get(handles.retSer(i),'string');
    string1 = sprintf('\n\nRET serial number: %s\nDevice type: %s.\n',SerNrRET,handles.DeTy{i});
    strport = sprintf('\nPort name: %s.\n\tFrequency band: %d - %d MHz.\n\tTilt values: %1.1f -
        %1.1f degrees.\n',portNr,handles.min{i},handles.max{i},handles.minT{i},handles.maxT{i});
    string = strcat(string,string1,strport);
end
% Creates the file path.
[file,path] = uiputfile('info.txt','Save information');
path_file = fullfile(path,file);
% Writes the infromation to the specified path.
fobj = fopen(path_file,'wt');
fprintf(fobj,string);
fclose(fobj);

guidata(hObject,handles);


% --- Executes on button press in Cal_all.
function Cal_all_Callback(hObject, eventdata, handles)
% hObject    handle to Cal_all (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);
try
    % Resets the reset var.
    set(handles.rsetBtn,'userdata',1);
    % Calibrates each of the connected RETs.
    for i = 1:length(handles.Addr)
        if get(handles.rsetBtn,'userdata')
            % Updates the sequence number.
            a = getappdata(0,'MyStruct');
            handles.slast = a;

            s = sprintf('Calibrate device %d.',i);
            set(handles.tiltinfo,'string',s);
            % Calibrate function.
            handles.slast{i} = calibrateGUI(handles.calitxt(i),handles.Gui1,handles.conOK,
                handles.Addr{i},handles.slast{i},handles.RET,handles.rsetBtn);
            s = sprintf('Updating device %d tilt value.',i);
            set(handles.tiltinfo,'string',s);
            % Get Tilt function.
            [g,handles.slast{i}] = getTiltGui(handles.Addr{i},handles.slast{i},handles.RET,
                handles.rsetBtn);
            set(handles.cTilt(i),'string', g);
            % Updates the sequence number.
            setappdata(0,'MyStruct',handles.slast);
        end
    end
    set(handles.tiltinfo,'string','All devices are calibrated.');
catch
    set(handles.tiltinfo,'string','Error: Not Calibrated.');
end
guidata(hObject,handles);

% --- Executes on button press in calibrateBtn.
function calibrateBtn_Callback(hObject, eventdata, handles)
% hObject    handle to calibrateBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);
try
    % Get which RET is going to be calibrated.
    i = str2double(hObject.Tag(end));

    s = sprintf('Calibrate device %d.',i);
```

```
428     set(handles.tiltinfo,'string',s);
429     % Resets the reset button.
430     set(handles.rsetBtn,'userdata',1);
431     % Updates the sequence number.
432     a = getappdata(0,'MyStruct');
433     handles.slast = a;
434
435     % Calls the calibrate command.
436     handles.slast{i} = calibrateGUI(handles.calitxt(i),handles.Gui1,handles.conOK,handles.Addr{i
            },handles.slast{i},handles.RET,handles.rsetBtn);
437     s = sprintf('Updating device %d tilt value.',i);
438     set(handles.tiltinfo,'string',s);
439     % Get Tilt function.
440     [g,handles.slast{i}] = getTiltGui(handles.Addr{i},handles.slast{i},handles.RET,handles.
            rsetBtn);
441     set(handles.cTilt(i),'string', g);
442     % Updates the sequence number.
443     setappdata(0,'MyStruct',handles.slast);
444     s = sprintf('Device %d calibrated.',i);
445     set(handles.tiltinfo,'string',s);
446 catch
447     set(handles.tiltinfo,'string','Error: Not Calibrated.');
448 end
449 guidata(hObject,handles);
450
451 % --- Executes on button press in tiltall.
452 function tiltall_Callback(hObject, eventdata, handles)
453 % hObject    handle to tiltall (see GCBO)
454 % eventdata  reserved - to be defined in a future version of MATLAB
455 % handles    structure with handles and user data (see GUIDATA)
456 handles = guidata(hObject);
457 value = get(handles.tiltAlltxt,'string');
458 % Reset reset button.
459 set(handles.rsetBtn,'userdata',1);
460 try
461     % Check for value.
462     if ~isempty(value)
463         for i = 1:length(handles.Addr)
464             % Check if reset button is pressed.
465             if get(handles.rsetBtn,'userdata')
466                 % Updates the sequence number.
467                 a = getappdata(0,'MyStruct');
468                 handles.slast = a;
469                 set(handles.setTilttxt(i),'string','');
470                 s = sprintf('Setting device %d tilt value.',i);
471                 set(handles.tiltinfo,'string',s);
472                 % Set Tilt function
473                 handles.slast{i} = setTiltGUI(handles.tiltinfo,handles.cTilt(i),handles.Gui1,
                        handles.conOK,value,handles.Addr{i},handles.slast{i},handles.RET,handles.
                        minT{i},handles.maxT{i},handles.rsetBtn);
474                 % Updates the sequence number.
475                 setappdata(0,'MyStruct',handles.slast);
476                 s = sprintf('Device %d tilt value set.',i);
477                 set(handles.tiltinfo,'string',s);
478             end
479         end
480     else
481         for i = 1:length(handles.Addr)
482             % Check of reset button is pressed.
483             if get(handles.rsetBtn,'userdata')
484                 % Updates the sequence number.
485                 a = getappdata(0,'MyStruct');
486                 handles.slast = a;
487                 % Get the own tilt value.
488                 value1 = get(handles.setTilttxt(i),'string');
489                 s = sprintf('Setting device %d tilt value.',i);
490                 set(handles.tiltinfo,'string',s);
491                 % Set Tilt function.
492                 handles.slast{i} = setTiltGUI(handles.tiltinfo,handles.cTilt(i),handles.Gui1,
                        handles.conOK,value1,handles.Addr{i},handles.slast{i},handles.RET,handles.
                        minT{i},handles.maxT{i},handles.rsetBtn);
493                 set(handles.setTilttxt(i),'string','');
494                 % Updates the sequence number.
495                 setappdata(0,'MyStruct',handles.slast);
496                 s = sprintf('Device %d tilt value set.',i);
497                 set(handles.tiltinfo,'string',s);
498             end
499         end
500     end
501     set(handles.tiltAlltxt,'string','');
502 catch
503     set(handles.tiltinfo,'string','Error: Tilt not set.');
504 end
505 guidata(hObject,handles);
506
507 % --- Executes on button press in setTiltBtn.
508 function setTiltBtn_Callback(hObject, eventdata, handles)
509 % hObject    handle to setTiltBtn (see GCBO)
510 % eventdata  reserved - to be defined in a future version of MATLAB
511 % handles    structure with handles and user data (see GUIDATA)
512 handles = guidata(hObject);
513 try
514     set(handles.tiltinfo,'string','Setting tilt...');
515     % Get the device number.
516     i = str2double(hObject.Tag(end));
```

```
517     % Resets the reset button
518     set(handles.rsetBtn,'userdata',1);
519     % Updates the sequence number.
520     a = getappdata(0,'MyStruct');
521     handles.slast = a;
522
523     % Gets the desired tilt value.
524     value = get(handles.setTilttxt(i),'string');
525     % Calls the setTilt command.
526     handles.slast{i} = setTiltGUI(handles.tiltinfo,handles.cTilt(i),handles.Gui1,handles.conOK,
            value,handles.Addr{i},handles.slast{i},handles.RET,handles.minT{i},handles.maxT{i},
            handles.rsetBtn);
527     set(handles.setTilttxt(i),'string','');
528     % Updates the sequence number.
529     setappdata(0,'MyStruct',handles.slast);
530 catch
531     set(handles.tiltinfo,'string','Error: Tilt not set.');
532 end
533 guidata(hObject,handles);
534
535
536
537 function setTiltedit_Callback(hObject, eventdata, handles)
538 % hObject    handle to setTiltedit (see GCBO)
539 % eventdata  reserved - to be defined in a future version of MATLAB
540 % handles    structure with handles and user data (see GUIDATA)
541
542 % Hints: get(hObject,'String') returns contents of setTiltedit as text
543 %        str2double(get(hObject,'String')) returns contents of setTiltedit as a double
544
545
546 % --- Executes during object creation, after setting all properties.
547 function setTiltedit_CreateFcn(hObject, eventdata, handles)
548 % hObject    handle to setTiltedit (see GCBO)
549 % eventdata  reserved - to be defined in a future version of MATLAB
550 % handles    empty - handles not created until after all CreateFcns called
551
552 % Hint: edit controls usually have a white background on Windows.
553 %       See ISPC and COMPUTER.
554 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
555     set(hObject,'BackgroundColor','white');
556 end
557
558
559 % --- If Enable == 'on', executes on mouse press in 5 pixel border.
560 % --- Otherwise, executes on mouse press in 5 pixel border or over comList.
561 function comList_ButtonDownFcn(hObject, eventdata, handles)
562 % hObject    handle to comList (see GCBO)
563 % eventdata  reserved - to be defined in a future version of MATLAB
564 % handles    structure with handles and user data (see GUIDATA)
565
566
567
568 % --- Executes during object deletion, before destroying properties.
569 function ExitBtn_DeleteFcn(hObject, eventdata, handles)
570 % hObject    handle to ExitBtn (see GCBO)
571 % eventdata  reserved - to be defined in a future version of MATLAB
572 % handles    structure with handles and user data (see GUIDATA)
573
574
575 % --- Executes on button press in getCombtn.
576 function getCombtn_Callback(hObject, eventdata, handles)
577 % hObject    handle to getCombtn (see GCBO)
578 % eventdata  reserved - to be defined in a future version of MATLAB
579 % handles    structure with handles and user data (see GUIDATA)
580 handles.output = hObject;
581 if ~handles.conOK
582     set(handles.portTx, 'string', 'Searching...')
583     % Waiting animation of the pointer.
584     oldpointer = get(handles.Gui1, 'pointer');
585     set(handles.Gui1, 'pointer', 'watch')
586     drawnow;
587     % Gets the availble COM ports and loads them into the popupmenu.
588     ports = getCOMs;
589     set(handles.comList,'string',ports);
590     set(handles.Gui1, 'pointer', oldpointer)
591     set(handles.portTx, 'string', '')
592 end
593 % Update handles structure
594 guidata(hObject, handles);
595
596
597
598 function retSer_Callback(hObject, eventdata, handles)
599 % hObject    handle to retSer (see GCBO)
600 % eventdata  reserved - to be defined in a future version of MATLAB
601 % handles    structure with handles and user data (see GUIDATA)
602
603 % Hints: get(hObject,'String') returns contents of retSer as text
604 %        str2double(get(hObject,'String')) returns contents of retSer as a double
605
606
607 % --- Executes during object creation, after setting all properties.
608 function retSer_CreateFcn(hObject, eventdata, handles)
609 % hObject    handle to retSer (see GCBO)
```

```
610 % eventdata  reserved - to be defined in a future version of MATLAB
611 % handles    empty - handles not created until after all CreateFcns called
612
613 % Hint: edit controls usually have a white background on Windows.
614 %       See ISPC and COMPUTER.
615 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
616     set(hObject,'BackgroundColor','white');
617 end
618
619
620
621 function AnSer_Callback(hObject, eventdata, handles)
622 % hObject    handle to AnSer (see GCBO)
623 % eventdata  reserved - to be defined in a future version of MATLAB
624 % handles    structure with handles and user data (see GUIDATA)
625
626 % Hints: get(hObject,'String') returns contents of AnSer as text
627 %        str2double(get(hObject,'String')) returns contents of AnSer as a double
628
629
630 % --- Executes during object creation, after setting all properties.
631 function AnSer_CreateFcn(hObject, eventdata, handles)
632 % hObject    handle to AnSer (see GCBO)
633 % eventdata  reserved - to be defined in a future version of MATLAB
634 % handles    empty - handles not created until after all CreateFcns called
635
636 % Hint: edit controls usually have a white background on Windows.
637 %       See ISPC and COMPUTER.
638 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
639     set(hObject,'BackgroundColor','white');
640 end
641
642
643 % --- Executes on selection change in vendorlist.
644 function vendorlist_Callback(hObject, eventdata, handles)
645 % hObject    handle to vendorlist (see GCBO)
646 % eventdata  reserved - to be defined in a future version of MATLAB
647 % handles    structure with handles and user data (see GUIDATA)
648
649 % Hints: contents = cellstr(get(hObject,'String')) returns vendorlist contents as cell array
650 %        contents{get(hObject,'Value')} returns selected item from vendorlist
651
652
653 % --- Executes during object creation, after setting all properties.
654 function vendorlist_CreateFcn(hObject, eventdata, handles)
655 % hObject    handle to vendorlist (see GCBO)
656 % eventdata  reserved - to be defined in a future version of MATLAB
657 % handles    empty - handles not created until after all CreateFcns called
658
659 % Hint: popupmenu controls usually have a white background on Windows.
660 %       See ISPC and COMPUTER.
661 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
662     set(hObject,'BackgroundColor','white');
663 end
664
665 function tiltAlltxt_Callback(hObject, eventdata, handles)
666 % hObject    handle to tiltAlltxt (see GCBO)
667 % eventdata  reserved - to be defined in a future version of MATLAB
668 % handles    structure with handles and user data (see GUIDATA)
669
670 % Hints: get(hObject,'String') returns contents of tiltAlltxt as text
671 %        str2double(get(hObject,'String')) returns contents of tiltAlltxt as a double
672
673
674 % --- Executes during object creation, after setting all properties.
675 function tiltAlltxt_CreateFcn(hObject, eventdata, handles)
676 % hObject    handle to tiltAlltxt (see GCBO)
677 % eventdata  reserved - to be defined in a future version of MATLAB
678 % handles    empty - handles not created until after all CreateFcns called
679
680 % Hint: edit controls usually have a white background on Windows.
681 %       See ISPC and COMPUTER.
682 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
683     set(hObject,'BackgroundColor','white');
684 end
685
686
687 % --- Executes on button press in netBtn.
688 function netBtn_Callback(hObject, eventdata, handles)
689 % hObject    handle to netBtn (see GCBO)
690 % eventdata  reserved - to be defined in a future version of MATLAB
691 % handles    structure with handles and user data (see GUIDATA)
692 handles.output = hObject;
693 % Check if Connected
694  try
695     cali = 0;
696     okport = length(handles.Addr);
697     % Check if all devices are calibrated and have port names.
698     for i = 1:length(handles.Addr)
699         cali = cali + strcmp(get(handles.calitxt(i),'string'),'Calibrated');
700         okport = okport - isempty(get(handles.portNa(i),'string'));
701     end
702     % If all ports are calibrated.
703     if cali == length(handles.Addr)
704         % If all ports have names.
```

```
        if okport == length(handles.Addr)
            % check if networkOpt is running.
            hfigure2 = getappdata(0,'fig2');
            % if not open networkOpt.
            if hfigure2
                networkOpt;
                setappdata(0,'fig2',0);
            end
        else
            msgbox('All ports needs to have a port name assigned.','Error')
        end
    else
        msgbox('All ports needs to be calibrated.','Error')
    end
catch
    msgbox('Not connected.','Error')
end
% Update handles structure
guidata(hObject, handles);



function nrofPP_Callback(hObject, eventdata, handles)
% hObject    handle to nrofPP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nrofPP as text
%        str2double(get(hObject,'String')) returns contents of nrofPP as a double


% --- Executes during object creation, after setting all properties.
function nrofPP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nrofPP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in rsetBtn.
function rsetBtn_Callback(hObject, eventdata, handles)
% hObject    handle to rsetBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.output = hObject;
set(handles.Gui1, 'pointer', 'arrow');
% Enable interrupt of running fuctions.
set(gcbo,'userdata',0)
set(handles.tiltinfo,'string','')
% if connected.
if handles.conOK
    for a = 1:length(handles.Addr)
        % Reset command
        RSET(handles.Addr{a},handles.RET);
        doneSnrm = 1;
        % SNRM command.
        snrm = snrmCMD(handles.Addr{a});
        fwrite(handles.RET,hex2dec(snrm),'uint8');
        while doneSnrm
            snrmAnw = fread(handles.RET);
            if ~isnan(snrmAnw)
                checkSNRM = convertAnswer(snrmAnw);
                b = checkCrc(snrmAnw);
                if b
                    fwrite(handles.RET,hex2dec(snrm),'uint8');
                else
                    doneSnrm = 0;
                end
            end
        end
        % Resets sequence number.
        handles.slast(a) = {'11'};
        % Updates sequence number.
        setappdata(0,'MyStruct',handles.slast);
    end
end
guidata(hObject, handles);


% --- Executes when user attempts to close Gui1.
function Gui1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to Gui1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(hObject)
```

Listing B.2: networkOpt

```
function varargout = networkOpt(varargin)
% NETWORKOPT MATLAB code for networkOpt.fig
%      NETWORKOPT, by itself, creates a new NETWORKOPT or raises the existing
%      singleton*.
%
%      H = NETWORKOPT returns the handle to a new NETWORKOPT or the handle to
%      the existing singleton*.
%
%      NETWORKOPT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in NETWORKOPT.M with the given input arguments.
%
%      NETWORKOPT('Property','Value',...) creates a new NETWORKOPT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before networkOpt_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to networkOpt_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help networkOpt

% Last Modified by GUIDE v2.5 16-May-2017 12:46:32

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @networkOpt_OpeningFcn, ...
    'gui_OutputFcn',  @networkOpt_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before networkOpt is made visible.
function networkOpt_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to networkOpt (see VARARGIN)

% Choose default command line output for networkOpt
handles.output = hObject;
handles.prevtiltnr = 0;
% Pixel value for creating TiltX boxes.
handles.kval = 181;
% Resets reset button.
set(handles.resetbutton,'userdata',1)
% Checks if RETcontrol window is not closed.
h = findobj('Tag','Gui1');
if ~isempty(h)
    % Gets variables from RETcontrol.
    g1data = guidata(h);
    % Pixel value for creating PORT checkboxes
    l = 185;
    for i = 1:length(g1data.Addr)
        % If the new Object is outside of the panel.
        if l-29*i < 10
            l = l - 29;
            % Moves the Panels.
            a = get(handles.Gui2,'position');
            set(handles.Gui2,'Units','pixels','position',[a(1) a(2)-29 a(3) a(4)+29]);
            a = get(handles.portpanel,'position');
            set(handles.portpanel,'Units','pixels','position',[a(1) a(2) a(3) a(4)+29]);
            a = get(handles.tiltpanel,'position');
            set(handles.tiltpanel,'Units','pixels','position',[a(1) a(2) a(3) a(4)+29]);
            changepos(handles.nroftilt,29);
            changepos(handles.pathbtn,29);
            changepos(handles.pathTxt,29);
            changepos(handles.measureBtn,29);
            changepos(handles.selectbtn,29);
            changepos(handles.IntraBtn,29);
            changepos(handles.InterBtn,29);
            % Moves the Checkboxes and satus texts.
            for j = 1:length(g1data.Addr)
                changepos(handles.portnr(j),29);
                changepos(handles.portstatus(j),29);
            end
        end
        % Creates the Checkboxes and status texts for the ports.
        handles.portnr(i) = uicontrol('parent',handles.portpanel,'Style','radiobutton',...
            'String',get(g1data.portNa(i),'string'),'Position',[24 l-29*i 45 23],...
```

```
96              'Callback',@(hObject,eventdata)networkOpt('portBtn_Callback',hObject,eventdata,
                    guidata(hObject)),'Tag',sprintf('port%d',i));
97          handles.portstatus(i) = uicontrol('parent',handles.portpanel,'Style','text','
                HorizontalAlignment','left',...
98              'Position',[67 (1+4)-29*i 69 14]);
99      end
100     % Save variables from RETcontrol into a handles object.
101     handles.g1 = g1data;
102 end
103 % Update handles structure
104 guidata(hObject, handles);
105
106 % UIWAIT makes networkOpt wait for user response (see UIRESUME)
107 % uiwait(handles.Gui2);
108
109
110 % --- Outputs from this function are returned to the command line.
111 function varargout = networkOpt_OutputFcn(hObject, eventdata, handles)
112 % varargout  cell array for returning output args (see VARARGOUT);
113 % hObject    handle to figure
114 % eventdata  reserved - to be defined in a future version of MATLAB
115 % handles    structure with handles and user data (see GUIDATA)
116
117 % Get default command line output from handles structure
118 varargout{1} = handles.output;
119
120
121 % --- Executes on button press in measureBtn.
122 function measureBtn_Callback(hObject, eventdata, handles)
123 % hObject    handle to measureBtn (see GCBO)
124 % eventdata  reserved - to be defined in a future version of MATLAB
125 % handles    structure with handles and user data (see GUIDATA)
126 handles.output = hObject;
127 set(handles.resetbutton,'userdata',1)
128 set(handles.textbox,'string','');
129
130 % Check if filepath is specified.
131 ifpath = get(handles.pathTxt,'string');
132 if ~isnan(ifpath)
133     % Check if Intra or Inter checkboxes is checked.
134     if get(handles.IntraBtn, 'Value')
135         % if Intra.
136         h = findobj('Tag','Gui1');
137         if ~isempty(h)
138             % Updates variables form RETcontrol if RETcontrol exists.
139             handles.g1 = guidata(h);
140         end
141         % Clear all port status texts.
142         for i = 1:length(handles.g1.Addr)
143             set(handles.portstatus(i),'string','');
144         end
145         for i = 1:length(handles.g1.Addr)
146             checked = get(handles.portnr(i), 'Value');
147             if checked
148                 % If one checkbox is checked, read number of tilt values.
149                 nroftilt  = str2double(get(handles.nroftilt,'string'));
150                 if nroftilt > 0
151                     % check that all tilt angles has a value.
152                     okport = nroftilt;
153                     for o = 1:nroftilt
154                         set(handles.tiltstatus(o),'string','');
155                         okport = okport - isempty(get(handles.tilt(o),'string'));
156                         ch = str2double(get(handles.tilt(o),'string'));
157                         if (handles.g1.minT{i} <= ch)&&(ch <= handles.g1.maxT{i})
158                         else
159                             okport = okport - 1;
160                         end
161                     end
162                     % If all tilt boxes has values.
163                     if okport == nroftilt
164                         set(handles.portstatus(i),'string','In Progress..','ForegroundColor','
                                black');
165                         % Start set tilt.
166                         for j = 1:nroftilt
167                             % checks if reset button has been pressed.
168                             if get(handles.resetbutton,'userdata')
169                                 s = sprintf('Setting Tilt%d...',j);
170                                 set(handles.textbox,'string',s);
171                                 value = get(handles.tilt(j),'string');
172                                 % Update sequence number.
173                                 a = getappdata(0,'MyStruct');
174                                 handles.g1.slast = a;
175                                 set(handles.tiltstatus(j),'string','In Progress..','
                                        ForegroundColor','black');
176                                 % Set tilt.
177                                 handles.g1.slast{i} = setTiltGUI(handles.textbox,handles.g1.
                                        cTilt(i),handles.Gui2,handles.g1.conOK,value,handles.g1.Addr
                                        {i},handles.g1.slast{i},handles.g1.RET,handles.g1.minT{i},
                                        handles.g1.maxT{i},handles.resetbutton);
178                                 % Update sequence number.
179                                 setappdata(0,'MyStruct',handles.g1.slast);
180                                 if get(handles.resetbutton,'userdata')
181                                     % Creates the filename.
182                                     tiltS = strcat('0',num2str(str2double(value)*10));
183                                     if length(tiltS) > 3
```

```
184                                     tiltS = tiltS(2:end);
185                                 end
186                                 sn = handles.g1.SerNrAn;
187                                 p = get(handles.portnr(i),'string');
188                                 % CREATE THE FILE NAME.
189                                 file = strcat('AUTIntraSWR_',sn,'_',p,'_',tiltS,'T.dat');
190                                 % Wait 10 seconds.
191                                 set(handles.textbox,'string','Waiting for clean sweep (10s).
                                        ');
192                                 pause(10);
193                                 % Measures the S-parameters.
194                                 set(handles.textbox,'string','Measuring S-parameters.');
195                                 getSparam(file,handles.path);
196
197                                 set(handles.tiltstatus(j),'string','OK','ForegroundColor','
                                        green');
198                                 s = sprintf('Tilt%d done.',j);
199                                 set(handles.textbox,'string',s);
200                             end
201                         else
202                             set(handles.textbox,'string','Stopped.')
203                             guidata(hObject, handles);
204                             return;
205                         end
206                     end
207                     set(handles.portstatus(i),'string','OK','ForegroundColor','green');
208                     set(handles.textbox,'string','Intra measurement done.');
209                 else
210                     msgbox('All downtilt angles needs values and the values needs to be
                            within the downtilt range.');
211                 end
212             end
213         end
214     end
215     % If Inter.
216 elseif get(handles.InterBtn, 'Value')
217     % Creates an array of i.g R1+, R1-, Y1+, Y1- from R1, Y1
218     gPort = get(handles.portnr(1:end),'string');
219     gPort = reshape(gPort,1,[]);
220     Allports = cell(1,length(gPort)*2);
221
222     for i = 1:length(handles.portnr)
223         % Check if Reset button is pressed.
224         if get(handles.resetbutton,'userdata')
225             % Creates an array of i.g R1+, R1-, Y1+, Y1- from R1, Y1
226             Allports(i*2-1) = strcat(gPort(i),'+');
227             Allports(i*2) = strcat(gPort(i),'-');
228
229             othertilt = str2double(get(handles.porttilt(i),'string'));
230             if ~isnan(othertilt)
231                 % If a tilt value other than default selected.
232                 if (othertilt) ~= str2double(get(handles.g1.cTilt(i),'string'))
233                     % Check if value is within downtilt range.
234                     if (handles.g1.minT{i} <= othertilt)&&(othertilt <= handles.g1.maxT{i})
235                         s = sprintf('Setting port%d Tilt to custom value.',i);
236                         set(handles.textbox,'string',s);
237                         % Update sequence number.
238                         a = getappdata(0,'MyStruct');
239                         handles.g1.slast = a;
240                         % Set tilt.
241                         handles.g1.slast{i} = setTiltGUI(handles.textbox,handles.g1.cTilt(i)
                                ,handles.Gui2,handles.g1.conOK,num2str(othertilt),handles.g1.
                                Addr{i},handles.g1.slast{i},handles.g1.RET,handles.g1.minT{i},
                                handles.g1.maxT{i},handles.resetbutton);
242                         % Update sequence number.
243                         setappdata(0,'MyStruct',handles.g1.slast);
244                     else
245                         msgbox('Tilt values needs to be within downtilt range.')
246                         return;
247                     end
248                 end
249             else
250                 % If no tilt value specified, use the lowest.
251                 if (handles.g1.minT{i}) ~= str2double(get(handles.g1.cTilt(i),'string'))
252                     s = sprintf('Setting port%d Tilt to the lowest value.',i);
253                     set(handles.textbox,'string',s);
254                     % Update sequence number.
255                     a = getappdata(0,'MyStruct');
256                     handles.g1.slast = a;
257                     % Set tilt.
258                     handles.g1.slast{i} = setTiltGUI(handles.textbox,handles.g1.cTilt(i),
                            handles.Gui2,handles.g1.conOK,num2str(handles.g1.minT{i}),handles.g1
                            .Addr{i},handles.g1.slast{i},handles.g1.RET,handles.g1.minT{i},
                            handles.g1.maxT{i},handles.resetbutton);
259                     % Update sequence number.
260                     setappdata(0,'MyStruct',handles.g1.slast);
261                 end
262             end
263         end
264     end
265     % Check if reset button is pressed.
266     if get(handles.resetbutton,'userdata')
267         set(handles.textbox,'string','');
268         % Check if more than 1 port pair is connected.
269         if length(gPort) > 1
```

```
                k = 1;
                try
                    % Go through all of the possible combinations of R1+
                    % Y1- etc.
                    for i = 1:length(Allports)-2
                        for j = 1+2*k:length(Allports)
                            str = [strcat('Connect NWA port1 to AP',{' '},Allports(i),' and NWA
                                port2 to AP',{' '},Allports(j), '.') 'Then press OK to measure.'
                                ];
                            uiwait(msgbox(str,'Connect ports'));
                            set(handles.textbox,'string','Waiting for clean sweep (10s).');
                            % Waits 10 seconds.
                            pause(10);
                            % Create the filename.
                            val(1) = round(i/2);
                            val(2) = round(j/2);

                            for g = 1:2
                                s = num2str(str2double(get(handles.g1.cTilt(val(g)),'string'))
                                    *10);
                                tiltS = strcat('0',s);
                                if length(tiltS) > 3
                                    tiltS = tiltS(2:end);
                                end
                                values(g,:) = tiltS(1:end);
                            end
                            sn = handles.g1.SerNrAn;
                            p = char(Allports(i));
                            p2 = char(Allports(j));
                            % CREATE THE FILE NAME.
                            file = strcat('AUTInterSWR_',sn,'_',p,values(1,:),'T',p2,values(2,:)
                                ,'T.dat');
                            set(handles.textbox,'string','Measuring S-parameters.');
                            % Measures the S-parameters.
                            getSparam(file,handles.path);
                        end
                        % Next in the list.
                        if ~mod(i,2)
                            k = k + 1;
                        end
                    end
                    set(handles.textbox,'string','Inter measurement done.');
                catch
                    msgbox('Cant connect to the network analyzer.','Error');
                end
            else
                msgbox('Only one port pair connected','Error');
            end
        end
    else
        msgbox('Select one: Intra or Inter.','Error');
    end
else
    msgbox('Save path needed.','Error');
end
guidata(hObject, handles);

% --- Executes on selection change in nroftilt.
function nroftilt_Callback(hObject, eventdata, handles)
% hObject    handle to nroftilt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns nroftilt contents as cell array
%        contents{get(hObject,'Value')} returns selected item from nroftilt


% --- Executes during object creation, after setting all properties.
function nroftilt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nroftilt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pathbtn.
function pathbtn_Callback(hObject, eventdata, handles)
% hObject    handle to pathbtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.output = hObject;
% Select folder path.
path = uigetdir('','Select Path');
set(handles.pathTxt,'string',path);
handles.path = path;
guidata(hObject, handles);

% --- Executes on button press in Exitbtn.
function Exitbtn_Callback(hObject, eventdata, handles)
% hObject    handle to Exitbtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% Closes the window.
close;


function pathTxt_Callback(hObject, eventdata, handles)
% hObject    handle to pathTxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pathTxt as text
%        str2double(get(hObject,'String')) returns contents of pathTxt as a double


% --- Executes during object creation, after setting all properties.
function pathTxt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pathTxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in selectbtn.
function selectbtn_Callback(hObject, eventdata, handles)
% hObject    handle to selectbtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.output = hObject;
% Gets the ammount of tilt values.
tiltnr = str2double(get(handles.nroftilt,'string'));
try
    % Controls number is less or equal to 30.
    if tiltnr >= 31
        msgbox('Value needs to be 30 or less','Error');
    else
        % Checks if the new value is different from the old one.
        if (tiltnr ~= handles.prevtiltnr)
            k = handles.kval;
            % If less values.
            if tiltnr < handles.prevtiltnr
                % Delete the excessive values.
                for i = tiltnr + 1:handles.prevtiltnr
                    delete(handles.tiltnr(i));
                    delete(handles.tilt(i));
                    delete(handles.tiltstatus(i))
                    c = get(handles.tiltpanel,'position');
                    if c(4) > 215
                        k = k - 29;
                        % Moves the GUI and Panels and buttons.
                        a = get(handles.Gui2,'position');
                        set(handles.Gui2,'Units','pixels','position',[a(1) a(2)+29 a(3) a(4)
                            -29]);
                        a = get(handles.portpanel,'position');
                        set(handles.portpanel,'Units','pixels','position',[a(1) a(2) a(3) a(4)
                            -29]);
                        a = get(handles.tiltpanel,'position');
                        set(handles.tiltpanel,'Units','pixels','position',[a(1) a(2) a(3) a(4)
                            -29]);
                        changepos(handles.nroftilt,-29);
                        changepos(handles.pathbtn,-29);
                        changepos(handles.pathTxt,-29);
                        changepos(handles.measureBtn,-29);
                        changepos(handles.selectbtn,-29);
                        changepos(handles.IntraBtn,-29);
                        changepos(handles.InterBtn,-29);
                        % Moves the Tilt objects.
                        for j = 1:tiltnr
                            changepos(handles.tiltnr(j),-29);
                            changepos(handles.tilt(j),-29);
                            changepos(handles.tiltstatus(j),-29);
                        end
                        % Moves the Port objects.
                        for j = 1:length(handles.g1.Addr)
                            changepos(handles.portnr(j),-29);
                            changepos(handles.portstatus(j),-29);
                        end
                    end
                end
            else
                for i = handles.prevtiltnr+1:tiltnr
                    % Checks if the boxes fits into the panel
                    if k-29*i < 5
                        k = k + 29;
                        % Moves the GUI and Panels and buttons.
                        a = get(handles.Gui2,'position');
                        set(handles.Gui2,'Units','pixels','position',[a(1) a(2)-29 a(3) a(4)
                            +29]);
                        a = get(handles.portpanel,'position');
                        set(handles.portpanel,'Units','pixels','position',[a(1) a(2) a(3) a(4)
                            +29]);
```

```
451                         a = get(handles.tiltpanel,'position');
452                         set(handles.tiltpanel,'Units','pixels','position',[a(1) a(2) a(3) a(4)
                                +29]);
453                         changepos(handles.nroftilt,29);
454                         changepos(handles.pathbtn,29);
455                         changepos(handles.pathTxt,29);
456                         changepos(handles.measureBtn,29);
457                         changepos(handles.selectbtn,29);
458                         changepos(handles.IntraBtn,29);
459                         changepos(handles.InterBtn,29);
460                         % Moves the Tilt objects.
461                         for j = 1:i-1
462                             changepos(handles.tiltnr(j),29);
463                             changepos(handles.tilt(j),29);
464                             changepos(handles.tiltstatus(j),29);
465                         end
466                         % Moves the Port objects.
467                         for j = 1:length(handles.g1.Addr)
468                             changepos(handles.portnr(j),29);
469                             changepos(handles.portstatus(j),29);
470                         end
471                     end
472                     % Creates the Tilt texts and editboxes.
473                     handles.tiltnr(i) = uicontrol('parent',handles.tiltpanel,'Style','text','
                            HorizontalAlignment','right','FontSize',9.0,...
474                         'String',sprintf('Tilt%d',i),'Position',[22 (k+4)-29*i 52 14]);
475                     handles.tilt(i) = uicontrol('parent',handles.tiltpanel,'Style','edit',...
476                         'String','','HorizontalAlignment','left','Position',[79 k-29*i 51 22]);
477                     handles.tiltstatus(i) = uicontrol('parent',handles.tiltpanel,'Style','text',
                            'HorizontalAlignment','left',...
478                         'Position',[134 (k+4)-29*i 69 14]);
479                 end
480             end
481             handles.kval = k;
482         end
483         % Updates the previous tilt number.
484         handles.prevtiltnr = tiltnr;
485     end
486 catch
487     msgbox('Wrong input. Input shall be a number.','Error');
488 end
489 guidata(hObject, handles);
490
491
492 % --- Executes on button press in IntraBtn.
493 function IntraBtn_Callback(hObject, eventdata, handles)
494 % hObject    handle to IntraBtn (see GCBO)
495 % eventdata  reserved - to be defined in a future version of MATLAB
496 % handles    structure with handles and user data (see GUIDATA)
497
498 % Hint: get(hObject,'Value') returns toggle state of IntraBtn
499 handles.output = hObject;
500 set(handles.InterBtn,'value',0);
501 set(handles.nroftilt,'Enable','on')
502 set(handles.selectbtn,'Enable','on')
503 % reenables the objects.
504 a = str2double(get(handles.nroftilt,'string'));
505 if ~isnan(a)
506     for i = 1:a
507         set(handles.tilt(i),'Enable','on');
508         set(handles.tiltstatus(i),'string','','ForegroundColor','black');
509     end
510 end
511 for i = 1:length(handles.g1.Addr)
512     try
513         delete(handles.porttilt(i));
514     catch
515     end
516     set(handles.portnr(i),'Enable','on');
517     set(handles.portstatus(i),'string','','ForegroundColor','black');
518 end
519 guidata(hObject, handles);
520
521 % --- Executes on button press in InterBtn.
522 function InterBtn_Callback(hObject, eventdata, handles)
523 % hObject    handle to InterBtn (see GCBO)
524 % eventdata  reserved - to be defined in a future version of MATLAB
525 % handles    structure with handles and user data (see GUIDATA)
526
527 % Hint: get(hObject,'Value') returns toggle state of InterBtn
528 handles.output = hObject;
529 set(handles.IntraBtn,'value',0);
530 set(handles.nroftilt,'Enable','off')
531 set(handles.selectbtn,'Enable','off')
532 a = str2double(get(handles.nroftilt,'string'));
533 % disables objects
534 if ~isnan(a)
535     for i = 1:a
536         set(handles.tilt(i),'Enable','off');
537         set(handles.tiltstatus(i),'string','','ForegroundColor','black');
538     end
539 end
540 l = 185;
541 nroftilt  = str2double(get(handles.nroftilt,'string'));
542 if nroftilt > 6
```

```
    inc = (nroftilt-6)*29;
else
    inc = 0;
end
% Creates the port tilt boxes for Inter measure.
for i = 1:length(handles.g1.Addr)
    handles.porttilt(i) = uicontrol('parent',handles.portpanel,'Style','edit','Tag',sprintf('
        ptilt%d',i),...
        'String','','HorizontalAlignment','left','Position',[85 1+inc-29*i 51 22]);
    set(handles.portnr(i),'Enable','off');
    set(handles.portstatus(i),'string','','ForegroundColor','black');
end
guidata(hObject, handles);

% --- Executes on button press in portBtn.
function portBtn_Callback(hObject, eventdata, handles)
% hObject    handle to IntraBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of IntraBtn
handles.output = hObject;
% If one port checkbox is checked, disable the other ones.
k = str2double(hObject.Tag(end));
h = findobj('Tag','Gui1');
if ~isempty(h)
    g1data = guidata(h);
    if get(hObject,'value')
        for i = 1:length(g1data.Addr)
            if i ~= k
                set(handles.portnr(i),'value',0);
            end
        end
    end
end
guidata(hObject, handles);


% --- Executes on button press in resetbutton.
function resetbutton_Callback(hObject, eventdata, handles)
% hObject    handle to resetbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.output = hObject;
set(handles.Gui2, 'pointer', 'arrow');
% Enable to interrupt running functions.
set(gcbo,'userdata',0);
set(handles.textbox,'string','')
% Send RESET and SNRM to each connected device.
for a = 1:length(handles.g1.Addr)
    % Reset command
    RSET(handles.g1.Addr{a},handles.g1.RET);
    doneSnrm = 1;
    % SNRM command.
    snrm = snrmCMD(handles.g1.Addr{a});
    fwrite(handles.g1.RET,hex2dec(snrm),'uint8');
    while doneSnrm
        snrmAnw = fread(handles.g1.RET);
        if ~isnan(snrmAnw)
            checkSNRM = convertAnswer(snrmAnw);
            b = checkCrc(snrmAnw);
            if b
                fwrite(handles.g1.RET,hex2dec(snrm),'uint8');
            else
                doneSnrm = 0;
            end
        end
    end
    % Resets sequence number.
    handles.g1.slast(a) = {'11'};
    % Updates sequence number.
    setappdata(0,'MyStruct',handles.g1.slast);
end
guidata(hObject, handles);


% --- Executes when user attempts to close Gui2.
function Gui2_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to Gui2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
% Indicates networkOpt is closed.
setappdata(0,'fig2',1);
delete(hObject);
```

Listing B.3: Add space between octets

```
function [b] = addSpace(input)
%ADDSPACE Addspaces between octets (hexadecimal).
%   Input:  String of hexadecimal numbers.
%   Output: String of hexadecimal numbers devided into pairs of two.
```

```
5
6  a = mat2cell(input, 1, 2*ones(1,numel(input)/2));
7  b = strjoin(a, ' ');
8  end
```

Listing B.4: Byte Inversion

```
1  function [invOut] = byteInversion(input)
2  %ByteInversion: Invert the bytes of the input.
3  %   Input:  Hexadecimal char string with no spaces.
4  %           Ex. FF BF 81 F0 04 is 'FFBF81F004'
5  %   Output: The bytewise inversion of the input.
6
7  xor = ones(length(input),4);                % Preallocation
8  invOut = zeros(1,length(input));
9  octets = zeros(1,length(input));
10
11 bin = dec2bin(hex2dec(input));              % Creates a binary vector from
12 bin = bin-'0';                              % the hexadecimal string.
13
14 while length(bin) < (length(input)*4)
15     bin = [0 bin];                          % Add zeros in the beginning if
16 end                                         % nessesary, to get a standard length.
17
18 for i = 1:length(input)
19     octets(i,:) = bin(i*4-3:i*4);           % Sorts the binary vectors,
20 end                                         % one symbol on each row.
21
22 inv = bitxor(octets,xor);                   % Inverts the bytes.
23
24 for m = 1:length(input)                     % Converts back to hexadecimal.
25     invOut(m) = dec2hex(bin2dec(num2str(inv(m,:))));
26 end
27 invOut = reshape(invOut,1,[]);
28 invOut = char(invOut);
29 end
```

Listing B.5: Bytewise Reflection

```
1  function [byteref] = bytewiseReflection(input)
2  %BytewiseReflection Bytewise Reflection:
3  %   Input:  Hexadecimal char string with no spaces.
4  %           Ex. FF BF 81 F0 04 is 'FFBF81F004'
5  %   Output: The bytewise reflection of the input.
6
7  b = zeros(length(input),4);                  % Preallocation.
8  bin = zeros(1,length(input)*4);
9  refhex = zeros(1,length(input));
10
11 for n = 1:length(input)
12     g = dec2bin(hex2dec(input(n)));          % Creates a binary vector from
13     while length(g)<(4)                      % the hexadecimal string.
14         g = strcat('0',g);
15     end
16     for o=1:4
17         bin(o+(n-1)*4) = g(o)-'0';
18     end
19 end
20 bin = logical(bin);
21
22 while length(bin)<(length(input)*4)
23     bin = [0 bin];                           % Add zeros in the beginning
24 end                                         % if nessesary.
25 for i = 1:length(input)
26     octets(i,:) = bin(i*4-3:i*4);            % Sorts the binary vectors,
27 end                                         % one symbol on each row.
28
29 for j = 1:length(input)
30     k(j,:) = fliplr(octets(j,:));            % Reflects each symbol.
31 end
32 for n = 1:(length(input)/2)
33     b(n*2-1,:) = k(n*2,:);                   % Swich the place of the 2 bytes.
34     b(n*2,:) = k(n*2-1,:);
35 end
36 for l = 1:length(input)                     % Converts back to hexadecimal.
37     refhex(l) = dec2hex(bin2dec(num2str(b(l,:))));
38 end
39 refhex = reshape(refhex,1,[]);
40 byteref = char(refhex);
41
42 end
```

Listing B.6: Calculate Control Frame

```
1  function [ctrlFrame] = calculateControlF(prevCtrlf,frameType)
2  %calculateControlF Calculates the control field of the HDLC frame
3  %   Input:  prevCtrlf: recived control frame in hexadecimal.
4  %           frameType: if it should be a S or I frame.
```

```
%   Output: The new control frame.

% Convets the control frame into a binary string.
prevCtrlf = dec2bin(hex2dec(prevCtrlf),8);
% Extracts the next expected frame number in decimal.
nr = bin2dec(prevCtrlf(1:3));
% Extracts the sequence number in decimal.
ns = bin2dec(prevCtrlf(5:7));
% Updates the new sequence number and next expected frame number.
if(nr > 0)
    ns = nr;
else
    ns = 0;
    nr = 0;
end

ns = dec2bin(ns,3);
nr = dec2bin(nr,3);
% Control frame looks different depending if it is an I-frame or S-frame.
switch frameType
    case 'I'
        ctrlFrame=[nr '1' ns '0'];
    case 'S'
        ns='000';
        ctrlFrame=[nr '1' ns '1'];
    otherwise
        error('Wrong frame type');
end
ctrlFrame = dec2hex(bin2dec(ctrlFrame));
end
```

Listing B.7: Calculate CRC

```
function [finalCrc] = calculateCRC(string)
%calculateCRC Calculates the cyclic redundancy check of type CRC-CCITT(0xFFFF).
%   Input:  String of the HDLC frame, in hexadecimal, without the flags.
%   Output: The 16-bit CRC code.

reflout = bytewiseReflection(string);
b = addSpace(reflout);
% Spits the octets into seperate elements.
crcIn = strsplit(b);
c = zeros(1,length(crcIn));
% Converts the octets into decimal number.
for i = 1:length(crcIn)
    c(i) = hex2dec(crcIn(i));
end
crcOut = crc_ccitt(uint16(c),length(c),'FFFF');
% Invert - string without spaces.
invOut = byteInversion(crcOut);
% Reflect - String without spaces.
finalCrc = bytewiseReflection(invOut);

end
```

Listing B.8: Create Calibrate Command

```
function [cali] = calibrateCMD(Addr,sqNr)
%calibrateCMD Constructs the Calibrate command.
%   Input:  Addr: the HDLC address of the RET.
%           sqNr: Control frame of the latest recieved frame.
%   Output: The whole HDLC frame with the calibrate command.

Flag = '7E';                                % HDLC flag.
ctrl = calculateControlF(sqNr,'I');
cmd = '31';                                 % Calibrate code.
cmdData = '0000';                           % No data is send.
tocrc = strcat(Addr,ctrl,cmd,cmdData);
crc = calculateCRC(tocrc);                  % Calculate CRC.
cali = strcat(Flag,tocrc,crc,Flag);         % Assemble the string.
cali = strsplit(addSpace(cali));            % Splits the string in pairs
                                            % of two.
end
```

Listing B.9: Calibrate for GUI

```
function [slast] = calibrateGUI(txt,fig,conOK,Addr,slast,RET,stop)
%CALIBRATEGUI Calls the Calibrate command with some error handling.
%       Input:  txt:    Text element for calibation.
%               fig:    Element for the pointer.
%               conOK:  If connected.
%               Addr:   HDLC Address of the RET.
%               slast:  Sequence Number of the RET.
%               RET:    Serial element.
%       Output: slast: New sequence number of the RET.

% Checks if connected.
if conOK
```

```
    set(txt,'string','In progress...','ForegroundColor','black');
    oldpointer = get(fig, 'pointer');
    set(fig, 'pointer', 'watch')
    drawnow;
    % Tries to call the Calibrate command.
    try
        [ok,slast] = exeCalibrate(Addr,slast,RET,stop);
        if ok
            set(txt,'string','Calibrated','ForegroundColor','green');
        else
            set(txt,'string','Error: Not calibrated.','ForegroundColor','red');
        end
    catch
        set(txt,'string','Error','ForegroundColor','red');
    end
    set(fig, 'pointer', oldpointer)
else
    set(txt,'string','Error: Not connected.','ForegroundColor','red');
end

end
```

Listing B.10: Delay from callback

```
function callbackDelay( obj, event, Addr, slast, RET )
%CALLBACKDELAY Adds a small delay between commands.
%   Detailed explanation goes here
for i = 1:2
    cmd = receiverReadyCMD(Addr,slast);
    fwrite(RET,hex2dec(cmd),'uint8');
end
end
```

Listing B.11: XID:Device scan and Address Assignment

```
function [Addr,SN,VnCd,DeTy] = callXID(RET,Vn,stop)
%CALLXID Executes the Exchange Identification command.
%   Input:  RET:    Serial port object.
%           Vn:     Vendor code.
%   Output: Addr:   Address of the secondary device.
%           SN:     Serial Number.
%           VnCd:   Vendor code.
%           DeTy:   Device type.

% List of HDLC addresses.
address = {'01' '02' '03' '04' '05' '06' '07' '08' '09' '10'};
% List of the last two bytes in the Unique ID of the RET.
n = {'0001' '0000' '0000' '0008' '0000' '0200' '0000' '0800'};
% List of the Bit Mask.
m = {'0001' '0002' '0004' '0008' '0100' '0200' '0400' '0800'};
% Preallocation.
Addr = cell(1,1);
SN = cell(1,1);
VnCd = cell(1,1);
DeTy = cell(1,1);
done = 1;
done2 = 1;
j = 1;
count = 1;

% Device scan ALL, Mask length = 0 and Unique ID length = 0.
cmd1 = XID('01','','03','');
while(done)&&get(stop,'userdata')
    fwrite(RET,hex2dec(cmd1),'uint8');
    anw = fread(RET);
    % Response OK or Corrupted.
    if ~isnan(anw)
        check1 = convertAnswer(anw);
        a = checkCrc(anw);
        % If response OK.
        if ~a
            if (check1(7:8) == '81')
                [Addr{count},SN{count},VnCd{count},DeTy{count}]=getXID(anw);
                adone = 1;
                % Address assignment Command.
                cmdA = XID('01',SN{count},'02',address{count});
                while(adone)
                    fwrite(RET,hex2dec(cmdA),'uint8');
                    anwA = fread(RET);
                    if ~isnan(anwA)
                        checkA = convertAnswer(anw);
                        if (checkA(7:8) == '81')
                            % Store the info.
                            [y,u,o,p] = getXID(anwA);
                            Addr{count} = y;
                            SN{count} = u;
                            DeTy{count} = p;
                            SN{count} = hex2char(SN{count});
                            VnCd{count} = hex2char(VnCd{count});
                            switch DeTy{count}
                                case '01'
```

```
                                    DeTy{count} = 'Single-Antenna RET Device';
                                case '11'
                                    DeTy{count} = 'Multi-Antenna RET Device';
                                case '02'
                                    DeTy{count} = 'Tower mounted amplifier (TMA)';
                            end
                            adone = 0;
                            done2 = 0;
                            done = 0;
                            % Point to the next column of the list.
                            count = count + 1;
                        end
                    else
                        break;
                    end
                end
            end
        end
        % If response Corrupt. Try to XID with Vendor Code and Bit Mask =
        % FFFF.
        while done2
            % Check for end of vendor list.
                cmd2 = XID('01',char2hex(Vn),'03','FFFF');
                fwrite(RET,hex2dec(cmd2),'uint8');
                anw = fread(RET);
                % Response OK or Corrupted.
                if ~isnan(anw)
                    check2 = convertAnswer(anw);
                    a = checkCrc(anw);
                    % If response OK.
                    if ~a
                        if (check2(7:8) == '81')
                            [Addr{count},SN{count},VnCd{count},DeTy{count}]=getXID(anw);
                            adone = 1;
                            % Address assignment.
                            cmdA=XID('01',SN{count},'02',address{count});
                            while(adone)
                                fwrite(RET,hex2dec(cmdA),'uint8');
                                anwA = fread(RET);
                                if ~isnan(anwA)
                                    checkA = convertAnswer(anw);
                                    if (checkA(7:8) == '81')
                                        [y,u,o,p]=getXID(anwA);
                                        Addr{count} = y;
                                        SN{count} = u;
                                        DeTy{count} = p;
                                        SN{count} = hex2char(SN{count});
                                        VnCd{count} = hex2char(VnCd{count});
                                        switch DeTy{count}
                                            case '01'
                                                DeTy{count} = 'Single-Antenna RET Device';
                                            case '11'
                                                DeTy{count} = 'Multi-Antenna RET Device';
                                            case '02'
                                                DeTy{count} = 'Tower mounted amplifier (TMA)';
                                        end
                                        adone = 0;
                                        done2 = 0;
                                        done = 0;
                                        count = count + 1;
                                    end
                                else
                                    break;
                                end
                            end
                        end
                    % If response Corrupted. Try XID with Vendor code and
                    % the last two bytes of the serial number.
                    else
                        for l = 1:8
                            for v = 1:8
                                mask3 = strcat(char2hex(Vn),'',n{v});
                                mask3_2 = strcat('FFFF',m{l});
                                cmd3 = XID('01',mask3,'03',mask3_2);
                                fwrite(RET,hex2dec(cmd3),'uint8');
                                anw = fread(RET);
                                if ~isnan(anw)
                                    check3 = convertAnswer(anw);
                                    a = checkCrc(anw);
                                    % If response OK.
                                    if ~a
                                        if (check3(7:8) == '81')
                                            [Addr{count},SN{count},VnCd{count},DeTy{count}]=
                                                getXID(anw);
                                            adone = 1;
                                            % Address assignment
                                            cmdA=XID('01',SN{count},'02',address{count});
                                            while(adone)
                                                fwrite(RET,hex2dec(cmdA),'uint8');
                                                anwA = fread(RET);
                                                if ~isnan(anwA)
                                                    checkA = convertAnswer(anw);
                                                    if (checkA(7:8) == '81')
                                                        [y,u,o,p]=getXID(anwA);
                                                        Addr{count} = y;
```

```
                                                    SN{count} = u;
                                                    DeTy{count} = p;
                                                    SN{count} = hex2char(SN{count});
                                                    VnCd{count} = hex2char(VnCd{count});
                                                    switch DeTy{count}
                                                        case '01'
                                                            DeTy{count} = 'Single-Antenna
                                                                RET Device';
                                                        case '11'
                                                            DeTy{count} = 'Multi-Antenna RET
                                                                Device';
                                                        case '02'
                                                            DeTy{count} = 'Tower mounted
                                                                amplifier (TMA)';
                                                    end
                                                    adone = 0;
                                                    count = count + 1;
                                                end
                                            else
                                                break;
                                            end
                                        end
                                    end
                                end
                            end
                            drawnow;
                        end
                    end
                    done2 = 0;
                    done = 0;
                end
            else
            end
        end
    end
    drawnow;
end
% If RETs are found, execute SNRM command.
if ~isempty(Addr)
    for a = 1:length(Addr)
        doneSnrm = 1;
        snrm = snrmCMD(Addr{a});
        fwrite(RET,hex2dec(snrm),'uint8');
        while doneSnrm && get(stop,'userdata')
            snrmAnw = fread(RET);
            if ~isnan(snrmAnw)
                checkSNRM = convertAnswer(snrmAnw);
                b = checkCrc(snrmAnw);
                if b
                    fwrite(RET,hex2dec(snrm),'uint8');
                else
                    doneSnrm = 0;
                end
            end
            drawnow;
        end
    end
end
end
```

Listing B.12: Change position of GUI object

```
function [] = changepos(obj,len)
%CHANGEPOS Change vertical position of elemnts in the GUI.
%   Input:  obj:    Object that is going to move
%           len:    Length to move in pixels.
a = get(obj,'position');
set(obj,'Units','pixels','position',[a(1) a(2)+len a(3) a(4)]);

end
```

Listing B.13: Char to hexadecimal conversion

```
function [ hx ] = char2hex( c )
%CHAR2HEX Converts a string into hexadecimal.
    hx = sprintf('%x', c);
end
```

Listing B.14: Check CRC code

```
function [error] = checkCrc(in)
%CHECKCRC Controls the recived frames CRC code.
%   Input:  The recived frame.
in=convertAnswer(in);
k = strfind(in,'7E');
if length(k) >= 2
    newin = in(k(1):k(2)+1);    % Control the flags.
    newin = flagrep(newin);
```

```
    echeck = newin(3:length(newin)-6);  % Extracts the Frame.
    a = mod(length(echeck),2);
    if a
        error = 1;
    else
        b = length(echeck);
        if b <= 0
            error = 1;
        else
            crc = calculateCRC(echeck);
            if crc == newin(length(newin)-5:length(newin)-2)
                error = 0;
            else
                error = 1;
            end
        end

    end

else
    error = 1;
end
end
```

Listing B.15: Check return code

```
function [] = checkReturnCode(input)
%CHECKRETURNCODE Function that checks the return code from the secondary
%device.
input = convertAnswer(input);
returnCode = strcat(input(13),input(14));

switch returnCode
    case '00'
        % OK. Normal response.
    case '02'
        error('Motor Jam')                      % Motor cannot move.
    case '03'
        error('ActuatorJam')                    % Actuator jam has been detected. No movement of
            the actuator, but movement of the motor was detected.
    case '05'
        error('Busy')                           % The device is busy and cannot execute the
            procedure until an ongoing activity is completed.
    case '06'
        error('ChecksumError')                  % Checksum incorrect for otherwise valid data.
    case '0B'
        error('FAIL')                           % Abnormal response. Indicates that a procedure
            has not been executed successfully.
    case '0E'
        error('NotCalibrated')                  % The device has not completed a calibration
            operation, or calibration has been lost.
    case '0F'
        error('NotConfigured')                  % Actuator configuration data is missing.
    case '11'
        error('HardwareError')                  % Any hardware error which cannot be classified.
             May not be reported as an alarm until the fault is likely to be persistent.
    case '13'
        error('OutOfRange')                     % A parameter given by an operator (e.g. tilt
            value or memory offset) is out of range.
    case '19'
        error('UnknownProcedure')               % Received procedure code is not defined.
    case '1D'
        error('ReadOnly')                       % Invalid device data parameter usage.
    case '1E'
        error('UnknownParameter')               % Specified parameter is not supported for the
            used procedure.
    case '21'
        error('WorkingSoftwareMissing')         % The unit is inDownloadMode state. Returned
            upon unsupported procedure when in DownloadMode state.
    case '22'
        error('InvalidFileContent')             % The data being downloaded is detected to be of
             wrong format or size.
    case '24'
        error('FormatError')                    % Procedure message is inconsistent or if an
            addressed field or antenna is invalid or the data parameter field length is
            inconsistent with the corresponding field length parameter.
    case '25'
        error('UnsupportedProcedure')           % The procedure is optional and not supported or
             the procedure does not apply to this device type.
    case '26'
        error('InvalidProcedureSequence')       % Procedure sequence as described in annex C is
            expected but not experienced by the secondary device.
    case '27'
        error('ActuatorInterference')           % An actuator movement outside the control of
            the RET unit has been detected. Probable cause is manual interference.
    case '1A'
        error('MinorTMAFault')                  % A fault in the TMA subunit is detected which
            reduces the gain performance but maintains its function.
    case '1B'
        error('MajorTMAFault')                  % A fault in the TMA subunit is detected. The
            fault prevents the function of the TMA subunit.
    case '1C'
        error('UnsupportedValue')               % The requested value is not supported.
```

```
    case '1F'
        error('BypassMode')                     % The TMA subunit is in bypass mode and cannot
            report a correct gain value.
    otherwise
        error('InvaildResponse')
end

end
```

Listing B.16: Close connection

```
function [] = closeConnection(RET)
%COLSECONNECTION Closes the connection to the Serial element and delete it.
% Input: Serial element.
fclose(RET);
delete(RET);
clear RET;
end
```

Listing B.17: Command to binary string

```
function [binstr] = cmd2binstr(string)
%UNTITLED Converts the hexadecimal command into a binary string.
input = regexprep(string,'[^\w'']','');
bin=zeros(1,length(input)*4);
for n=1:length(input)
    g=dec2bin(hex2dec(input(n)));
    while length(g)<(4)
        g=strcat('0',g);                        %Add zeros in the beginning
    end                                         %if nessesary.
    for o=1:4
        bin(o+(n-1)*4)=g(o)-'0';
    end
end
binstr=bin;
end
```

Listing B.18: Compare tilt values

```
function [value,ok] = compareTilt(desired,current)
%COMPARETILT Compare two tilt values.
%   Input:  The desired value and the current tilt value.
%   Output: ok: if desired value and current value are equal. 1 = yes, 0 =
%               no.
%           value: returns the current tilt value.

d = hex2dec(strcat(desired(1),desired(2)))*0.1;
if d == current
    ok = 1;
else
    ok = 0;
end
    value = current;
end
```

Listing B.19: Convert Answer

```
function [out] = convertAnswer(anw)
%convertAnswer Converts the answer from the RET to the right form.
%   Input:  RET answer in form of decimal numbers in one column.
%   Output: RET anwser in form of hexadecimal numbers in one row.
anw = dec2hex(anw);
out = reshape(anw.',1,[]);

end
```

Listing B.20: CRC CCITT (0xFFFF)

```
function [crc] = crc_ccitt(input_str,num_bytes,start_value)
%CRC_CCITT Does the computation of the CRC code.
%   Input:  input_str:   Hexadecimal string.
%           num_bytes:   The length of the string.
%           start_value: According to CRC-CCITT(0xFFFF) this value is
%           0xFFFF.
%   Output: The CRC code.

crc_tabccitt_init = 0;
crc_tabccitt = uint16(zeros(1,256));
crc = uint16(hex2dec(start_value));

if ~crc_tabccitt_init
    for i = 1:256
        crc2 = 0;
```

```
        c = bitshift(i-1,8);
        for j = 8:-1:1
            if bitand(bitxor(crc2,c),hex2dec('8000'))
                crc2 = bitxor(bitshift(crc2, 1),hex2dec('1021'));
            else
                crc2 = bitshift(crc2,1);
            end
            c = bitshift(c,1);
        end
        crc_tabccitt(i) = bitand(crc2,hex2dec('FFFF'));
    end
    crc_tabccitt_init = 1;
end

for a=1:num_bytes
    tmp = bitand(hex2dec('00FF'),bitxor(bitshift(crc,-8),input_str(a)),'uint16');
    crc = bitxor(bitshift(crc,8),crc_tabccitt(tmp+1));
end
crc = uint16((bitand(bitxor(crc,hex2dec('0000')),hex2dec('FFFF'))));
crc = dec2hex(crc);
if length(crc)<4
    for i = length(crc):3
        crc = strcat('0',crc);
    end
end
end
```

Listing B.21: Delay between commands

```
function delayBetweenCMD(Addr,slast,RET)
%DELAYBETWEENCMD Adds a small delay between commands.
%   Input:  Addr:   HDLC address.
%           slast:  Sequence number.
%           RET:    Serial object.
for i = 1: 2
    cmd = receiverReadyCMD(Addr,slast);
    fwrite(RET,hex2dec(cmd),'uint8');
    f = fread(RET,12);
end
end
```

Listing B.22: Disconnect

```
function [] = DISC(Addr,RET)
%DISC   Sends the DISC command to the secondary station indicating to
%       disconnect the connection.
%   Input:  Addr:   HDLC address.
%           RET:    Serial object.

donedisc = 1;
% Creates the command.
disc = discCMD(Addr);
% Writes the command to the serial object.
fwrite(RET,hex2dec(disc),'uint8');
while donedisc
    discAnw = fread(RET);
    if ~isnan(discAnw)
        % Check if answer is corrupt.
        b = checkCrc(discAnw);
        if b
            fwrite(RET,hex2dec(disc),'uint8');
        else
            donedisc = 0;
        end
    end
end
end
```

Listing B.23: Create Disconnect command

```
function [DISC_CMD] = discCMD(Addr)
%discCMD    Disconnect. This command is used to disconnect the
%           secondary station.
%   Input:  Addr: the HDLC address of the RET.
%   Output: The whole HDLC frame with the DISC command.

DISC = '53';                                % DISC code.
crc = calculateCRC(strcat(Addr,DISC));
DISC_CMD = strcat('7E',Addr,DISC,crc,'7E');
DISC_CMD = strsplit(addSpace(DISC_CMD));
end
```

Listing B.24: Check for error

```
function errorCheck( input,SERIAL,len)
%ERRORCHECK Cecks for error, if error, reread.
```

```
%   input:  The anwser from the RET.
%   SERIAL: The serial element created for the COM port.
%   len:    The length fread is going to read. 0 if unknown.

% Checks for error in the frame.
e = checkCrc(input);
if len == 0
    while e
        input = fread(SERIAL);
        e = checkCrc(input);
    end
else
    while e
        input = fread(SERIAL,len);
        e = checkCrc(input);
    end
end
end
```

Listing B.25: exeCalibrate

```
function [ok,slast] = exeCalibrate(Addr,slast,RET,stop)
%EXECALIBRATE Executes the calibration of the RET
%   Input:  Addr:   Address of the secondary device.
%           slast:  Sequence number.
%           RET:    Serial port object.
%   Output: ok:         1 indicates that the calibration was successful.
%           slast:  The next sequence number.

ok = 0;
 % Creates the calibrate command.
cmd1 = calibrateCMD(Addr,slast);
% Writes the calibrate command to the serial port.
fwrite(RET,hex2dec(cmd1),'uint8');
t = tic;
% Continously reads the serial port untill the right response is received
% or untill t = 4 min or if user presses the reset button.
while (toc(t) < 240) && get(stop,'userdata')
    returncode=fread(RET,24);
    if ~isnan(returncode)
        check = convertAnswer(returncode);
        % Checks if the response is the calibrate resposne.
        if (check(7:8) == '31')
            if ~checkCrc(returncode)
                checkReturnCode(returncode);
                % Updates the sequence number.
                slast = Slast(returncode);
                ok = 1;
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);
                return;
            else
                error('Not calibrated');
            end
        end
    else
        % Polling with Receiver Ready commands.
        cmd = receiverReadyCMD(Addr,slast);
        fwrite(RET,hex2dec(cmd),'uint8');
    end
    drawnow;
end
end
```

Listing B.26: Flag representation

```
function [newStr] = flagrep(string)
%FLAGREP    If Anwser contains 7D5E convert it into 7E and if anwser
%           contains 7D5D convert into 7D.
    newStr = strrep(string,'7D5E','7E');
    newStr = strrep(newStr,'7D5D','7D');
end
```

Listing B.27: getAserial

```
function [serNr,slast] = getAserial(Addr,slast,RET,stop)
%GETASERIAL  Function that returns the Minimum and Maximum supported
%               electrical tilt in degrees.
%   Input:  Addr:   Address of the secondary device.
%           slast:  Sequence number.
%           RET:    Serial port object.
%   Output: min:    Minimum electrical tilt value in degrees.
%           max:    Maximum electrical tilt value in degrees.
%           slast:  The next sequence number.
done = 1;
k = 1;
% Creates the GetDeviceData command with get Antenna Serial Number.
cmd = getBandwidthCMD(Addr,slast,'02');
```

```
% Writes the GetDeviceData command to the serial port.
fwrite(RET,hex2dec(cmd),'uint8');

% Continously reads the serial port untill the right response is received.
% or user presses the reset button.
while (done) && get(stop,'userdata')
    getser=fread(RET,54);
    if ~isnan(getser)
        check = convertAnswer(getser);
        % Checks if the response is the GetDeviceData resposne.
        if (check(7:8) == '0F')
            if ~checkCrc(getser)
                checkReturnCode(getser);
                serNr = check(13:end-6);
                for i = 1:17*2
                    if serNr(i) == '0'
                        k = k + 1;
                    else
                        break;
                    end
                end
                serNr = serNr(k:end);
                serNr = hex2char(serNr);
                % Updates the sequence number.
                slast = Slast(getser);
                done = 0;
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);
                return;
            end
        end
    else
        % Polling with Receiver Ready commands.
        cmd = receiverReadyCMD(Addr,slast);
        fwrite(RET,hex2dec(cmd),'uint8');
    end
    drawnow;
end

end
```

Listing B.28: Get Device Data: 0x03 response

```
function [r,min,max] = getBand3(input)
%GETBAND3 Gets the operating bands from the command getBandwidthCMD
%   Returns the bandwidths within the 0x03 field and if field 0x08 and/or
%   field 0x09 is defined.
%   Input:  input:  Anwser from the RET.
%   Output: r:      If field 0x08 (8) or 0x09 (9) or both (98) is defined.
%                   Otherwise (0).
%           min:    Minimum frequency.
%           max:    Maximum frequency.

% Converts the awser.
resp = convertAnswer(input);
% Codes for the row in frequnecy bands matrix.
opBandCode3 = [-1 -2 14 13 12 11 10 9 8 7 1 2 3 4 5 6];
FreqBands = [1920 2170;1850 1990;1710 1880;1710 2155;824 894;830 885;
    2500 2690;880 960;1749.9 1879.9;1710 2170;1427.9 1495.9;699 746;
    777 756;758 798;0 0;0 0;704 746;815 875;830 890;791 862;1447.9 1510.9;
    3410 3590;2000 2200;1626.5 1559;1850 1995;814 894;807 869;703 803;0 0;
    2305 2360;452.5 467.5;0 0;1900 1920;2010 2025;1850 1910;1930 1990;
    1910 1930;2570 2620;1880 1920;2300 2400;2496 2690;3400 3600;3600 3800;
    703 803;1447 1467;5150 5925;0 0;3550 3700;0 0;0 0;0 0;0 0;0 0;0 0;0 0;
    0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;1920 2200;1710 2200;0 0;698 783;
    0 0;1695 2020];
% Extracts the input value from the answer (little endian).
inte = dec2bin(hex2dec(strcat(resp(end-7),resp(end-6),resp(end-9),...
    resp(end-8)))) - '0';
% Makes sure it is 2 byte long.
while length(inte) < 16
    inte = [0 inte];
end
j=1;
% Preallocating.
band = zeros(1,length(find(inte=='1')));
% Coverts the input value into code words.
for i = 1:length(inte)
    if inte(i) == 1
        band(j) = opBandCode3(i);
        j=j+1;
    end
end
% Checks if 0x08 and/or 0x09 is defined.
if band(1) == -1
    r = 9;
    bw = band(2:end);
    if band(2) == -2
        r = 98;
        bw = band(3:end);
    end
elseif band(1) == -2
    r = 8;
```

```
    bw = band(2:end);
else
    r = 0;
    bw = band;
end
% Compares the code words with the frequency band list and takes the
% absolute min and max values.
for k = 1:length(bw)
    n = FreqBands(bw(k),1);
    x = FreqBands(bw(k),2);
    if k == 1
        min = n;
        max = x;
    end
    if n < min
        min = n;
    end
    if x > max
        max = x;
    end
end

end
```

Listing B.29: Get Device Data: 0x08 response

```
function [min,max] = getBand8(input)
%GETBAND8 Returns the operating bands if the field 0x08 is defined
%   Returns the minimum mand maximum bandwidth in the 0x08 field.
%   Input:  input:  Anwser from the RET.
%   Output: min:    Minimum frequency.
%           max:    Maximum frequency.

resp = convertAnswer(input);
% Codes for the row in frequnecy bands matrix.
opBandCode8 = [30 29 28 27 26 25 24 23 22 21 20 19 18 17 0 0];
FreqBands = [1920 2170;1850 1990;1710 1880;1710 2155;824 894;830 885;
    2500 2690;880 960;1749.9 1879.9;1710 2170;1427.9 1495.9;699 746;
    777 756;758 798;0 0;0 0;704 746;815 875;830 890;791 862;1447.9 1510.9;
    3410 3590;2000 2200;1626.5 1559;1850 1995;814 894;807 869;703 803;0 0;
    2305 2360;452.5 467.5;0 0;1900 1920;2010 2025;1850 1910;1930 1990;
    1910 1930;2570 2620;1880 1920;2300 2400;2496 2690;3400 3600;3600 3800;
    703 803;1447 1467;5150 5925;0 0;3550 3700;0 0;0 0;0 0;0 0;0 0;0 0;0 0;
    0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;1920 2200;1710 2200;0 0;698 783;
    0 0;1695 2020];
% Extracts the input value from the answer (little endian).
inte = dec2bin(hex2dec(strcat(resp(end-7),resp(end-6),resp(end-9),...
    resp(end-8)))) - '0';
% Makes sure it is 2 byte long.
while length(inte) < 16
    inte = [0 inte];
end
j=1;
% Preallocating.
bw = zeros(1,length(find(inte=='1')));
% Coverts the input value into code words.
for i = 1:length(inte)
    if inte(i) == 1
        bw(j) = opBandCode8(i);
        j=j+1;
    end
end
% Compares the code words with the frequency band list and takes the
% absolute min and max values.
for k = 1:length(bw)
    n = FreqBands(bw(k),1);
    x = FreqBands(bw(k),2);
    if k == 1
        min = n;
        max = x;
    end
    if n < min
        min = n;
    end
    if x > max
        max = x;
    end
end
end
```

Listing B.30: Get Device Data: 0x09 response

```
function [r, min, max] = getBand9(input)
%GETBAND9 Returns the operating bands if the field 0x09 is defined
%   Returns the minimum mand maximum bandwidth in the 0x09 field.
%   Input:  input:  Anwser from the RET.
%   Output: min:    Minimum frequency.
%           max:    Maximum frequency.

resp = convertAnswer(input);
% Codes for the row in frequnecy bands matrix.
```

```
opBandCode9 = [-1 65 32 31 44 43 42 41 40 39 38 37 36 35 34 33];
FreqBands = [1920 2170;1850 1990;1710 1880;1710 2155;824 894;830 885;
    2500 2690;880 960;1749.9 1879.9;1710 2170;1427.9 1495.9;699 746;
    777 756;758 798;0 0;0 0;704 746;815 875;830 890;791 862;1447.9 1510.9;
    3410 3590;2000 2200;1626.5 1559;1850 1995;814 894;807 869;703 803;0 0;
    2305 2360;452.5 467.5;0 0;1900 1920;2010 2025;1850 1910;1930 1990;
    1910 1930;2570 2620;1880 1920;2300 2400;2496 2690;3400 3600;3600 3800;
    703 803;1447 1467;5150 5925;0 0;3550 3700;0 0;0 0;0 0;0 0;0 0;0 0;0 0;
    0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;1920 2200;1710 2200;0 0;698 783;
    0 0;1695 2020];
% Extracts the input value from the answer (little endian).
inte = dec2bin(hex2dec(strcat(resp(end-7),resp(end-6),resp(end-9),...
    resp(end-8)))) - '0';
% Makes sure it is 2 byte long.
while length(inte) < 16
    inte = [0 inte];
end
j=1;
% Preallocating.
band = zeros(1,length(find(inte=='1')));
 % Coverts the input value into code words.
for i = 1:length(inte)
    if inte(i) == 1
        band(j) = opBandCode9(i);
        j=j+1;
    end
end
% Checks if 0x08 and/or 0x09 is defined.
if band(1) == -1
    r = 10;
    bw = band(2:end);
else
    r = 0;
    bw = band;
end
% Compares the code words with the frequency band list and takes the
% absolute min and max values.
for k = 1:length(bw)
    n = FreqBands(bw(k),1);
    x = FreqBands(bw(k),2);
    if k == 1
        min = n;
        max = x;
    end
    if n < min
        min = n;
    end
    if x > max
        max = x;
    end
end
end
```

Listing B.31: Get Device Data: 0x0A response

```
function [min,max] = getBandA(input)
%GETBANDA Returns the operating bands if the field 0x08 is defined
%   Returns the minimum mand maximum bandwidth in the 0x08 field.
%   Input:  input:  Anwser from the RET.
%   Output: min:    Minimum frequency.
%           max:    Maximum frequency.

resp = convertAnswer(input);
% Codes for the row in frequnecy bands matrix.
opBandCodeA = [0 0 0 0 0 0 0 0 0 0 0 46  68 45 67 66];
FreqBands = [1920 2170;1850 1990;1710 1880;1710 2155;824 894;830 885;
    2500 2690;880 960;1749.9 1879.9;1710 2170;1427.9 1495.9;699 746;
    777 756;758 798;0 0;0 0;704 746;815 875;830 890;791 862;1447.9 1510.9;
    3410 3590;2000 2200;1626.5 1559;1850 1995;814 894;807 869;703 803;0 0;
    2305 2360;452.5 467.5;0 0;1900 1920;2010 2025;1850 1910;1930 1990;
    1910 1930;2570 2620;1880 1920;2300 2400;2496 2690;3400 3600;3600 3800;
    703 803;1447 1467;5150 5925;0 0;3550 3700;0 0;0 0;0 0;0 0;0 0;0 0;0 0;
    0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;0 0;1920 2200;1710 2200;0 0;698 783;
    0 0;1695 2020];
% Extracts the input value from the answer (little endian).
inte = dec2bin(hex2dec(strcat(resp(end-7),resp(end-6),resp(end-9),...
    resp(end-8)))) - '0';
% Makes sure it is 2 byte long.
while length(inte) < 16
    inte = [0 inte];
end
j=1;
% Preallocating.
bw = zeros(1,length(find(inte=='1')));
% Coverts the input value into code words.
for i = 1:length(inte)
    if inte(i) == 1
        bw(j) = opBandCodeA(i);
        j=j+1;
    end
end
% Compares the code words with the frequency band list and takes the
% absolute min and max values.
```

```
for k = 1:length(bw)
    n = FreqBands(bw(k),1);
    x = FreqBands(bw(k),2);
    if k == 1
        min = n;
        max = x;
    end
    if n < min
        min = n;
    end
    if x > max
        max = x;
    end
end
end
```

Listing B.32: Create Get Device Data command

```
function [cmd] = getBandwidthCMD(Addr,sqNr,code)
%GETBANDWIDTHCMD The secondary device shall return the Bandwidth span.
%   Input:  Addr: the HDLC address of the RET.
%           sqNr: Control frame of the latest recieved frame.
%   Output: The whole HDLC frame with the Get Bandwidth command.

Flag = '7E';
ctrl = calculateControlF(sqNr,'I');
cmd = '0F';                                     % Get device data code.
cmdData = strcat('0100',code);                  % Antenna operating band.
tocrc = strcat(Addr,ctrl,cmd,cmdData);
crc = calculateCRC(tocrc);
cmd = strsplit(addSpace(strcat(Flag,tocrc,crc,Flag)));
end
```

Listing B.33: Get COM ports

```
function [text] = getCOMs()
%GETCOMS Gets a list of all the COM ports available.

% Runs the mode in the system cmd to get all info of avaiable COM-ports
[~,res] = system('mode');
% Extracts just the COMX ports.
ports = regexp(res,'COM\d+','match')';
text = reshape(ports,1,[]);
if isempty(text)
    text = 'NaN';
end
end
```

Listing B.34: Get Information response

```
function [ProdNr,SerNr,HwVer,SwVer,port] = getInfo(info)
%GETINFO Handels the resopnse of getInformation command.
%   Input:  The response form the RET (string).
%   Output: Product number, Serial number, Hardware version and Software
%   version (Strings).

info = convertAnswer(info);
port = '';

% Extracts the length of the product number.
proLen = hex2dec(strcat(info(15),info(16)));
% Extracts the product number.
a = info(17:16+proLen*2);
ProdNr = hex2char(a);

% Extracts the length of the serial number.
serLen = hex2dec(strcat(info(17+proLen*2),info(18+proLen*2)));
% Extracts the serial number.
b = info(19+proLen*2:18+proLen*2+serLen*2);
SerNr = hex2char(b);
% If the serial number contains the port name, extract it.
if SerNr(end-2) == '-'
    port = SerNr(end-1:end);
end

% Extracts the length if the hardware version.
hwLen = hex2dec(strcat(info(19+proLen*2+serLen*2),...
    info(20+proLen*2+serLen*2)));
% Exracts the hardware version.
c = info(21+proLen*2+serLen*2:20+proLen*2+serLen*2+hwLen*2);
HwVer = hex2char(c);

% Extracts the length if the software version.
swLen = hex2dec(strcat(info(21+proLen*2+serLen*2+hwLen*2),...
    info(22+proLen*2+serLen*2+hwLen*2)));
% Exracts the software version.
d = info(23+proLen*2+serLen*2+hwLen*2:22+proLen*2+serLen*2+hwLen*2+swLen*2);
SwVer = hex2char(d);
end
```

Listing B.35: Create Get Information command

```
function [getInfo] = getInfoCMD(Addr,sqNr)
%getInfoCMD Returns product and serial number.
%           If known also hardware and software version
%   Input:  Addr: the HDLC address of the RET.
%           sqNr: Control frame of the latest recieved frame.
%   Output: The whole HDLC frame with the Get Infromation command.

Flag = '7E';
ctrl = calculateControlF(sqNr,'I');
cmd = '05';                                          % Get Information code.
cmdData = '0000';
tocrc = strcat(Addr,ctrl,cmd,cmdData);
crc = calculateCRC(tocrc);
getInfo = strsplit(addSpace(strcat(Flag,tocrc,crc,Flag)));
end
```

Listing B.36: getInformation

```
function [ProdNr,SerNr,HwVer,SwVer,port,slast] = getInformation(Addr,slast,RET,stop)
%GETINFORMATION Returns product and serial number. If known also hardware
%               and software version.
%   Input:  Addr:   Address of the secondary device.
%           slast:  Sequence number.
%           RET:    Serial port object.
%   Output: ProdNr: Product number.
%           SerNr:  Serial number.
%           HwVer:  Hardware version.
%           SwVer:  Software version.
%           slast:  The next sequence number.
done = 1;
port = '';
% Creates the GetInformation command.
cmd = getInfoCMD(Addr,slast);
% Writes the GetInformation command to the serial port.
fwrite(RET,hex2dec(cmd),'uint8');

 % Continously reads the serial port untill the right response is received.
 % or user presses the reset button.
while (done) && get(stop,'userdata')
    info=fread(RET);
    if ~isnan(info)
        check = convertAnswer(info);
        % Checks if the response is the GetInformation resposne.
        if (check(7:8) == '05')
            if ~checkCrc(info)
                checkReturnCode(info);
                 % Updates the sequence number.
                slast = Slast(info);
                done = 0;
                % Extracts the Product number, Serial number, hardware
                % version and sofware version from the response.
                [ProdNr,SerNr,HwVer,SwVer,port] = getInfo(info);
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);
                return;
            end
        end
    else
        % Polling with Receiver Ready commands.
        cmd = receiverReadyCMD(Addr,slast);
        fwrite(RET,hex2dec(cmd),'uint8');
    end
    drawnow;
end
end
```

Listing B.37: getMinMaxTilt

```
function [min,max,slast] = getMinMaxTilt(Addr,slast,RET,stop)
%GETMINMAXTILT  Function that returns the Minimum and Maximum supported
%               electrical tilt in degrees.
%   Input:  Addr:   Address of the secondary device.
%           slast:  Sequence number.
%           RET:    Serial port object.
%   Output: min:    Minimum electrical tilt value in degrees.
%           max:    Maximum electrical tilt value in degrees.
%           slast:  The next sequence number.
done = 1;
% Creates the GetDeviceData command with get Maximum electrical tilt.
cmd = getBandwidthCMD(Addr,slast,'06');
% Writes the GetDeviceData command to the serial port.
fwrite(RET,hex2dec(cmd),'uint8');


while (done) && get(stop,'userdata')
    maxi=fread(RET,24);
    if ~isnan(maxi)
        check = convertAnswer(maxi);
        % Checks if the response is the GetDeviceData resposne.
```

```
        if (check(7:8) == '0F')
            if ~checkCrc(maxi)
                checkReturnCode(maxi);
                % Updates the sequence number.
                slast = Slast(maxi);
                done = 0;
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);

                done = 1;
                % Creates the GetDeviceData command with get Minimum electrical tilt.
                cmd = getBandwidthCMD(Addr,slast,'07');
                % Writes the GetDeviceData command to the serial port.
                fwrite(RET,hex2dec(cmd),'uint8');

                % Continously reads the serial port untill the right response
                % is received or if the user presses the reset button.
                while (done) && get(stop,'userdata')
                    mini=fread(RET,24);
                    if ~isnan(mini)
                        check = convertAnswer(mini);
                        % Checks if the response is the
                        % GetDeviceData resposne.
                        if (check(7:8) == '0F')
                            if ~checkCrc(mini)
                                checkReturnCode(mini);
                                % Updates the sequence number.
                                slast = Slast(mini);
                                done = 0;
                                max = getTilt(maxi);
                                min = getTilt(mini);
                                % Polling with Receiver Ready commands.
                                delayBetweenCMD(Addr,slast,RET);
                                return;
                            end
                        else
                            % Polling with Receiver Ready commands.
                            cmd = receiverReadyCMD(Addr,slast);
                            fwrite(RET,hex2dec(cmd),'uint8');
                        end
                    end
                    drawnow;
                end
            end
        else
            % Polling with Receiver Ready commands.
            cmd = receiverReadyCMD(Addr,slast);
            fwrite(RET,hex2dec(cmd),'uint8');
        end
    end
    drawnow;
end

end
```

Listing B.38: getOperatingBand

```
function [min,max,slast] = getOperatingBand(Addr,sqNr,RET,stop)
%GETOPERATINGBAND Function that returnes the antennas operating bands.
%   Input:  Addr:   Address of the secondary device.
%           slast:  Sequence number.
%           RET:    Serial port object.
%   Output: min:    The lowest frequency band.
%           max:    The highest frequency band.
%           slast:  The next sequence number.
done = 1;
% Creates the GetDeviceData command with Antenna operating band selected.
cmd = getBandwidthCMD(Addr,sqNr,'03');
% Writes the GetDeviceData command to the serial port.
fwrite(RET,hex2dec(cmd),'uint8');

% Continously reads the serial port untill the right response is received.
% or if the user presses the reset button.
while (done) && get(stop,'userdata')
    bw=fread(RET,24);
    if ~isnan(bw)
        check = convertAnswer(bw);
        if (check(7:8) == '0F')
            if ~checkCrc(bw)
                checkReturnCode(bw);
                % Updates the sequence number.
                slast = Slast(bw);
                done = 0;
                % Gets the operating bands and if there are additional operating bands.
                [r1, min, max] = getBand3(bw);
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);
            end
        else
            % Polling with Receiver Ready commands.
            cmd = receiverReadyCMD(Addr,sqNr);
            fwrite(RET,hex2dec(cmd),'uint8');
        end
```

```
37
38      end
39      drawnow;
40  end
41  if get(stop,'userdata')
42      % If bit 15 where set in the first response, field 0x09 is defined.
43      if r1 == 9 || r1 == 98
44          done = 1;
45          % Creates the GetDeviceData command with Antenna operating band selected with the 0x09
                field defined.
46          cmd = getBandwidthCMD(Addr,slast,'09');
47          fwrite(RET,hex2dec(cmd),'uint8');
48          % Continously reads the serial port untill the right response is received.
49          % or if the user presses the reset button.
50          while (done) && get(stop,'userdata')
51              bw2=fread(RET,24);
52              if ~isnan(bw2)
53                  check = convertAnswer(bw2);
54                  if (check(7:8) == '0F')
55                      if ~checkCrc(bw2)
56                          checkReturnCode(bw2);
57                          % Updates the sequence number.
58                          slast = Slast(bw2);
59                          done = 0;
60                          % Gets the operating bands and if there are an additional operating band
                                .
61                          [r2,min2,max2] = getBand9(bw2);
62                          % Compares the new minimum and maximum values and select the highest and
                                lowest of them.
63                          if min2 < min
64                              min = min2;
65                          end
66                          if max2 > max
67                              max = max2;
68                          end
69                      end
70                  else
71                      % Polling with Receiver Ready commands.
72                      cmd = receiverReadyCMD(Addr,sqNr);
73                      fwrite(RET,hex2dec(cmd),'uint8');
74                  end
75              end
76              drawnow;
77          end
78          if get(stop,'userdata')
79              % If bit 15 where set in the first and second response, field 0x0A is defined.
80              if r2 == 10
81                  % Polling with Receiver Ready commands.
82                  delayBetweenCMD(Addr,slast,RET);
83
84                  done = 1;
85                  % Creates the GetDeviceData command with Antenna operating band selected with
                        the 0x0A field defined.
86                  cmd = getBandwidthCMD(Addr,slast,'0A');
87                  fwrite(RET,hex2dec(cmd),'uint8');
88
89                  % Continously reads the serial port untill the right response is received.
90                  % or if the user presses the reset button.
91                  while (done) && get(stop,'userdata')
92                      bw3=fread(RET,24);
93                      if ~isnan(bw3)
94                          check = convertAnswer(bw3);
95                          if (check(7:8) == '0F')
96                              if ~checkCrc(bw3)
97                                  checkReturnCode(bw3);
98                                  % Updates the sequence number.
99                                  slast = Slast(bw3);
100                                 done = 0;
101
102                                 [min4,max4] = getBandA(resp4);
103                                 % Compares the new minimum and maximum values and select the
                                        highest and lowest of them.
104                                 if min4 < min
105                                     min = min4;
106                                 end
107                                 if max4 > max
108                                     max = max4;
109                                 end
110                             end
111                         end
112                     else
113                         % Polling with Receiver Ready commands.
114                         cmd = receiverReadyCMD(Addr,sqNr);
115                         fwrite(RET,hex2dec(cmd),'uint8');
116                     end
117                     drawnow;
118                 end
119             end
120         end
121     % If bit 14 where set in the first response, field 0x08 is defined.
122     elseif r1 == 8 || r1 == 98
123         done = 1;
124         % Creates the GetDeviceData command with Antenna operating band selected with the 0x08
                field defined.
125         cmd = getBandwidthCMD(Addr,slast,'08');
```

```
126         fwrite(RET,hex2dec(cmd),'uint8');
127         % Continously reads the serial port untill the right response is received.
128         % or if the user presses the reset button.
129         while (done) && get(stop,'userdata')
130             bw2=fread(RET,24);
131             if ~isnan(bw2)
132                 check = convertAnswer(bw2);
133                 if (check(7:8) == '0F')
134                     if ~checkCrc(bw2)
135                         checkReturnCode(bw2);
136                         % Updates the sequence number.
137                         slast = Slast(bw2);
138                         done = 0;
139                         % Gets the operating bands.
140                         [min3,max3] = getBand8(bw2);
141                         % Compares the new minimum and maximum values and select the highest and
                                lowest of them.
142                         if min3 < min
143                             min = min3;
144                         end
145                         if max3 > max
146                             max = max3;
147                         end
148                     end
149                 else
150                     % Polling with Receiver Ready commands.
151                     cmd = receiverReadyCMD(Addr,sqNr);
152                     fwrite(RET,hex2dec(cmd),'uint8');
153                 end
154             end
155             drawnow;
156         end
157     end
158     % Polling with Receiver Ready commands.
159     delayBetweenCMD(Addr,slast,RET);
160 end
161 end
```

Listing B.39: getSparam

```
1  function [] = getSparam(file,path)
2  %GETSPARAM Gets the S-parameters from the Network Analyzer.
3  % output:   Return the complex response values of all traces. The traces in
4  %            the catalog list are read one after another: The response array
5  %            contains n (number of points) pairs of real and imaginary values
6  %            for S11, followed by n pairs of values for S12, S21, and S22.
7
8  try
9      % Path for the tcpip functions. Change if nessesary.
10     addpath '\\dfns\proj\antennmatning\A-15 Matstracka\Matlab program\Matlab_TCPIP'
11     s=tcpip_open('10.75.145.14','5025');
12      tcpip_write(s, ['*IDN?', 10])
13      r = tcpip_read(s);
14      if ~isnan(r)
15         newPathStart = strfind(path,'A-15');
16         newPath = path(newPathStart:end);
17
18 %          tcpip_write(s,['CALC1:PAR:SDEF ''' 'Trc1' '''' ',' '''' 'S11' '''', 10]);  %Def Sparam
19 %          tcpip_write(s,['CALC1:PAR:SDEF ''' 'Trc2' '''' ',' '''' 'S21' '''', 10]);
20 %          tcpip_write(s,['CALC1:PAR:SDEF ''' 'Trc3' '''' ',' '''' 'S12' '''', 10]);
21 %          tcpip_write(s,['CALC1:PAR:SDEF ''' 'Trc4' '''' ',' '''' 'S22' '''', 10]);
22         tcpip_write(s, ['FREQ:STAR 650Mhz', 10]);  %start freq
23         tcpip_write(s, ['FREQ:STOP 2.7Ghz', 10]);  %stop freq
24         tcpip_write(s, ['SENS1:BANDwidth:RESolution 1000', 10]);
25         tcpip_write(s, ['SENS1:SWE:POIN 2051', 10]); % number of points
26         i = strcat('Z:\',newPath,'\',file);
27         news = strcat('MMEM:STOR:TRAC:CHAN 1,','  '' ',i,'''','');
28         tcpip_write(s,[news, 10]);
29         tcpip_close(s)
30     end
31 catch
32     msgbox('Cant connect to the Network Analyzer.','Error')
33 end
34
35 end
```

Listing B.40: getTilt

```
1  function [tilt] = getTilt(resp)
2  %getTilt Handels the response from the Get Tilt command.
3  %   Input:  Response string.
4  %   Output: Tilt value in degrees.
5
6  resp = convertAnswer(resp);
7  % Extracts the tilt value.
8  inte = hex2dec(strcat(resp(end-7),resp(end-6),resp(end-9),resp(end-8)))*0.1;
9  tilt = inte;
10 end
```

Listing B.41: Creates the Get Tilt command

```
function [getTilit] = getTiltCMD(Addr,sqNr)
%getTiltCMD On receipt of the initiating message the secondary
%           device shall return the current tilt value.
%   Input:  Addr: the HDLC address of the RET.
%           sqNr: Control frame of the latest recieved frame.
%   Output: The whole HDLC frame with the getTilit command.

Flag = '7E';
ctrl = calculateControlF(sqNr,'I');
cmd = '34';                                          % Get Tilit code.
cmdData = '0000';
tocrc = strcat(Addr,ctrl,cmd,cmdData);
crc = calculateCRC(tocrc);
getTilit = strsplit(addSpace(strcat(Flag,tocrc,crc,Flag)));
end
```

Listing B.42: Get Tilt for GUI

```
function [ value,slast ] = getTiltGui(Addr,slast,RET,stop)
%GETTILTGUI Summary of this function goes here
%   Detailed explanation goes here
done = 1;
% Creates the GetTilt command.
cmd2 = getTiltCMD(Addr,slast);
% Writes the GetTilt command to the serial port.
fwrite(RET,hex2dec(cmd2),'uint8');

% Continously reads the serial port untill the right response is received.
% or if the users presses the reset button.
while (done) && get(stop,'userdata')
    currentTilt = fread(RET,22);
    if ~isnan(currentTilt)
        check = convertAnswer(currentTilt);
        % Checks if the response is the GetTilt resposne.
        if (check(7:8) == '34')
            if ~checkCrc(currentTilt)
                % Extracts the current tilt value.
                value = getTilt(currentTilt);
                % Updates the sequence number.
                slast = Slast(currentTilt);
                done = 0;
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);
            end
        else
            % Polling with Receiver Ready commands.
            cmd = receiverReadyCMD(Addr,slast);
            fwrite(RET,hex2dec(cmd),'uint8');
        end
    end
    drawnow;
end
end
```

lstinputlisting[caption = Get vendor]MATLAB/getVendor.m

Listing B.43: Get response from XID

```
function [Addr,SN,VnCd,DeTy] = getXID(answer)
%GETFIRSTXID Handels the response from the XID command.
%   Input:  Response string.
%   Output: Depending on what was asked for in the command.

VnCd = '';
DeTy = '';
anw = convertAnswer(answer);
% Extracts the HDLC address.
Addr = strcat(anw(3),anw(4));
totlen = hex2dec(strcat(anw(11),anw(12)));
lenFirst = hex2dec(strcat(anw(15),anw(16)));

% Extracts the Serial number of the RET.
firstAnw = anw(17:16+lenFirst*2);
SN = firstAnw;

type = (strcat(anw(17+lenFirst*2),anw(18+lenFirst*2)));
lenSecond=hex2dec(strcat(anw(19+lenFirst*2),anw(20+lenFirst*2)));
secondAnw = anw(21+lenFirst*2:20+lenFirst*2+lenSecond*2);
switch type
    case '04'
        DeTy = secondAnw;
    case '06'
        VnCd = secondAnw;
end

a = 8+2*(lenFirst+lenSecond);
% If there is a thrid information item.
if (a) < (totlen*2)
```

```
    type2 = (strcat(anw(21+lenSecond*2+lenFirst*2),...
        anw(22+lenSecond*2+lenFirst*2)));
    lenThird = hex2dec(strcat(anw(23+lenSecond*2+lenFirst*2),...
        anw(24+lenSecond*2+lenFirst*2)));
    thirdAnw = anw(25+lenFirst*2+lenSecond*2:24+lenFirst*2+lenSecond*2+lenThird*2);
    switch type2
        case '04'
            DeTy = thirdAnw;
        case '06'
            VnCd = thirdAnw;
    end
end
end
```

Listing B.44: Hexadecimal to char conversion

```
function [c] = hex2char(hx)
% Converts hexadecimal number into a character according to the ASCII
% table.
c = char(sscanf(hx,'%2X').');
end
```

Listing B.45: Open serial connection

```
function [Addr,SN,VnCd,DeTy,slast,RET] = openConnection(com,Vn,stop)
%OPENCONNECTION Creates the serial element all calls for a device scan and
%address asignment, XID (Exchange Identification).
%       Input:  com:    COM port.
%               Vn:     Vendor code.
%       Output: Addr:   List of all the HDLC addresses of the connected
%                       RETs.
%               SN:     List of all the RETs Serial numbers.
%               VnCd:   List of all the RETs Vendor Codes.
%               DeTy:   List of all the RETs device types.
%               slast:  List of all the RETs sequence numbers.
%               RET:    Serial element.
delete(instrfindall);
% Create serial object
RET=serial(com,'TimeOut',0.02);
% Open transmissionA
fopen(RET);
% Calls for device scan and address asignment.
[Addr,SN,VnCd,DeTy]=callXID(RET,Vn,stop);
% Initialize sequence number for each RET.
slast = cell(1,length(Addr));
slast(1:end) = {'11'};
end
```

Listing B.46: Creates the RR command

```
function [rr] = receiverReadyCMD(Addr,SqNr)
%RRCMD  This command and response is used to inform the opposite station
%       (primary or secondary) that the transmitting station has empty buffers,
%       i.e. is ready to receive an I frame.
%   Input:  Addr: the HDLC address of the RET.
%           sqNr: Control frame of the latest recieved frame.
%   Output: The whole HDLC frame with the Receiver Ready command.

Flag = '7E';
ctrl = calculateControlF(SqNr,'S');
tocrc = strcat(Addr,ctrl);
crc = calculateCRC(tocrc);
rr = strcat(Flag,tocrc,crc,Flag);
rr = strsplit(addSpace(rr));

end
```

Listing B.47: Remove space between octets

```
function [out] = removeSpace(string)
%REMOVESPACE Removes the spaces in a string.
out = regexprep(string,'[^\w'']','');
end
```

Listing B.48: Reset

```
function [] = RSET(Addr,RET)
%RSET Sends the RSET command to the secondary station indicating to
%       reset its sequence numbers.
%   Input:  Addr:   HDLC address.
%           RET:    Serial object.

donerset = 1;
```

```
% Creates the command.
rset = rsetCMD(Addr);
% Writes the command to the serial object.
fwrite(RET,hex2dec(rset),'uint8');
while donerset
    rsetAnw = fread(RET);
    if ~isnan(rsetAnw)
        % Check if answer is corrupt.
        b = checkCrc(rsetAnw);
        if b
            fwrite(RET,hex2dec(rset),'uint8');
        else
            donerset = 0;
        end
    end
end
end
```

Listing B.49: Create Reset command

```
function [RSET_CMD] = rsetCMD(Addr)
%discCMD    Reset. This command is used to reset the
%           secondary station.
%   Input:  Addr: the HDLC address of the RET.
%   Output: The whole HDLC frame with the RSET command.

RSET = '9F';                                    % RSET code.
crc = calculateCRC(strcat(Addr,RSET));
RSET_CMD = strcat('7E',Addr,RSET,crc,'7E');
RSET_CMD = strsplit(addSpace(RSET_CMD));
end
```

Listing B.50: setTilt

```
function [value,ok,slast] = setTilt(Addr,slast,RET,value,stop)
%SETTILT Sets a desired tilt value.
%   Input:  Addr:   Address of the secondary device.
%           slast:  Sequence number.
%           RET:    Serial port object.
%           value:  Desired tilt value in degrees.
%   Output: value:  The new tilt value.
%           ok:     1 if the current and desired tilt values are the same,
%                   otherwise 0.
%           slast:  The next sequence number.

% Converts the decimal tilt value into hexadecimal.
value = dec2hex(value*10,4);
value = strcat(value(3),value(4),value(1),value(2));
% Creates the SetTilt command.
cmd = setTiltCMD(Addr,slast,value);
% Writes the SetTilt command to the serial port.
fwrite(RET,hex2dec(cmd),'uint8');
t = tic;
% Continously reads the serial port untill the right response is received.
% or untill t = 2 min or if the user presses the reset button.
while (toc(t) < 120) && get(stop,'userdata')
    tiltok = fread(RET,20);
    if ~isnan(tiltok)
        check1 = convertAnswer(tiltok);
        % Checks if the response is the SetTilt resposne.
        if (check1(7:8) == '33')
            if ~checkCrc(tiltok)
                % Updates the sequence number.
                slast = Slast(tiltok);
                % Polling with Receiver Ready commands.
                delayBetweenCMD(Addr,slast,RET);
                % Creates the GetTilt command.
                cmd2 = getTiltCMD(Addr,slast);
                % Writes the GetTilt command to the serial port.
                fwrite(RET,hex2dec(cmd2),'uint8');
                %  Reads the serial port 3 times.
                for i = 1:3
                    currentTilt = fread(RET,22);
                    if ~isnan(currentTilt)
                        check = convertAnswer(currentTilt);
                        % Checks if the response is the GetTilt resposne.
                        if (check(7:8) == '34')
                            if ~checkCrc(currentTilt)
                                % Extracts the current tilt value.
                                ctilt = getTilt(currentTilt);
                                % Compares the current and desired tilt values.
                                [value,ok] = compareTilt(value,ctilt);
                                if ok
                                    % Updates the sequence number.
                                    slast = Slast(currentTilt);
                                    % Polling with Receiver Ready commands.
                                    delayBetweenCMD(Addr,slast,RET);
                                    return;
                                else
                                    error('Tilt not set');
                                end
```

```
                            end
                        else
                            % Polling with Receiver Ready commands.
                            cmd = receiverReadyCMD(Addr,slast);
                            fwrite(RET,hex2dec(cmd),'uint8');
                        end
                    end
                end
            end
        else
            % Polling with Receiver Ready commands.
            cmd = receiverReadyCMD(Addr,slast);
            fwrite(RET,hex2dec(cmd),'uint8');
        end
    end
    drawnow;
end

end
```

Listing B.51: Create the Set Tilt command

```
function [setTilt] = setTiltCMD(Addr,sqNr,tilt)
%setTiltCMD On receipt of the initiating message the secondary device shall
%           set the electrical tilt of the antenna addressed by the antenna
%           number in increments of 0.1 degree.
%   Input:  Addr: the HDLC address of the RET.
%           sqNr: Control frame of the latest recieved frame.
%           tilt: The value of the tilit.
%   Output: The whole HDLC frame with the Set Tilt command.

Flag = '7E';
ctrl = calculateControlF(sqNr,'I');
cmd = '33';                                     % Set Tilit code.
dataLen = num2str(dec2hex(length(tilt)/2)); % Calculates the length
                                                % of the data.
if length(dataLen) < 2                          % Makes sure it is two octets.
    dataLen = strcat('0',dataLen);
end
dataLen = strcat(dataLen,'00');
tocrc = strcat(Addr,ctrl,cmd,dataLen,tilt);
crc = calculateCRC(tocrc);
setTilt = strcat(Flag,tocrc,crc,Flag);
setTilt = strsplit(addSpace(setTilt));
end
```

Listing B.52: Set Tilt for GUI

```
function [slast] = setTiltGUI(okTxt,tilttxt,fig,conOK,value,Addr,slast,RET,min,max,stop)
%SETTILTGUI Executes the set Tilt command with some error handling.
%       Input:  okTxt:      Text element for setTilt.
%               tilttxt:    Text for the current tilt value.
%               fig:        Element for the pointer.
%               conOK:      If connected.
%               Addr:       HDLC Address of the RET.
%               slast:      Sequence Number of the RET.
%               RET:        Serial element.
%               min:        Minimum down tilt.
%               max:        Maximum down tilt.
%       Output: slast:  New sequence number of the RET.

% Checks if connected.
if conOK
    % Controlls that a value has been entered.
    if ~isempty(value)
        % Converts the value.
        value = str2double(value);
        % Controlls that the value is within the specified range.
        if value <= max && value >= min
            oldpointer = get(fig, 'pointer');
            set(fig, 'pointer', 'watch')
            drawnow;
            try
                % Sets the new tilt value.
                [val,ok,slast] = setTilt(Addr,slast,RET,value,stop);
                if ok
                    set(tilttxt,'string',val);
                    set(okTxt,'string','Tilt value set.');
                else
                    set(okTxt,'string','Error: Tilt value not set.');
                end
            catch
                set(okTxt,'string','Error: Tilt value not set.');
            end
            set(fig, 'pointer', oldpointer)

        else
            set(okTxt,'string','Error: Value needs to be within the Down Tilt range.');
        end
    else
        set(okTxt,'string','Error: No tilt value entered.');
```

```
    end
else
    set(okTxt,'string','Error: Not connected.');
end
end
```

Listing B.53: Slast

```
function [slast] = Slast(rFrame)
%SLAST Extracts the control frame from HDLC frame.
rFrame=convertAnswer(rFrame);
slast = strcat(rFrame(5),rFrame(6));
end
```

Listing B.54: Creates the SNRM command

```
function [SNRM_CMD] = snrmCMD(Addr)
%snrmCMD    Set Normal Response Mode (SNRM). This command is used to set the
%           secondary station in connected mode and reset its sequence
%           number variables.
%   Input:  Addr: the HDLC address of the RET.
%   Output: The whole HDLC frame with the SNRM command.

SNRM = '93';                                % SNRM code.
crc = calculateCRC(strcat(Addr,SNRM));
SNRM_CMD = strcat('7E',Addr,SNRM,crc,'7E');
SNRM_CMD = strsplit(addSpace(SNRM_CMD));
end
```

Listing B.55: Creates the XID command

```
function [b] = XID(PI1,data1,PI2,data2)
%XID Exchange Identification
%   Values of PI1 and PI2.
%   bit0: 0x01: Unique ID (1-19 octet(s))
%   bit1: 0x02: HDLC Address (1 octet)
%   bit2: 0x03: Bit Mask (for Unique ID), indicates a device scan
%               (1-19 octet(s))
%   bit3: 0x04: Device Type, 0x01 Single-Antenna RET,
%               0x11 Multi-Antenna RET, 0x02 TMA (1 octet)
%   bit4: 0x05: 3GPP Release ID (1 octet)
%   bit5: 0x06: Vendor Code (2 octets)
%   bit6: 0x07: Reset device (0 octets)
%   XID twice, 1st: Get Unique ID and device scan. 2nd: Call with Unique ID
%   to get Device type.
ADDR = 'FF';
Ctrl = 'BF';
FI = '81';
GI = 'F0';
PV1 = data1;
PL1 = num2str(dec2hex(length(PV1)/2));
if length(PL1) < 2
    PL1 = strcat('0',PL1);
end
PV2 = data2;
PL2 = num2str(dec2hex(length(PV2)/2));
if length(PL2) < 2
    PL2 = strcat('0',PL2);
end
data = strcat(PI1,PL1,PV1,PI2,PL2,PV2);
GL = num2str(dec2hex(length(data)/2));
if length(GL) < 2
    GL = strcat('0',GL);
end
tocrc = strcat(ADDR,Ctrl,FI,GI,GL,data);
CRC = calculateCRC(tocrc);
exXID = strcat('7E',tocrc,CRC,'7E');
b = strsplit(addSpace(exXID));
end
```