FGI-GSRx Multi-GNSS Software Receiver

The FGI-GSRx software receiver has been extensively used as a research platform for the last one decade in different national and international Research and Development (R&D) projects to develop, test and validate novel receiver processing algorithms for robust, resilient and precise Position, Navigation and Timing (PNT). At present, the FGI-GSRx can process GNSS signals from multiple constellations, including GPS, Galileo, BeiDou, GLONASS, and NavIC. The software receiver is intended to process raw Intermediate Frequency (IF) signals in post-processing. The processing chain of the software receiver consists of GNSS signal acquisition, code and carrier tracking, decoding the navigation message, pseudorange estimation, and Position, Velocity, and Timing (PVT) estimation. The software architecture is built in such a way that any new algorithm can be developed and tested at any stage in the receiver processing chain without requiring significant changes to the original codes.

Multi-Constellation FGI-GSRx software receiver offers diversity and improved accuracy as it evolves from a GPS-only receiver to a more extensive receiver with capabilities from multiple global/regional constellations like Galileo, GLONASS, BeiDou, and NavIC. The FGI-GSRx software receiver was released as open source for the whole GNSS community to utilize it. The FGI-GSRx is also used as a fundamental software receiver tool in the 'GNSS Software Receivers' book published by Cambridge University Press, a next edition of one of the fundamental GNSS receiver textbooks [1].

Download and Installation

You need a current version of MATLAB® in order to run the FGI-GSRx. The FGI-GSRx was developed using MATLAB® R2016b. Thus, any newer version should also be compatible. If not already installed, please download and install MATLAB® using the installation guide from MathWorks website. The MATLAB® Communications toolbox is required in order to use FGI-GSRx.

You can download the FGI-GSRx MATLAB® source codes from following link:
https://github.com/nlsfi/FGI-GSRx.

The most updated version as of March 07, 2024, is v2.0.0. There is a corresponding release note with this new version where the changes to the previous v1.0.0 is highlighted.

Some example data files (raw IQ data files and processed MATLAB® data files) can be downloaded from this link: https://tiedostopalvelu.maanmittauslaitos.fi/tp/julkinen/lataus/tuotteet/FGI-GSRx-OS-DATAFILES. FGI also shared a new spoofing data repository named as 'FGI-SpoofRepo', which can be downloaded from the following link: https://doi.org/10.23729/f188197f-e6f1-4bf6-a870-62f8e81d5c50.

You may wish to follow this web page for FGI-GSRx related information:
https://www.maanmittauslaitos.fi/en/fgi-gsrx-os

Detail documentation about FGI-GSRx implementation can be found in the book [1], which is available for purchasing from Cambridge University Press.

After you have downloaded the source code directory to your directory of choice, you also need to download "Raw IQ Data" folder or already processed MATLAB® files under the folder named 'Example MATLAB® Data Files' folder. Now you can open MATLAB® and navigate to the said directory. At first you should add the main folder and all associated subfolders (i.e., the directory that contains source codes) to MATLAB® path by right clicking the correct folder and pressing "Add to Path → Selected Folder and Subfolders".

Execution

This section presents an overview on how to execute the FGI-GSRx. To run the receiver with default configurations use the MATLAB® editor to navigate to "/FGI-GSRx/param/ defaultReceiverConfiguration.txt" and open it. In this file you will find multiple path information given for example, in line 42 or in line 84. The files can be already processed MATLAB® data files to load (in case of 'sys,loadDataFile,true,') or the raw Inphase/Quadrature (I/Q) GNSS data files (in case of 'sys,loadDataFile,false,') that the user would like to execute for satellite acquisition, tracking and then to compute PVT.

In order for the program to find these files, please check that you have offered the right path and the right IQ file name in the configuration file. Once you have changed the path name of the file containing the IQ or already processed *.mat data, you should be able to run the software receiver.

Running the receiver in default configuration mode can be achieved by simply navigating to "/MATLAB/FGI-GSRx/main" and calling "gsrx('*name.txt*')" whereas *name.txt* can be the file the user just created. The user has to also specify the path where 'name.txt' is located, if the created '*name.txt*' configuration file is outside the directory of FGI-GSRx.

Running the receiver in non-default configuration is done in a similar fashion as in default parameters. In order to create a configuration file, familiarize yourself with the receiver configuration parameters. Only the parameters you want to change need to be changed in the personalized configuration file.

Data Management

The receiver architecture has been designed so that the intermediate data from acquisition and tracking can be saved in *.mat file, and processing can start from any pre-saved data file. A set of user parameters for the multi-GNSS receiver processing are read from a text file. The parameters specify the configurations of the IF data which include sampling frequency, data type, and bandwidth for further signal processing. If requested by the user, the acquisition is executed using the IF data stored in the file system, and the results are stored to the memory on the computer. Optionally, if the acquisition has been carried out before and the already stored acquisition data can be retrieved from the file system, the user can choose to bypass the acquisition process by setting appropriate parameter in the user parameter file. The result is the same regardless of which approach the user takes. The acquisition output is passed on to the tracking stage. The same options are available for tracking. If tracking has been carried out and the already stored tracking data can be retrieved from the file system, the user can choose to bypass the tracking. The result from tracking is then passed on to position computation. Figure 1 shows the functional blocks of the multi-GNSS receiver highlighting the feature that allows the user to skip acquisition and tracking and use pre-stored results. In any case, the pre-stored data are in *.mat file format, and the software receiver searches for the availability of relevant acquisition and tracking variables in the saved *.mat file in case of 'sys,loadDataFile,true,'.

Parameter System

The flexibility of modifying different receiver parameters is one of the main advantages of a software defined implementation. In FGI-GSRx, the parameter system is based on a configuration text file. All default values are kept in a default parameter file, and each user can have their own parameter file depending on their specific needs. Therefore, changing parameters do not really require any detailed changes to the actual underlying code. This flexibility makes it very easy to test different algorithms with many different data sets and in a way that can be easily reproduced later. The multi-GNSS receiver is therefore a very useful platform for researchers, as it enables them to independently develop and to test new algorithms in each stage of the receiver chain from raw IF samples to PVT solution. The user can configure different parameters related to the acquisition, tracking and position calculation modules, as well as parameters specifying the front end configuration used to store the

IF data. Given the possibility to set center frequency, sampling frequency, sample size, data type and bandwidth, the software receiver can easily be used with different front ends just by changing IQ data configuration parameters. Furthermore, it is possible for the user to enable/disable certain GNSS signals and to specify the number of milliseconds to process in the data file.

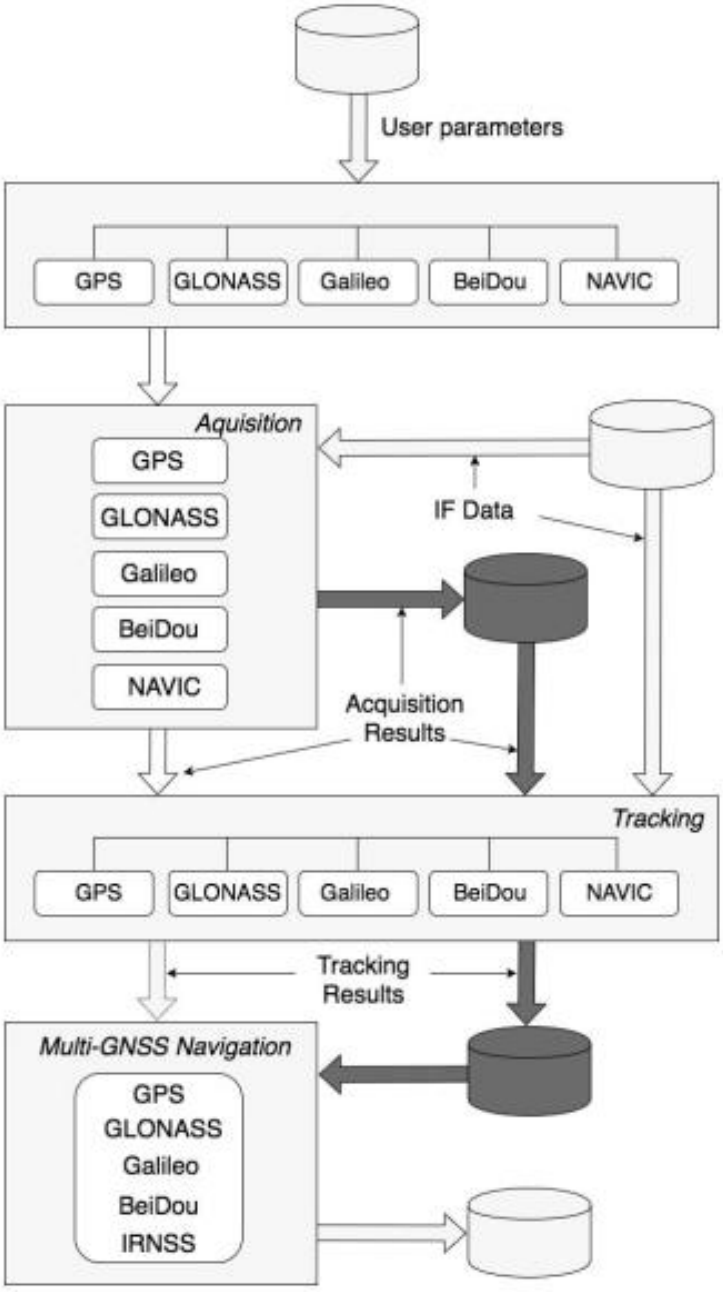

Figure 1: Functional blocks in the multi-GNSS FGI-GSRx receiver. The dark gray parts indicate the option to use pre-stored output from acquisition and tracking.

Folder Structure

The following folders are included in the FGI-GSRx:

- Acquisition (acq) folder contains all scripts related to the acquisition of all signals.
- Correlation (corr) contains all scripts related to signal correlation.
- Frame decoding (frame) contains all scripts related to data frame decoding.

- Geo (geo) folder contains all the scripts related to coordinate transformation.
- Ionex (ionex) contains all the scripts related to the handling of ionex files.
- Ionopsheric (iono) folder contains all the scripts related to ionospheric models.
- Least Square Estimation (lse) contains all the scripts related to the least square navigation filter.
- Main (main) folder contains the main execution scripts
- Modulation (mod) folder contains all the scripts related to modulation of the signals.
- Navigation (nav) folder contains all the scripts related to navigation.
- Observation (obs) folder contains all the scripts related to observation handling.
- Parameter (param) folder contains all the scripts related to the parameter system.
- Plot (plot) folder contains all the scripts related to plotting of data.
- Radio front-end (rf) folder contains subfolders for all supported radio frontends. Each sub folder
- contains configuration files for the various supported front ends.
- Satellite (sat) folder contains all the scripts related to satellite orbit calculations.
- Statistics (stats) folder contains all the scripts related to statistical analysis of the outputs.
- Time (time) folder contains all the functions related to time conversion.
- Tracking (track) folder contains all the scripts related to tracking signals.
- Troposphere (tropo) folder contains all the scripts related to the tropospheric models.
- User Interface (ui) folder contains all the scripts related to the output of data to command line.
- Utility (utils) folder contains utility script needed in various parts of the receiver.

General Operation

The main function is called 'gsrx.m'. It calls some necessary functions mentioned in Table 1 to carry out the GNSS receiver functionalities. The user must run the main function 'gsrx.m' with a desired parameter file given as string input to the function like gsrx('MyReceiverConfigurationParameters.txt'). An example parameter file is given with the software. The parameter input format must be followed for proper functioning of the FGI-GSRx multi-GNSS receiver.

Table 1: List of necessary functions called from the main script.

Function name	Tasks performed	Example code
gsrx()	The main function to carry our multi-GNSS receiver processing	gsrx('default.txt')
readSettings()	Read user-defined parameters files	settings = readSettings(varargin1);
generateSpectra()	Generate spectra and bin distributions	generateSpectra(settings);
doAcquisition()	Acquire signals	acqData = doAcquisition(settings);
plotAcquisition()	Plot the acquisition results	plotAcquisition(acqData.gale1b,settings, 'gale1b'); %Example showed only with Galileo E1B signal, 'gale1b'
doTracking()	Track signals	trackData = doTracking(acqData, settings);
plotTracking()	Plot the tracking results	plotTracking((trackData, settings);
generateObservations()	Generate observations	obsData = generateObservations(trackData, settings);
doFrameDecoding()	Decode navigation data from the ephemeris	[obsData, ephData] = doFrameDecoding(obsData, trackData, settings);
doNavigation()	Perform navigation on the processed observables	[[obsData,satData,navData] = doNavigation(obsData, settings, ephData);
calcStatistics()	Calculate receiver statistics	statResults = calcStatistics(navResults);

Acquisition

The acquisition block searches and acquires satellite signals one signal at a time. After the search is completed for one GNSS signal acquisition, it will start for the next enabled signal. After all signals have been searched for, the results are handed over to tracking and then finally to navigation. Plots for all acquired signals are generated and a combined histogram plot presents searched and found signals for each constellation. Table 2 presents an example of acquisition parameter configuration for Galileo E1B signal.

Table 2: Example of acquisition parameters.

File, parameter name, value	Explanation
gale1b,acqSatelliteList,[1:30],	Specify what GPS satellites to search for [PRN numbers]
gale1b,nonCohIntNumber,2,	Number of non-coherent integration rounds for signal acquisition
gale1b,cohIntNumber,1,	Coherent integration time for signal acquisition [ms]
gale1b,acqThreshold,9,	Threshold for the signal presence decision rule
gale1b,maxSearchFreq,6000,	Maximum search frequency in one direction
gale1b,modType,'CBOC',	Can take input either of the two modulation types: 'CBOC' or 'SinBOC'

The acquisition block is executed via a function call, shown below:

```
acqData = doAcquisition(settings);
```

All acquisition specific functions are listed in Table 3. The acquisition data is stored in a structure called 'acqData'. There is one structure for each found signal (e.g. acqResults.gpsl1, acqData.gale1b, etc.). Each structure has information concerning the signal acquired (e.g. nrObs, duration, signal, channel, etc.) as well as statistics from the acquisition phase (e.g. peakMetric, peakValue, variance, baseline, SvId: [1x1 struct], codePhase, carrFreq, etc.) for each search satellite of that particular GNSS signal.

Table 3: List of acquisition functions.

Function name	Tasks performed	Example code
doAcquisition()	Attempts to acquire signal for each enabled GNSS constellation (i.e., 'gpsl1' or 'gale1b', etc.)	acqData = doAcquisition(settings);
initAcquisition()	This function initializes the acquisition structure in a favorable way	acqResults = initAcquisition(allSettings);
getDataForAcquisition()	Reads 100 milliseconds (ms) of data for acquisition: the FGI-GSRx receiver does not attempt any reacquisition. Therefore, this is the only data that is used for acquisition.	[pRfData,sampleCount] = getDataForAcquisition(signalSettings, msToSkip, 100);
acquireSignal()	Generates codes, performs the modulation and up- sampling before doing the FFT-based correlation. Use FFT-based acquisition technique for acquiring the satellites from the enabled constellation	acqResults.(signal) = acquireSignal(pRfData, signalSettings);
searchFreqCodePhase()	This function performs the parallel code phase search acquisition for a given satellite signal	results = searchFreqCodePhase(upSampledCode, signalSettings, pRfData, PRN);
plotAcquisition()	Plot the acquisition results	plotAcquisition(acqData.gale1b, settings, 'gale1b'); %Example showed only with Galileo E1B signal, 'gale1b'

Tracking

The tracking block uses the output from the acquisition and correlates the incoming signal with signal replicas to produce the raw measurements from all signals and satellites. The processing is done for all found signals in parallel and the results are sent to navigation. All tracking specific functions are listed in Table 4. The tracking block is executed via a function call, shown below:

```
trackData = doTracking(acqData, settings);
```

Table 4: List of tracking functions.

Function name	Tasks performed	Example code
doTracking()	Main tracking function which in turn calls other associated tracking functions in order to carry out the code tracking and the carrier tracking successfully	trackData = doTracking(acqData, settings);
initTracking()	Initializes tracking channels from acquisition data	trackResults = initTracking(acqResults, allSettings);
GNSSCorrelation()	Performs code and carrier correlation	trackResults.(signal) = GNSSCorrelation(trackResults.(signal), channelNr);
GNSSTracking()	Performs state-based tracking for the received signal	trackResults.(signal) = GNSSTracking(trackResults.(signal), channelNr);
showTrackStatus()	Prints the status of all track channels to the command window	showTrackStatus(trackResults, allSettings, loopCnt);
plotTracking()	Plots all tracking related results for each tracked channel. Plots also the C/N0 of all satellites in each constellation	plotTracking(trackData, settings);

FGI-GSRx-v2.0.0 offers execution of tracking in two different modes: i) Sequential and ii) Parallel. Sequential tracking mode is the default option, where all the satellites tracking is carried out sequentially one after another until the user specified duration within one MATLAB® instance. In contrast, in case of parallel tracking mode, the software receiver executes one instance of MATLAB® for tracking each individual satellite. To facilitate this operation, following functions are added in the new release. A detail user instruction for running the software receiver in parallel tracking mode is explained at the end of this user manual.

Table 5: List of parallel tracking functions.

Function name	Tasks performed	Example code
doTrackingParallel()	This function creates a batch script '*.bat' file, that can be then run from the command prompt. The batch file name and path will be provided by the user, in case parallel tracking option is used.	doTrackingParallel(trackDataFileName,settings);
doTrackingSingleChannel()	This function is designed to run for each single tracking channel. At the end, it saves the tracking channel results in the *.mat file, that will then later be combined with other tracking channels of the same signal.	doTrackingSingleChannel(acqData,trackResults, allSettings);
initializeAndSplitTrackingPerChannel()	In case of parallel tracking mode, this function is used. It initializes the tracking channels from the acquisition, and then split the channel so that only 1 channel will be processed by one MATLAB® instance.	trackDataFileName = initializeAndSplitTrackingPerChannel(acqData, settings);
combineSingleTrackChannelData()	In case of parallel tracking option, the user can use this function to combine different signals and tracking channels *.mat data files into one.	trackData = combineSingleTrackChannelData(settings);
showTrackStatusSingle()	Single satellite tracking status is displayed with this function. It is functionally like showTrackStatus.m function.	showTrackStatusSingle(trackResults,allSettings,loopCnt);

Data Decoding and Navigation

The data decoding block uses the output from the tracking and convert the processed tracking data to ephemeris for different available GNSS constellations. It finds the preambles first and then starts decoding the ephemeris for that constellation.
The navigation block has three main tasks: to convert the measurements to observations, to calculate the satellite positions and velocities, and finally to calculate the position of the user. Each element of the obsData, satData and navData data structure contains data for one signal. The settings structure contains the parameters for each enabled signal. Navigation functions are listed in Table 5.

Table 5: List of navigation functions.

Function name	Tasks performed	Example code
doNavigation()	This is the main loop for navigation. It utilizes receiver observables, satellite specific information, correction information, and offers a navigation solution based on the user configuration settings	[obsData,satData,navData] = doNavigation(obsData, settings, ephData);
initNavigation()	This function calculates some initial values needed for the main navigation loop	[obsData, nrOfEpochs, startSampleCount, navData, samplesPerMs] = initNavigation(obsData, allSettings);
applyObservationCorrections()	This is the main function to apply all available corrections to the specific observable	obsData = applyObservationCorrections(allSettings, obsData, satData, navData, corrInputData);
checkObservations()	This function decides which current observations are going to be used in the navigation computation	obsData = checkObservations(obsData, satData, allSettings, navData);
getNavSolution()	Calculates Least Square Estimation (LSE) based navigation solutions for the measured receiver observables	[obsData, satData, navData] = getNavSolution(obsData, satData, navData, allSettings);
showNavStatus()	Prints the status of navigation to the command window	showNavStatus(allSettings, currMeasNr, navData, obsData, satData);

References

[1] Borre K, Fernández-Hernández I, López-Salcedo JA, Bhuiyan MZH, eds. GNSS Software Receivers. Cambridge University Press; 2022.

User Instruction to run Parallel Tracking Mode in FGI-GSRx-v2.0.0

The parallel processing is done only at the tracking stage where one MATLAB® instance is dedicated for each allocated satellite tracking. The three important steps for using FGI-GSRx-v2.0.0 for this purpose are explained below:

1. Splitting the data and preparing the respective data files for individual satellites and generating a *.bat shell file.
2. Running the batch (Windows) or shell (Linux) command to initiate parallel MATLAB® instances for tracking each satellite individually.
3. Combining the processed *.mat data into a single file to be used for processing for NAV solution.

The parameter file settings and procedure to execute each step is explained below:

Step 1:

In the user parameter file, use the following settings for parallel tracking mode:

1. Make sure Input/Output file name categories are set to false.

   ```
   % Input file names
   sys,loadDataFile, false,     % Defines if data file is to be loaded
   sys,dataFileIn,"",
   % Output file names
   sys,saveDataFile, false,     % Defines if data should be stored to file
   sys,dataFileOut,"",
   ```

2. Process raw IQ data via parallel implementation following the settings.
   a. 'parallelChannelTracking' should be set to true. Initially, the tracking block would look for any files in the 'trackDataFilePath' folder.
      I. If the folder is empty, FGI-GSRx-v2.0.0 would generate the initial files corresponding to all the satellites acquired in the same folder.
      II. If the folder is not empty, it will combine (more details are explained in Step 3) each single satellite's 'trackData' file into one combined big file for navigation processing.
   b. Specifiy the current working directory of FGI-GSRx v2.0.0 in the 'currentWorkingDirectoryForFGIGSRx' command.
   c. Specifiy the directory where the parallel processing files are to be stored in 'trackDataFilePath'. It is recommended to make a separate folder for these files.
   d. Specify the name of the batch/shell file to be utilized in STEP 2.
   e. Specify MATLAB® path in 'matlabpath'. This is optional and is only required in case MATLAB® is not defined as a system variable in the general Windows settings.

   ```
   %Tracking option: parallel vs sequential
   sys,parallelChannelTracking,true,
   %Specifiy the current working directory of FGI-GSRx-v2.0.0
   sys,currentWorkingDirectoryForFGIGSRx,'D:\FGI-GSRx-v2.0.0\',
   %Specifiy the directory for each tracking data files for each satellite to be saved
   sys,trackDataFilePath,'D:\ TG-DFMC\Parallel Processing\',
   %FGI-GSRx v2.0.0 creates this batch file to be processed later in windows command
   %prompt
   sys,batchFileNameToRunParallelTracking,'runGNSSSingleSatelliteTracking.bat',
   %MATLAB® path to be provided in case MATLAB® is not defined as a system variable
   %in the general Windows settings
   sys,matlabpath,'C:\Program Files\MATLAB\R2023a\bin\matlab',
   ```

Step 2:

1. Check the folder specified in Step 1: 2c. It should contain initialized tracking data file for all the acquired satellites. An example of the files can be seen in Figure 2.

Figure 2: Example ‘trackData’ files for GPS L1 and Galileo E1 signals. They are initialized based on the acquisition results.

2. Run the batch file generated in Step 1: 2e.
    a. By default, this file is saved in in the ‘D:\FGI-GSRx-v2.0.0\main’ directory. The contents of the Windows batch file can be seen in Figure 3.

```
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_5.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_7.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_8.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_9.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_13.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_18.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_20.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_27.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gpsl1_Satellite_ID_30.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gale1b_Satellite_ID_3.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gale1b_Satellite_ID_5.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gale1b_Satellite_ID_13.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gale1b_Satellite_ID_24.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('C:\FGI GSRx - V2\')); load 'C:\resampledTG-SOFT\Parallel Processing\trackData_gale1b_Satellite_ID_31.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"
```

Figure 3: Example of an FGI-GSRx-v2.0.0 created Windows batch file.

    b. By default, Windows batch file is created which can be converted into Linux shell file (Please see Annex A).
    c. Executing this file would automatically initiate multiple MATLAB® instances for tracking each satellite separately, and then store the processed ‘trackData’ files.

This process intends to maximize CPU performance by running simultaneous sessions of MATLAB®. Therefore, it may slow down the computer performance for other tasks.

Step 3:

Once, the tracking is complete for all the satellites in multiple MATLAB® instances, the data would be stored in the folder specified in Step 1: 2c. The next step is to combine these files into one single file for further processing by FGI-GSRx-v2.0.0. For that purpose, following steps are to be done.

a. Make settings for ‘loadDataFile’ in the input file name section of configuration file true and specify the first processed satellite file path as the input file. It will be enough to offer any single channel's track data file, the receiver will automatically combine the necessary ‘trackData’ files into one combined big file for navigation processing, for example:

```
% Input file names
sys,loadDataFile,true,     % Defines if data file is to be loaded
sys,dataFileIn,"D:\TG-DFMC\Parallel Processing\trackData_gale1b_Satellite_ID_3.mat",
```

b. Make settings for ‘saveDataFile’ in the input file name section of configuration file true and specify the first processed satellite file path as the input file. For example:
% Output file names
sys,saveDataFile,true, % Defines if data should be stored to file
sys,dataFileOut,"D:\TG_DFMC\trackData_combined_GPS_L1_Galileo_E1.mat",

c. Parallel tracking settings should remain the same as was given in Step 1.

Step 4:

FGI-GSRx-v2.0.0 can be used now with combined tracking data file generated in Step 3 for further processing.

% Input file names
% Defines if data file is to be loaded
sys,loadDataFile,true,
sys,dataFileIn,"D:\TG_DFMC\trackData_combined_GPS_L1_Galileo_E1.mat",

Annex A:

Converting windows batch file to Linux shell file:

- Add ‘&’ after each command line.
- Add ‘wait’ at the end of the file.
    - The contents of the ‘*.sh’ file can be seen in Figure 4.

```
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_5.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_7.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_9.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_13.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_16.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_20.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_27.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_gps1l_Satellite_ID_30.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_3.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_5.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_13.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_15.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_24.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_25.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
matlab -nosplash -nodesktop -minimize -r "addpath(genpath('/home/boost-user/FGI GSRx - V2/')); load '/home/boost-user/ParallelProcessingtrackData_galeIb_Satellite_ID_31.mat'; doTrackingSingleChannel(acqData,trackResultsSingle,allSettings);"&
wait
```

Figure 4: Example after the conversion from FGI-GSRx-v2.0.0 created Windows batch file to a Linux shell file (‘*.sh’).

- Save the file in ‘.sh’ extension.
- Run the command window from the current folder where *.sh file is located and type the following:
    - Set execute permission on your script using ‘chmod’ command:
        - chmod +x script_name_here.sh e.g.,
          chmod +x runGNSSSingleSatelliteTracking.sh
    - Execute the shell script in Linux.
        - ./runGNSSSingleSatelliteTracking.sh