The Multi-Sensor Navigation Analysis Tool (MuSNAT) Architecture, LiDAR, GPU/CPU-GNSS-Signalprocessing

D. Dötterböck, H. Gomez-Martinez, M. Subhan Hammed, F. Hörkner, T. Kraus, D. Maier, T. Pany, D. Sanchez-Morales, A. Schütz, *Universität der Bundeswehr München, Neubiberg, Germany*

P. Klima, D. Ebert, *Axtesys GmbH, Graz, Austria*

Presentation Outline

- Motivation
- MuSNAT Overview
  - Processing Core / Analysis-Visualization
- Input Sources
  - IF samples (File, TCP/IP), LiDAR, IMU, Trajectory, RINEX
- LiDAR-Processing
  - Data-flow, algorithm, libraries, results
- Precision GNSS-Signal Processing
  - Code/phase tracking performance verification
  - GPU implementation for unlimited channels?
- Integration filter
  - GNSS/IMU RTK-LIB extension
- SQL-Logging
- Automated Test Framework

Motivation

State of the art for autonomous driving:
GNSS (PPP) + Inertial + LiDAR/Camera/Radar

Highly complex processing algorithms
- GNSS processing (**cycle-slips**, multipath, PPP)
- Inertial strapdown
- Visual odometry (LiDAR clustering, **matching**, …)
- Integration filter (**IMU parameters and ambiguity estimation**)
- **Integrity**

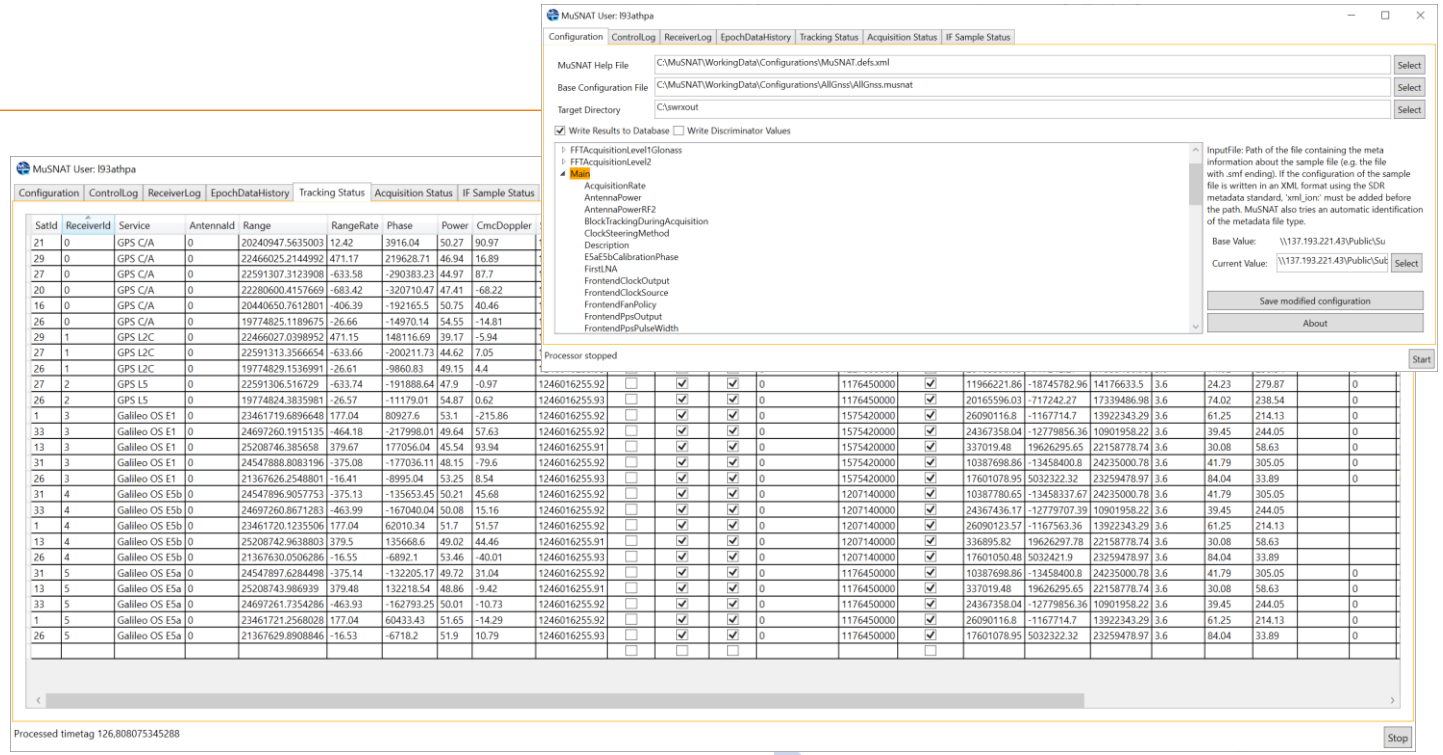
New boundary conditions
- New GNSS signals (multi-carrier, authentication, **higher frequencies**, …)
- New signal processing methods (meta-signal, Bayesian **direct position estimation**
- Antenna **array processing**
- GNSS/LTE/**5G** integration

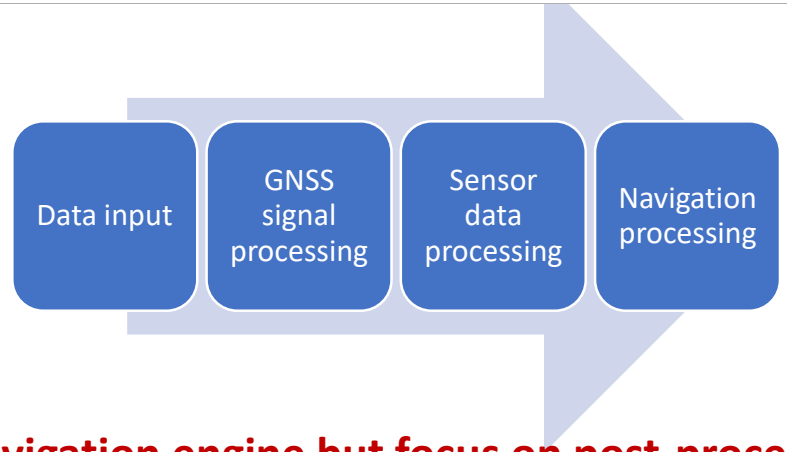
Analysis tool to get consistent data insight until the lowest possible level.

Debugging, debugging, debugging, …

MuSNAT-Core

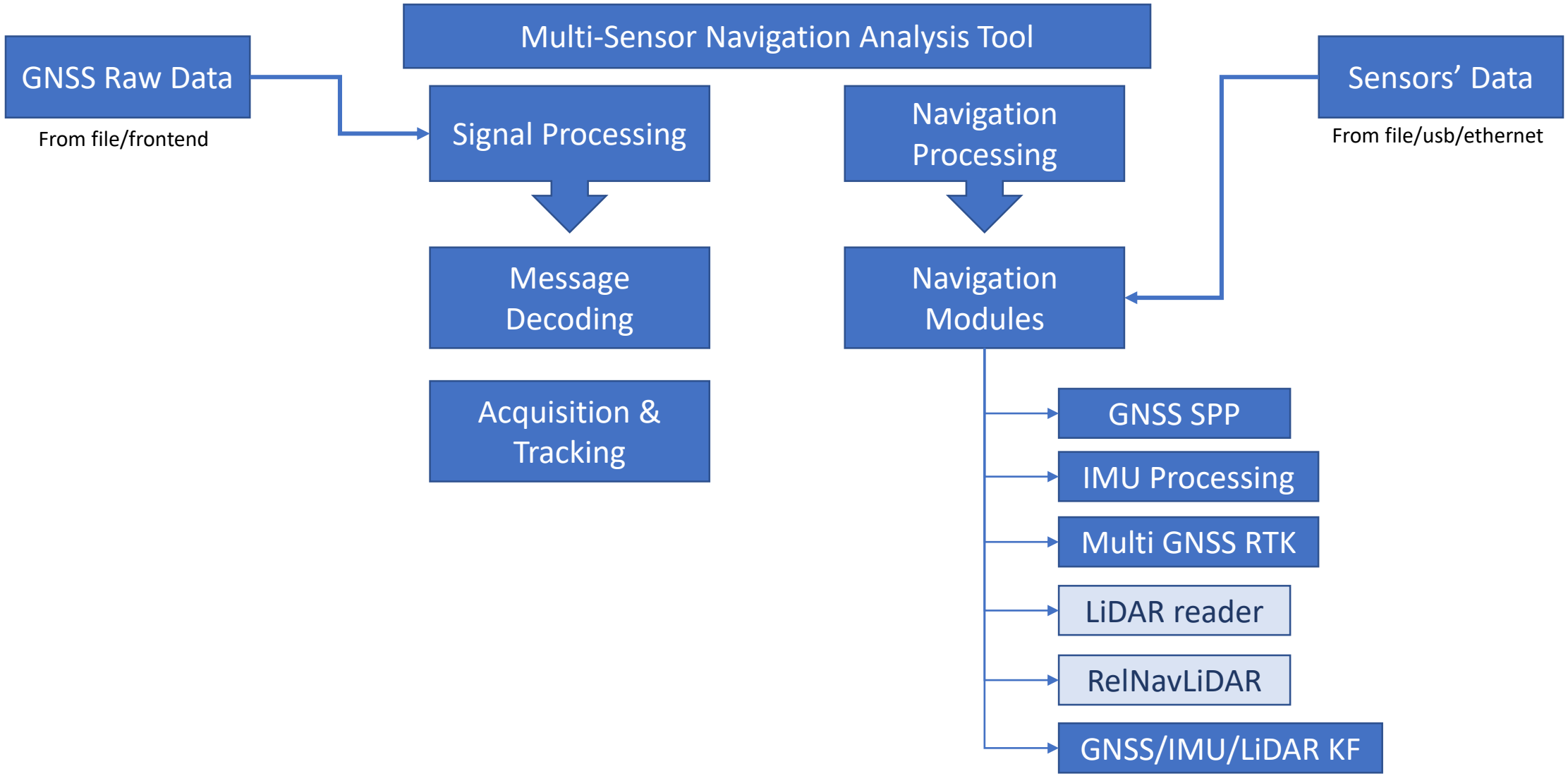
- Microsoft Visual Studio 2017
  - C++ for processing
  - C# for GUI and data management
- ipexSR GNSS software receiver
- Intel Performance Primitives (IPP)
  - signal processing CPU
- nVidia CUDA
  - signal processing GPU
- Point Cloud Library (PCL) 1.9
  - LiDAR processing
- RTK-LIB
  - RTK, PPP, GNSS/IMU-filter
- SQlite
  - Archiving processing results
- Eigen
  - Matrix operations
- Qt, OpenCV, …
  - Auxiliary functions





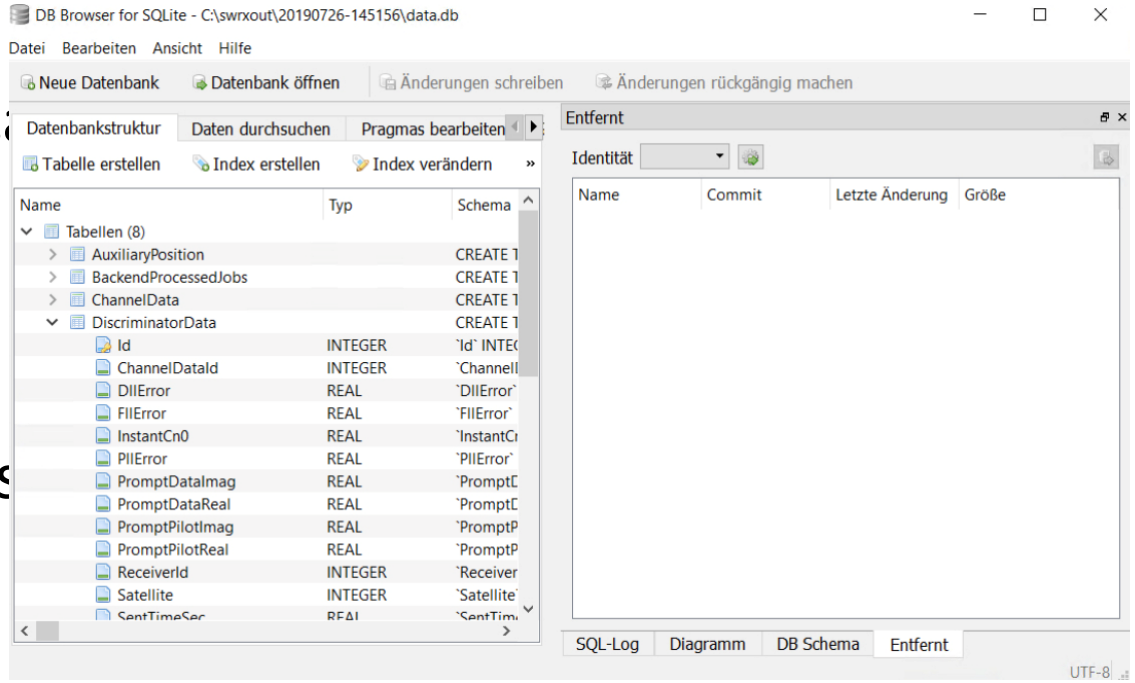
Real-time capable navigation engine but focus on post-processing

Data flow in MuSNAT



SQL-Logging

- All MuSNAT-Core processing results stored in a single SQL database
  - IF signal spectra, acquisition plots, tracking correlators, discriminators, multi-correlator, GNSS raw data (pseudoranges, …), … (see next slide)
  - To be read and visualized by MuSNAT-Analyzer or Matlab, python, …
- Data collection MuSNAT-Core
  - serialization of a multi-threaded process
- SQL database structure created automatic
  - Object-Relational-Mapper (ORM)
- Written via SQlite
  - open source framework
- Identified to be potentially time-critical tas



MuSNAT-Analyzer

- **Time synchronized analysis**

- Raw Data
  - GNSS samples
  - Sensor data
- Signal processing
  - Correlator, discriminator values
  - Acquisition plots, multi-correlator
- RINEX-level data
- Navigation processing
  - Position, velocities, accuracy, residuals
  - Ambiguity ratio, RTK/PPP metrics
  - GNSS/IMU biases, attitude, …
  - LiDAR TBD

Data Sources

- GNSS/LTE/5G samples
  - **File input**
    - Internal reader
    - Support of ION SDR sample standard (V0.4)
  - **TCP/IP input (real-time)**
    - Interface to NI USRP
    - Generic real-time interface to other frontends
- Sensor data
  - **LiDAR**
    - TCP/IP (real-time), file input (PCL based)
  - **IMU**
    - File based, real-time (USB?) in development

National Instruments NI-2955

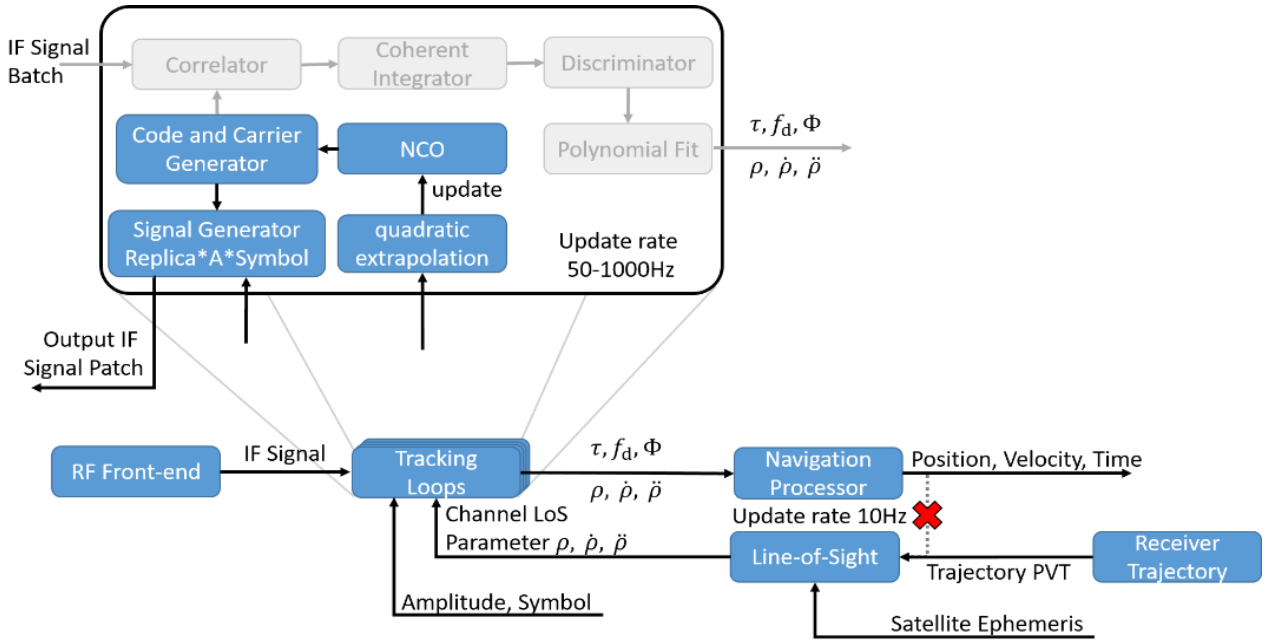
- Four RF channels
  - 0.01 – 6 GHz, BW < 80 MHz
  - All GNSS, C-Band, LTE/5G
- PCIe (x4) (832 MByte/s)
- Sensor input port
  - programmable
- Clock I/O
- On-board FPGA
  - Gain control
  - Interference mitigation
  - Decimation, packaging, buffering

- **First target version (LabView)**
  - **2 channels @ 20 MHz, 16-bit I/Q**
- **Final version (LabView+FPGA)**
  - **4 channels @ 80 MHz, 2-4 bit I/Q**

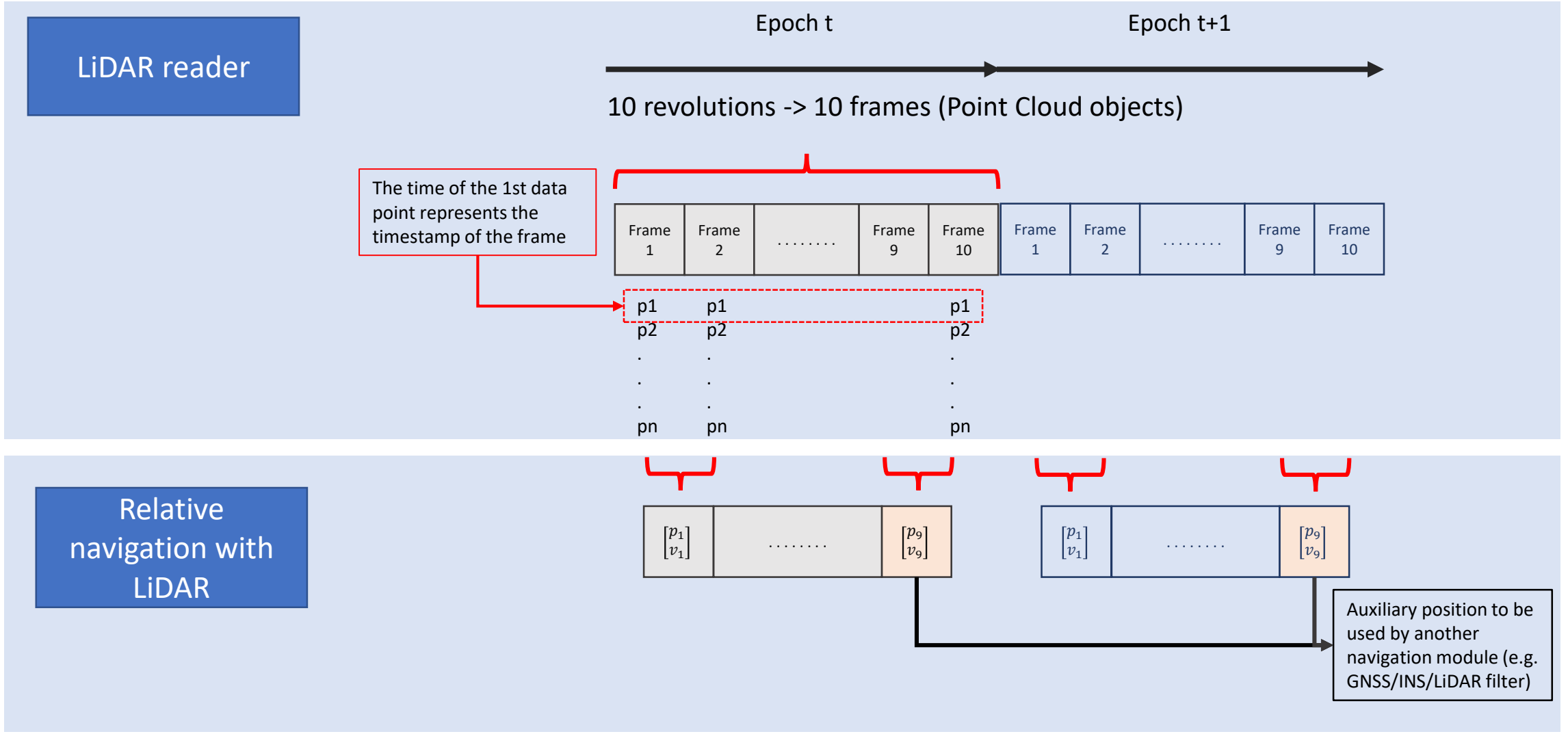
Non-Standard Input Sources

- **Direct input of RINEX files (instead of IF samples)**
  - Bypass GNSS-signal processing, time-saving
  - Analysis of commercial receivers, Android raw data, …

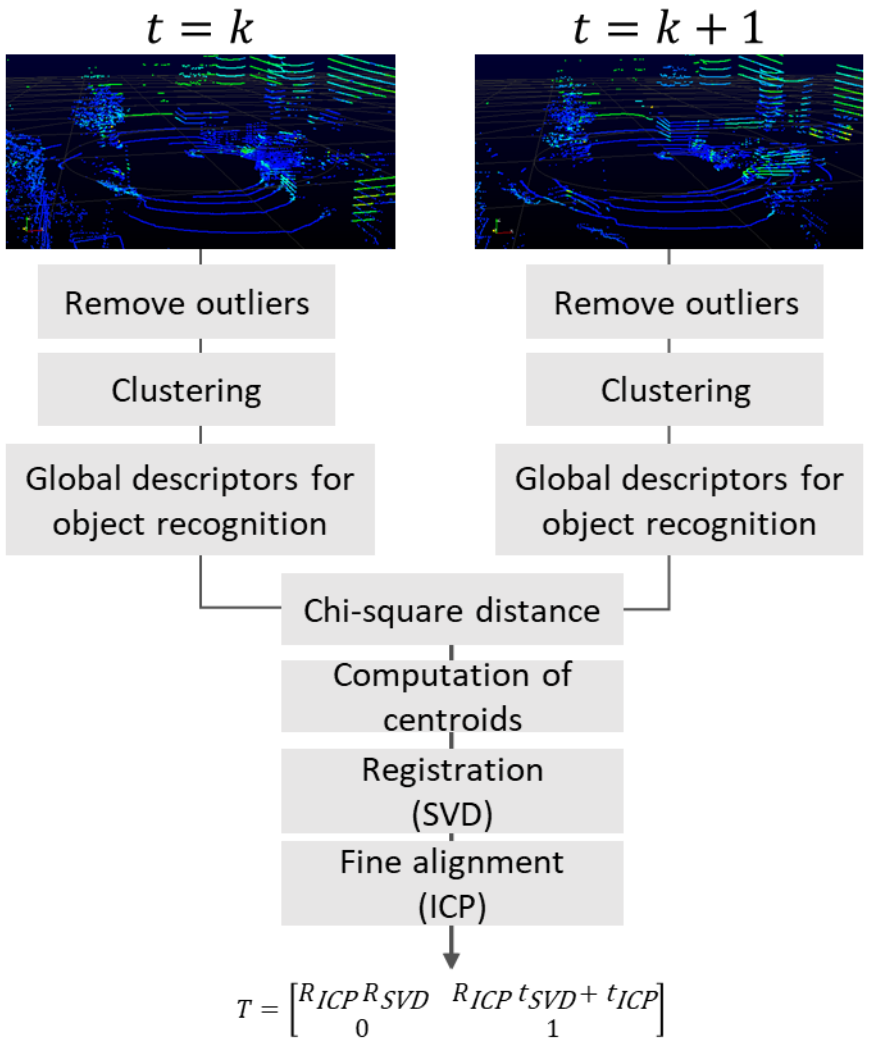
- **Trajectory input**
  - Used if software is run a GNSS signal generator
  - Transceiver concept

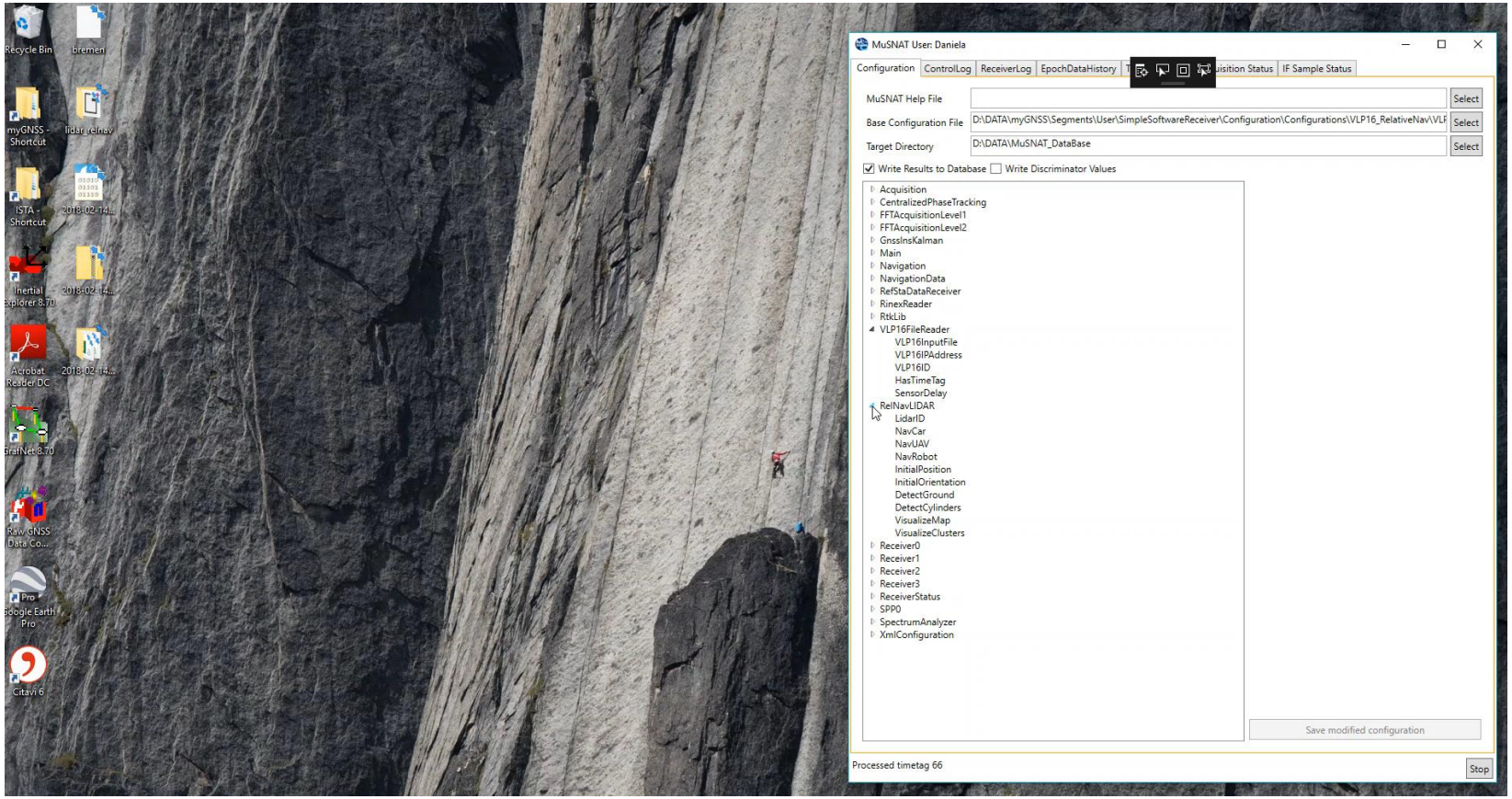


LiDAR – Data flow in MuSNAT



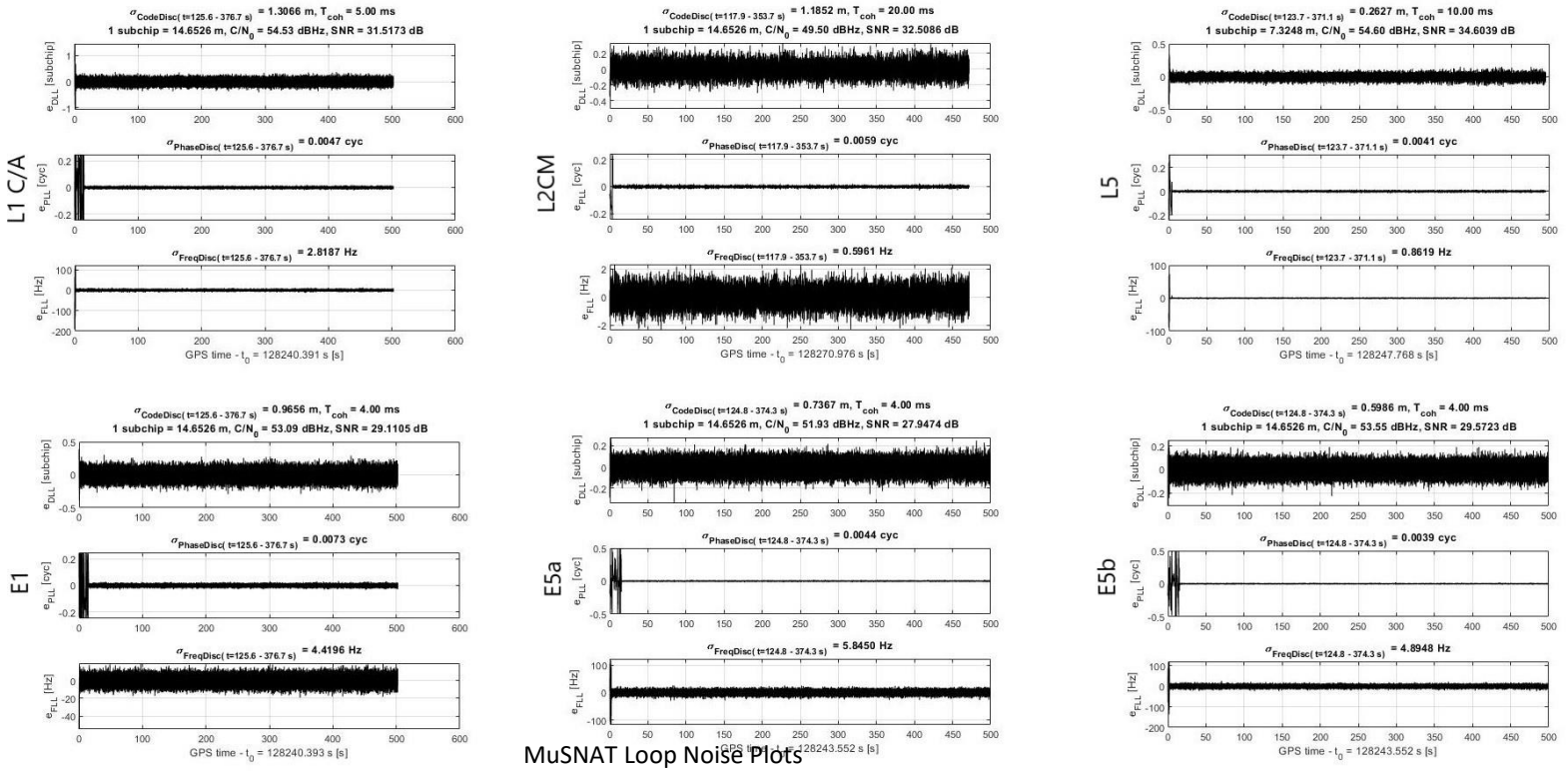
LiDAR – first results





GNSS – Signal Tracking - CPU

- Correlator based tracking
- Different modes
  - **Precise**: text book like
  - **Fast**: numerical approx.
- Verification methodology
- GPS/Gal. L1/L5/E1/E5a/E5b
- Code discriminator noise compared against semi-analytic computations



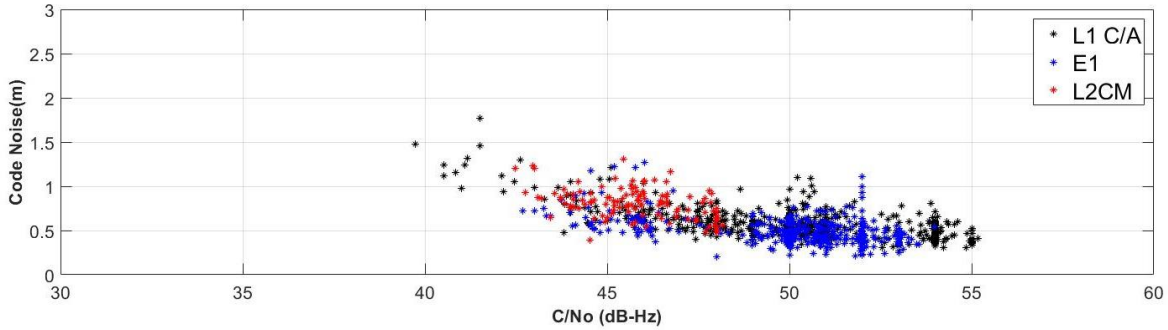
MuSNAT Loop Noise Plots

Precise mode

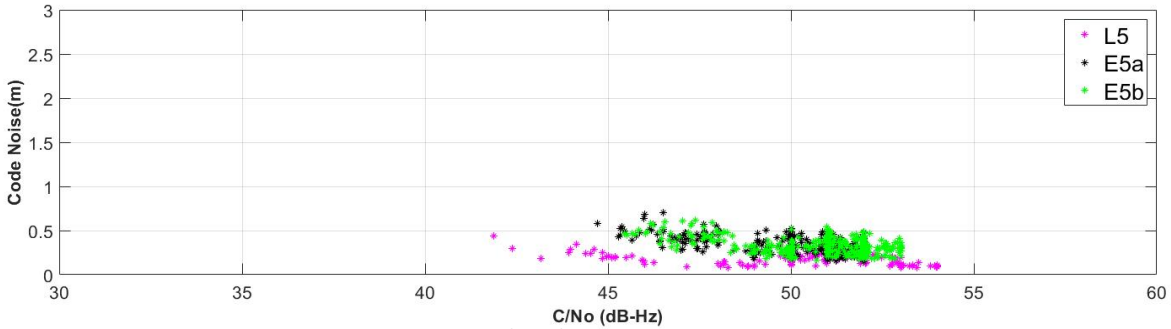
Signal	Coherent Int Time (ms)	SNR (dB)	Code Disc. Noise Measured (m)	Code Disc. Noise Analytic (m)	Percentage Deviation (%)
L1C/A	5	31.52	1.3066	1.2921	1.12
L2C	20	32.51	1.1852	1.1621	1.99
L5	10	34.6	0.2627	0.2762	4.89
E1	4	29.11	0.9656	0.9743	0.89
E5a	4	27.94	0.7367	0.6027	22.23
E5b	4	29.57	0.5986	0.4959	20.71

GNSS – Signal Tracking - CPU

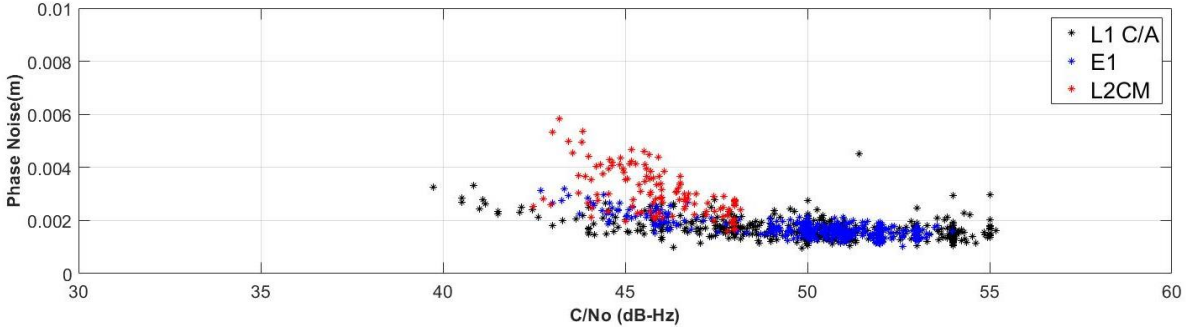
Code/Phase noise vs C/No



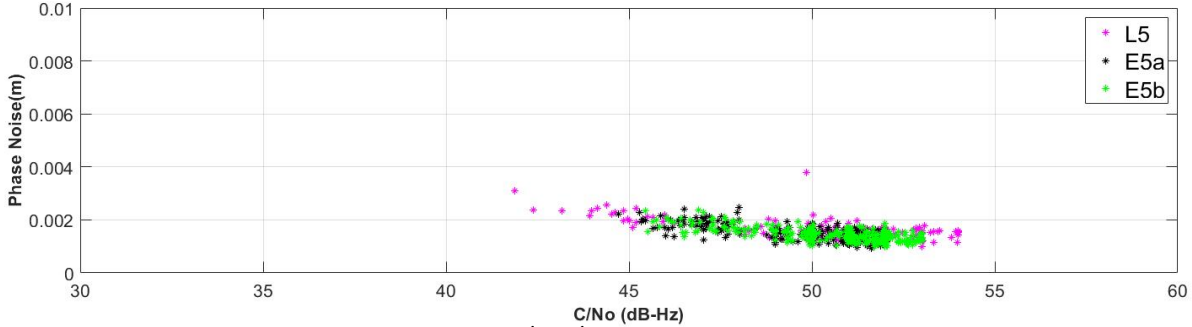
L1/E1/L2CM Code Noise



L5/E5a/E5b Code Noise



L1/E1/L2CM Phase Noise



L5/E5a/E5b Phase Noise

Precise mode

Verification of MuSNAT in a zero-baseline setup with commercial receiver

GPU – Signal Processing Motivation

- Why use graphics cards for GNSS signal processing?

- Ease of implementation
  - Use of high-level language
  - R&D cost efficient (compared to FPGA)
- No implementation loss
  - Precise mode
  - 16-bit signal processing
  - No need for look-up-tables or other approximations
- Promise for a virtually unlimited number of channels?

23.5 TFLOP (16-bit)

GPU – Replica Generation Sample Code

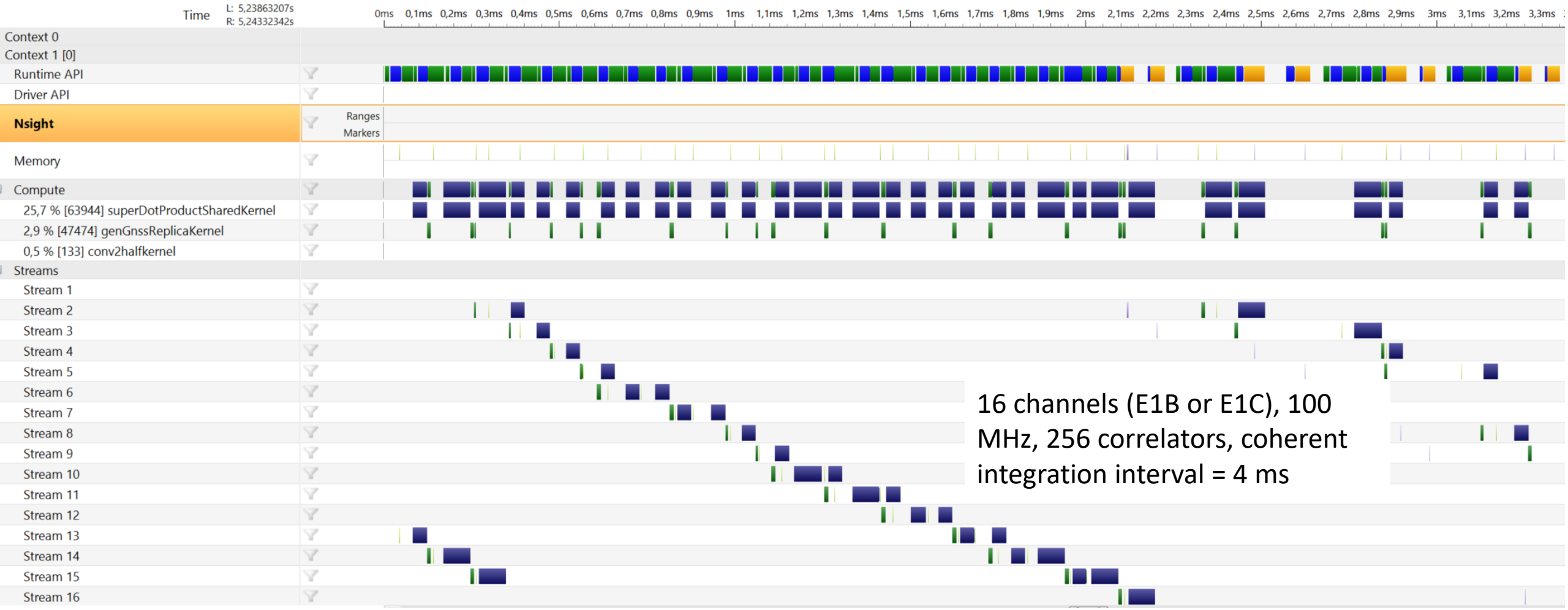
Code replica: generic waveform (BPSK, CBOC, TMBOC, …)
Carrier replica

```
        // CUDA loop over all samples
        for (int k = index; k < N; k += stride)
        {
                // code replica
                fCodePhase = fIniCodePhase + k * fDeltaCodePhase;
                nSubchipIdx = floorf(fCodePhase);
                hSubChipValue = __half2half2(baseband[nSubchipIdx]);

                // carrier replica (cos in .x, sin in .y)
                fCarrPhase = fIniPhase + k * fDeltaPhase;
                fCarrPhase = (fCarrPhase - floorf(fCarrPhase))*f2Pi; // phase computation done with 32-bit
float
                hCarrier = h2sin(    // carrier NCO
                                                        __hadd2(
                                                                __half2half2(
                                                                        __float2half(fCarrPhase) //
convert 32-bit phase to 16-bit
                                                                )
                                                        , hCarrOffset ) // apply pi/2 offset for cosine
                                                );

                // product
                replica[k] = __hmul2(hSubChipValue, hCarrier );
        }
```

GPU Numerical Performance Analysis



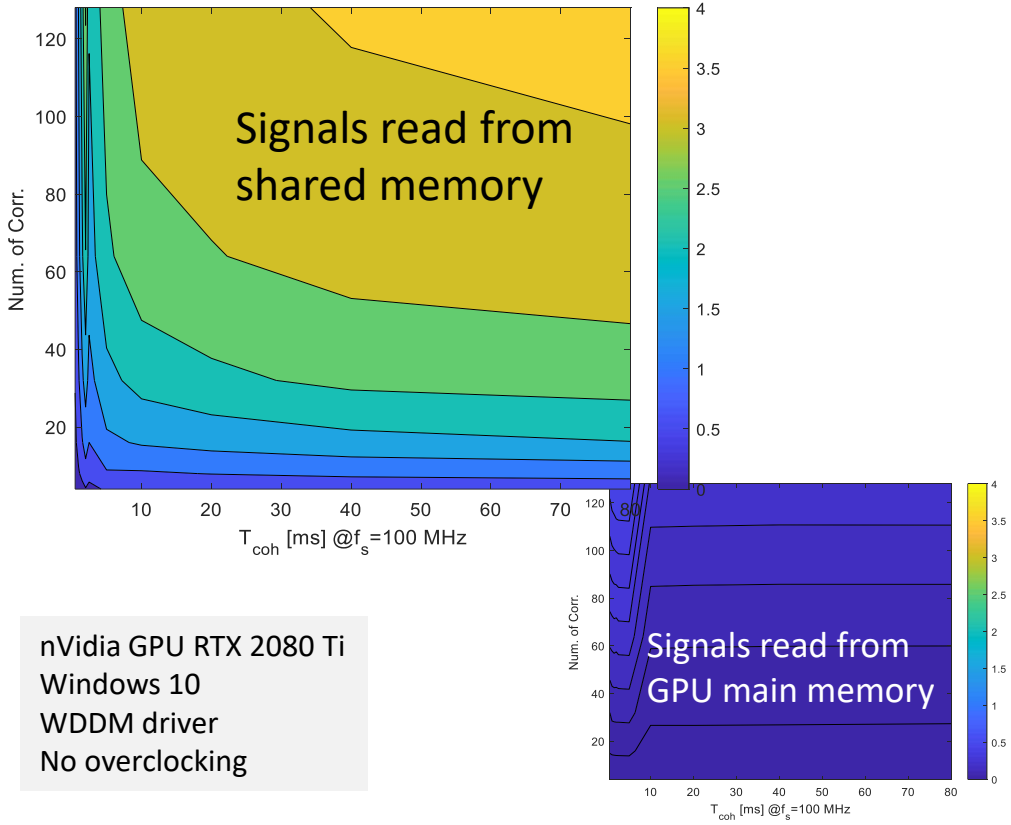
16 channels (E1B or E1C), 100 MHz, 256 correlators, coherent integration interval = 4 ms

GPU correlator performance

- Replica generation
  - 4 ms replica, 100 MHz
    - Divided into 2 batches
  - ~16 µs measured from slide before
  - Generation rate

$$f_g = 100\ MHz \frac{4ms}{16\mu s} = 25\ \text{Gsamples/s}$$

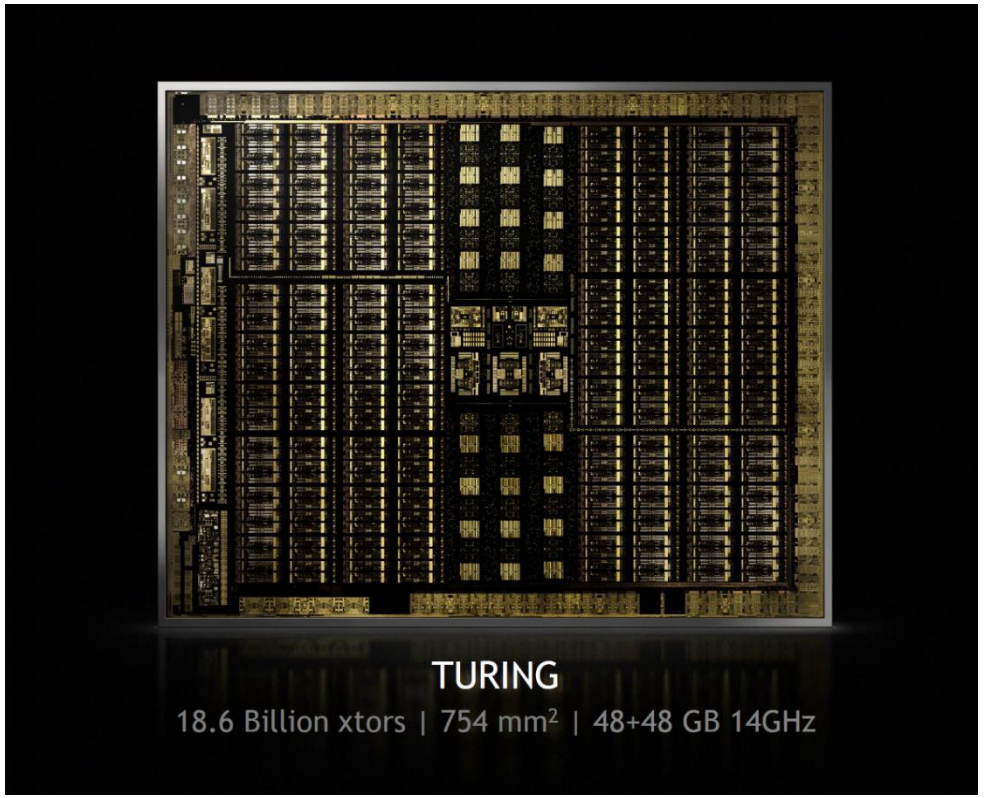
- Dot-Product (=Correlation)
  - Computational performance TFLOP, 16-bit, multiy-and-add



nVidia GPU RTX 2080 Ti
Windows 10
WDDM driver
No overclocking

GPU Architectural Constraints

- Theoretical limit $\#N$ of number of channels
- Replica generation limit
  - $\#N < \frac{f_g}{f_s} = 250$
- Correlation limits
  - Theoretical compute limit: $\#N < \frac{F}{4n_c f_s} = 2031$
  - Compute +memory limit:
    - $\#N < \frac{F_{eff}}{4n_c f_s} = 78$
- Evocation (launch) limit:
  - $\#N < \frac{T_{coh}}{kT_{ker}} = 38 \xrightarrow[k=7/24=0.3]{\text{Combined launch}} \#N < \frac{T_{coh}}{kT_{ker}} = 888$

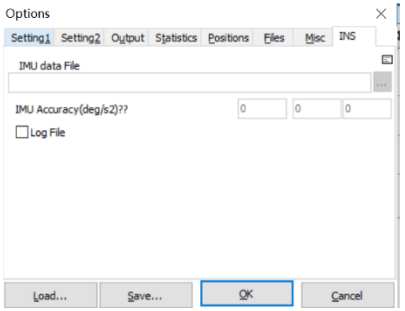


Assumptions:
sample rate $\boldsymbol{f_s}$ **=100 MHz**, $\boldsymbol{n_c}$ $=$ **32 correlators**, F =26 TFLOP, F_{eff} =1 TFLOP, k $=$ 7 GPU kernel launches per interval and channel (current implementation)
T_{coh} = coherent integration time = 4 ms, T_{ker}= time to launch one GPU kernel ~ 15 µs

GNSS/IMU (LiDAR) Integration Filter

- Based on RTK-LIB, co-op with Tokyo University of Marine Science
- GNSS/IMU Filter developed in RTK-LIB style
  - Pure C, Hand-coded matrix operations
  - To be a built-in RTK-LIB feature, ensuring compatibility and ease of integration
  - Support and distribution via RTK-LIB (github, RTK-LIB viewer)
- Filter
  - 15 states loose coupling
  - Strapdown algorithm implemented using Euler angles or quaternions
  - Filter position error estimation implemented using radians or meter
  - Currently evaluated and tuned

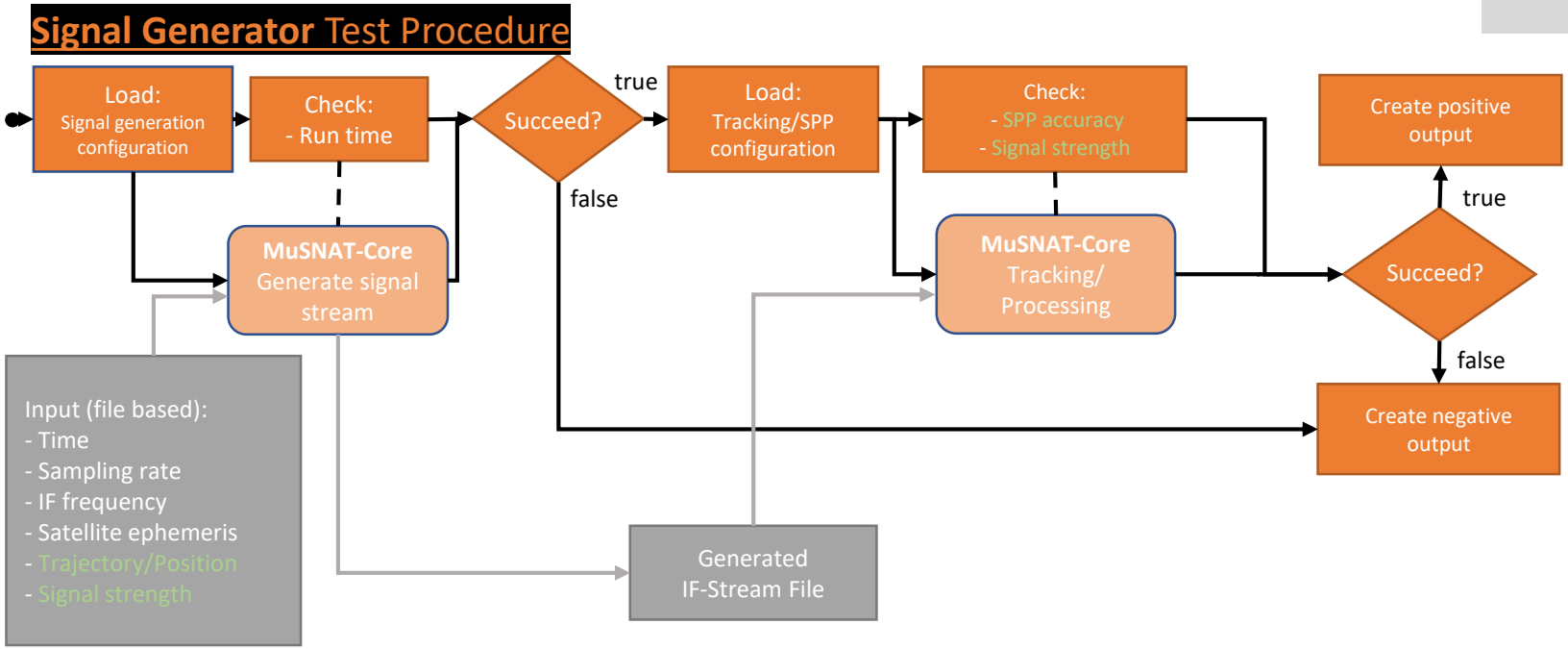
```
/* Form skew symmetric matrix; type = 0: Euler, type = 1: Quaternion */
int skew(double *omega, double *skewMat, int type)
{
    if (type == 0)
    {
        skewMat[0] = 0.0;
        skewMat[1] = omega[2];
        skewMat[2] = -omega[1];
        skewMat[3] = -omega[2];
        skewMat[4] = 0.0;
        skewMat[5] = omega[0];
        skewMat[6] = omega[1];
        skewMat[7] = -omega[0];
        skewMat[8] = 0.0;
```

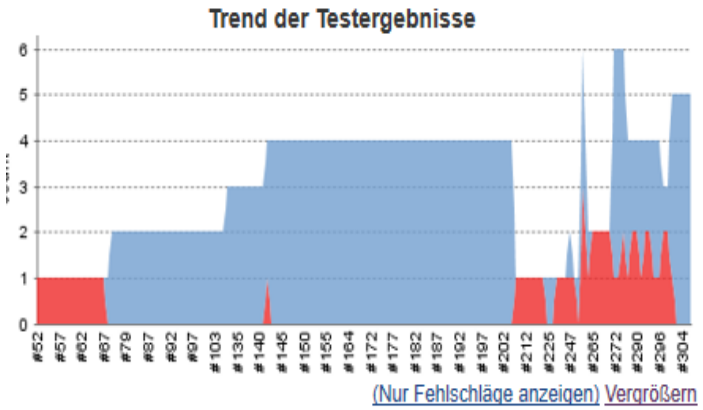


Automated Test Framework

- Jenkins
  - Automatically builds the current version of the MuSNAT
  - Test the build with Skripts
  - Evaluates the Test cases
  - Provide statistics and trends

Integrated Test cases
- RinexFilter / GnssInsFilter
- Start-Stop / Receive Epoch Data
- Memory Leak
- Repetition
- Signal Generator





Thank You !

The work has been made possible due to funding from the German ministry of economic affairs managed by the

DLR Deutsches Zentrum für Luft- und Raumfahrt

Test-users welcome!