THALES

OBC-hosted Star Tracker software

Summary

- **Activity overview**
- **Context and objectives**
- **Star Tracker SW:**
  - Problem Statement
  - Target platform
  - Star Tracker algorithm innovations (UoL)
  - Components of Star Tracker SW
  - Software architecture
  - Performance analysis
  - Profiling and optimization Optimizations
- **Star Tracker SW:**
  - Description of Testing Environment
  - Methodology and results

Activity overview

The main objective of the R&D project :

- Design, Development and Test of an innovative OBC hosted Star Tracker Software

Involved parties (The prime Contractor and Sub-Contractors):

- TSR (PC) will be mainly in charge with:
  - Overall management of Project
  - Implementation of SW versions on target and testing the SW versions.
  - Acquisition of flight software expertise & design rules
- TAS-F (SC)will be mainly in charge with:
  - SW requirements & Validation plan; aid in analysis of results; providing test scenarios
- UoL (SC) will be mainly in charge with:
  - Design of algorithms and modification of algorithms in order to meet the req. specified by TAS-F, support of TSR for porting of SW versions
- TAS-E (SC) CCN1 activity:
  - SW specification, integration testing and support for LEON3 UART & SPI drivers for experimental payload.

Context and Objectives

Recent innovations by UoL in the Star Tracker algorithms opened up the possibility of embedding, in the same central computer, Star Tracker Software the typical Platform software

Main advantages:

- Data processing is done on a processor shared with platform computer
- Simplification of HW design for complete Star Tracker solutions
- Potentially lower cost of Star Tracker Optical Heads
- Increased flexibility: Star Tracker performance and robustness will be contained in the Star Tracker software itself, with implementation of advanced functionalities such as data fusion or rate estimation.

Context and Objectives

Objectives:

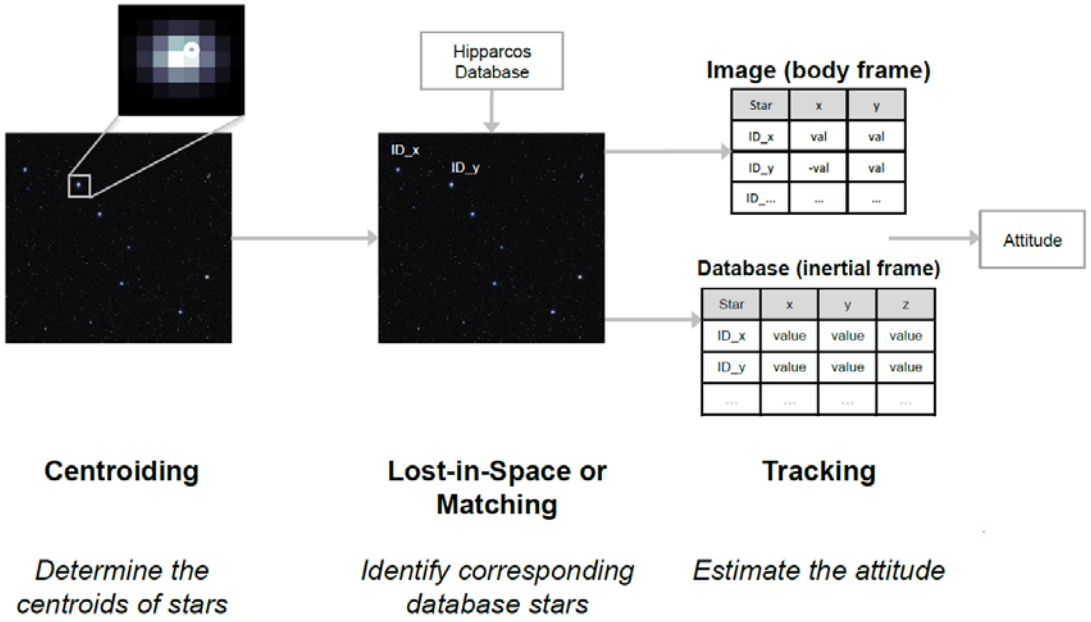
- Provide TAS-F some knowledge in Star Tracker algorithms design
- Provide UoL the possibility to test its algorithms against constraints and needs, and optimize them to take into account the latest technologies that will be soon embedded in Optical Heads
- Provide TSR the necessary competences and skills to develop space-qualified software

Problem statement

Unambiguously determine attitude of Satellite/Spacecraft based on images of stars

Star tracker in a nutshell:

- Identify stars in OH field of view (body frame)
- Match stars with those in Star database (inertial frame)
- Solve **Wahba's problem** to extract rigid transformation between frames

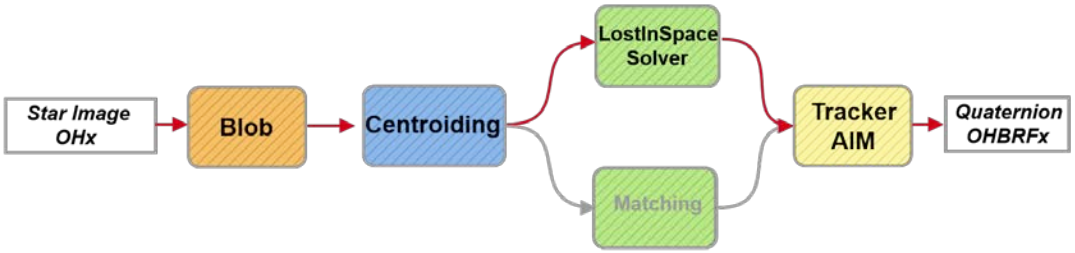


STR-SW typical processing flow

Two main processing modes:

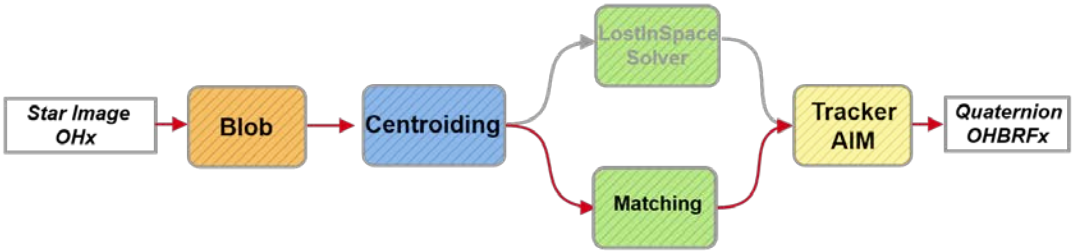
- Cold start
- Normal operation

Initial acquisition or after loss of attitude



What can be improved ?

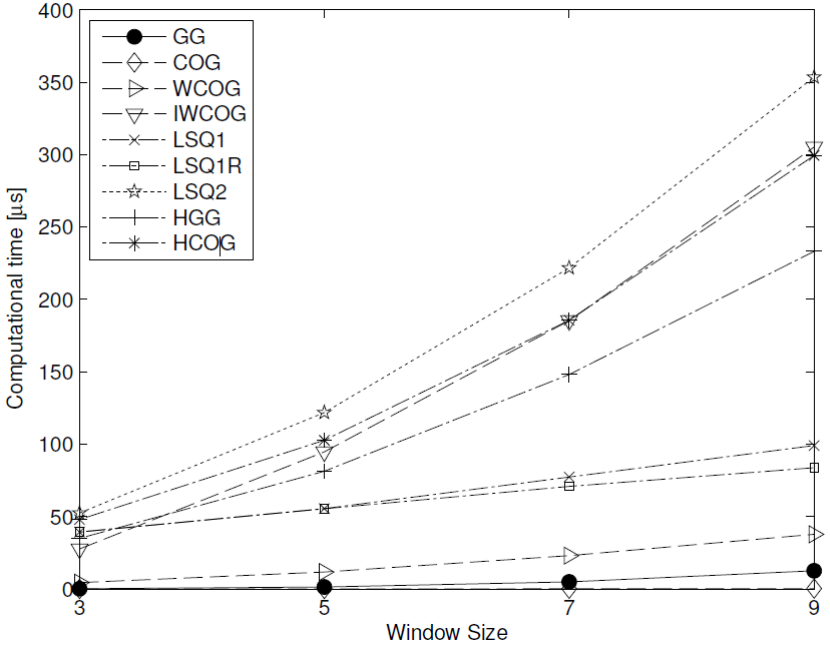
Normal operation



STR-SW Innovations (UoL)

Centroiding

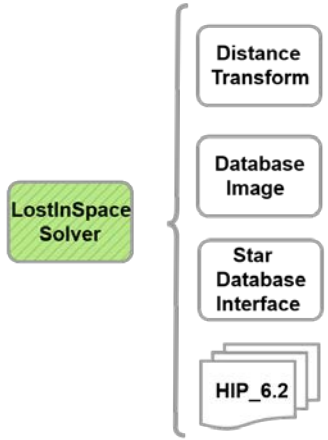
- Fitting a point spread function (PSF)
  - LSQ1, LSQ1R, LSQ2
  - Advantage: High accuracy
  - Disadvantage: High computational load
- Calculating center of gravity & variations
  - CoG, WCoG, IWCoG
  - Advantage: Low computational load
  - Disadvantage: Lower accuracy
- ***Gaussian Grid (GG) algorithm (UoL)***
  - Fits an elliptical Gaussian
  - Closed form expressions can be derived
  - Advantage: ***High accuracy & Low computational load***

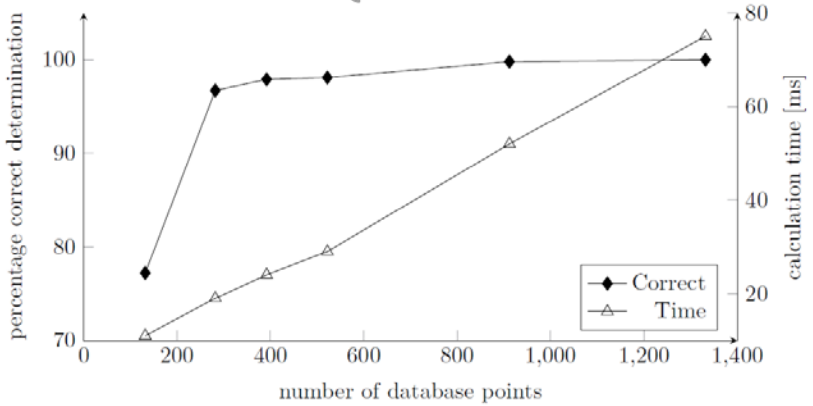


STR-SW Innovations (UoL) cont'd

Lost-in-space Algorithm

- **Robust to false stars**
- **Robust to missing brightest stars**
- Use of features such as inter-star angle and star magnitude as pre-filter
- Use of image processing technique Shortest Distance Transform for validation
- Advantage:
  - Transforms camera image into 2D look-up table which can be used to compare camera image with database





STR-SW Innovations (UoL) cont'd

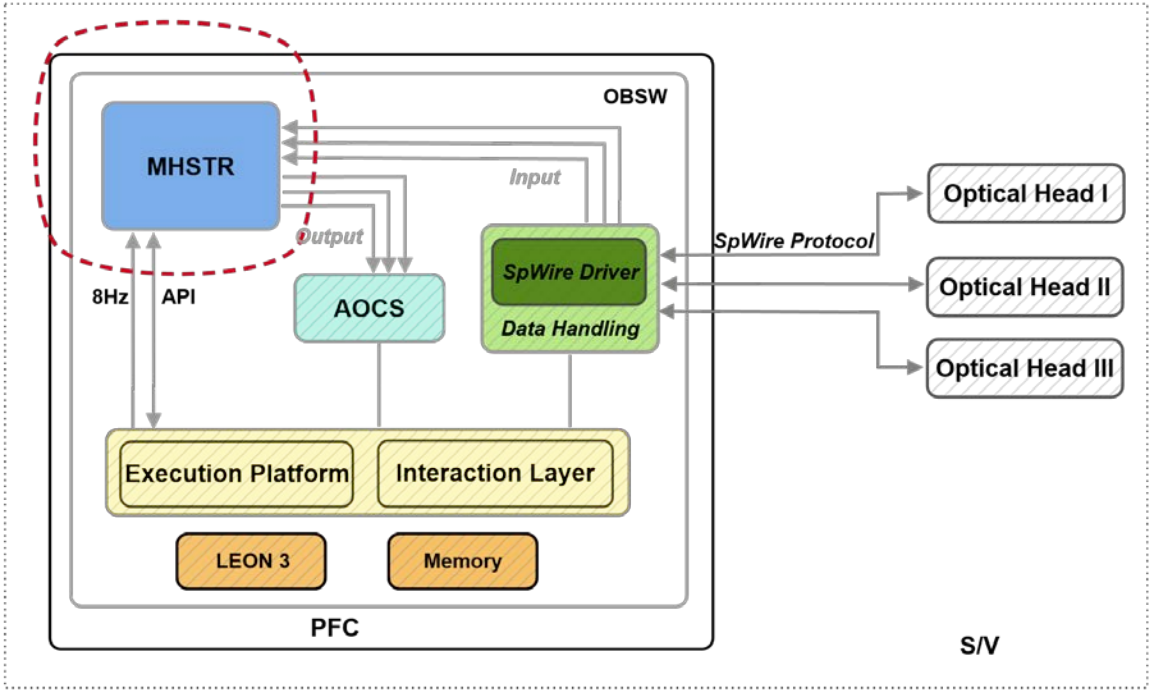
Tracking - AIM

- Determines the attitude using focal plane coordinates
- Conversion of unit vectors of the database stars determined in previous steps resulting in increased efficiency
- Requires a coarse estimate of the current attitude (from LISA or direct usage of previous quaternion)
- **Extremely fast procedure**
- **Closed form solution**

Target platform

Expected Target Hardware:

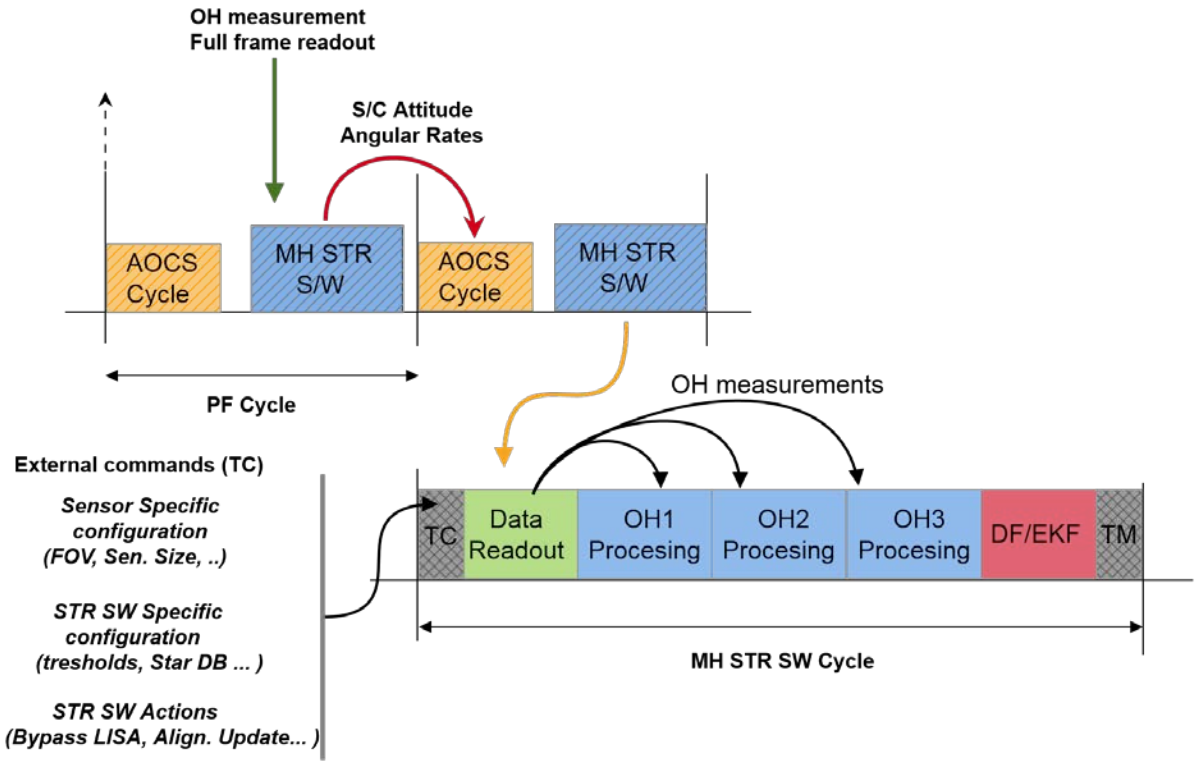
- Generic LEON3 mono-core processor (specificities of UT699 to be later considered)
- Memory Budget:
  - less than 1Mbytes for EEPROM.
  - less than 10.24Mbytes RAM



Target platform cont'd

Example of how STR SW integrates into the OBSW platform cycle

- Restrictions:
  - Use 20% of PF cycle for readout and execution
  - Process all OH frames in same cycle

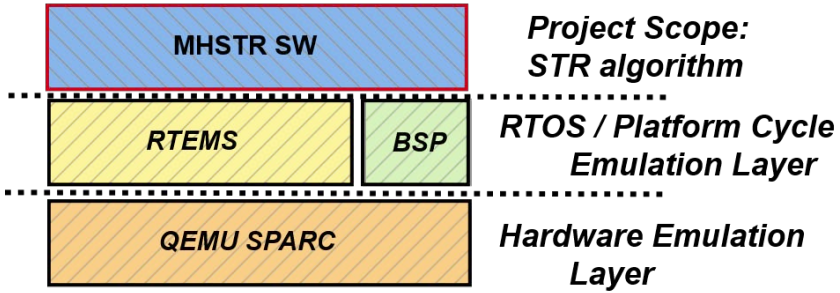


THALES

Target platform cont'd

Narrowing the scope of the R&D effort

- Focus on porting STR-SW algorithms provided by UoL
- **QEMU SPARC** is selected as simulator for the target processor
- Compiled QEMU from sources as included in the RTEMS toolchain source builder (RSB)
- SPW interactions not explicitly modeled (outside the scope of current R&D project)
- Software time & memory budgets must still be considered



Tools Used:

IBM Rational Rhapsody:

- UML tool
- Code generation
- Automatic documentation generation (ReporterPlus)
- Unit testing (Test Conductor)

MATLAB/Octave:

- Generate reference runs
- Analyze test scenarios provide by TAS-F

STR-SW development activity (overview)

UoL provided :

- Algorithm building blocks: Blob, Centroiding, LISA, Matching, AIM in C++
- MATLAB implementation of EKF (at later step in development cycle)

TAS-F provided:

- STR-SW requirements
- SW testing scenarios

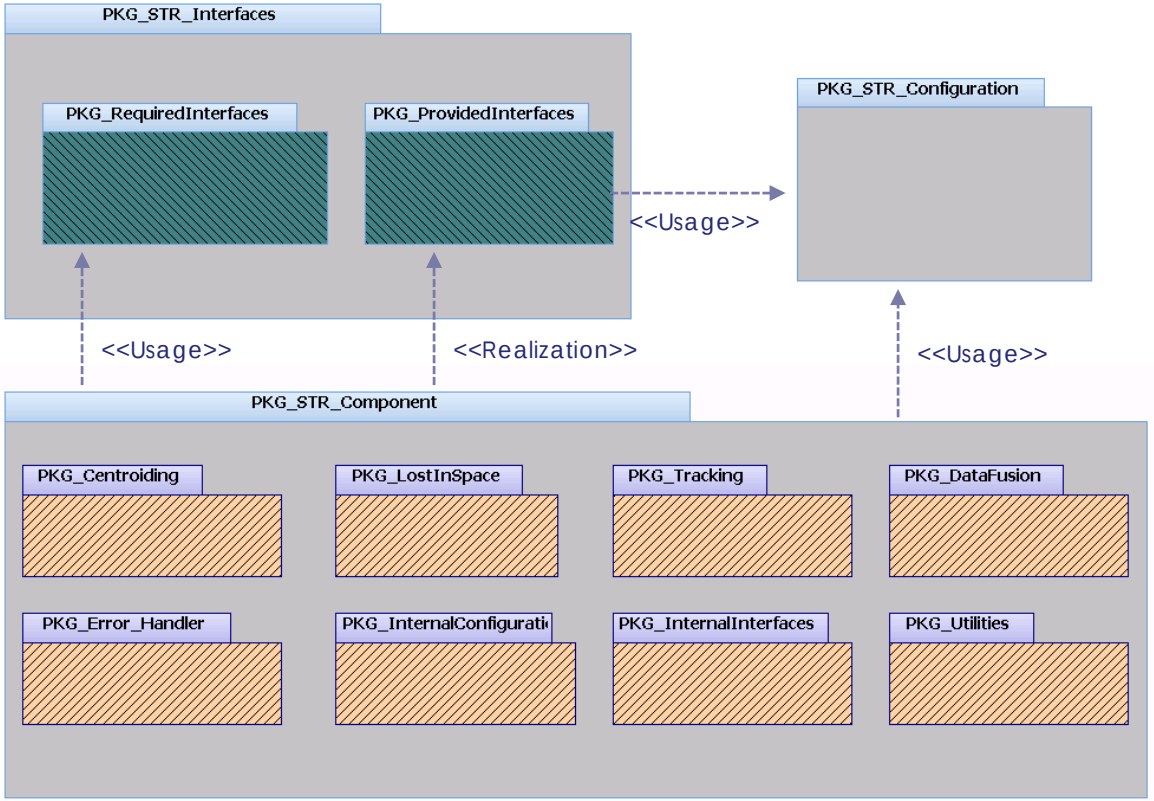
TSR :

- Ported algorithm meant for single OH to C language
- Extended algorithm to manage three optical heads
- Added support TC (external configuration of STR-SW parameters) and TM (STR-SW outputs in unified manner)
- Implement EKF in C language

Software Architecture: Overview (Rhapsody OMD)

Three top level packages:

- *External SW interfaces* (handle interaction with host environment)
- *SW Configuration parameters*
- *Main SW component* containing all algorithmic building blocks



Software Architecture: External Interfaces

StrProcAcqIf

- Get quaternions, angular rates, quality values of processed results

StrProcCmdIf

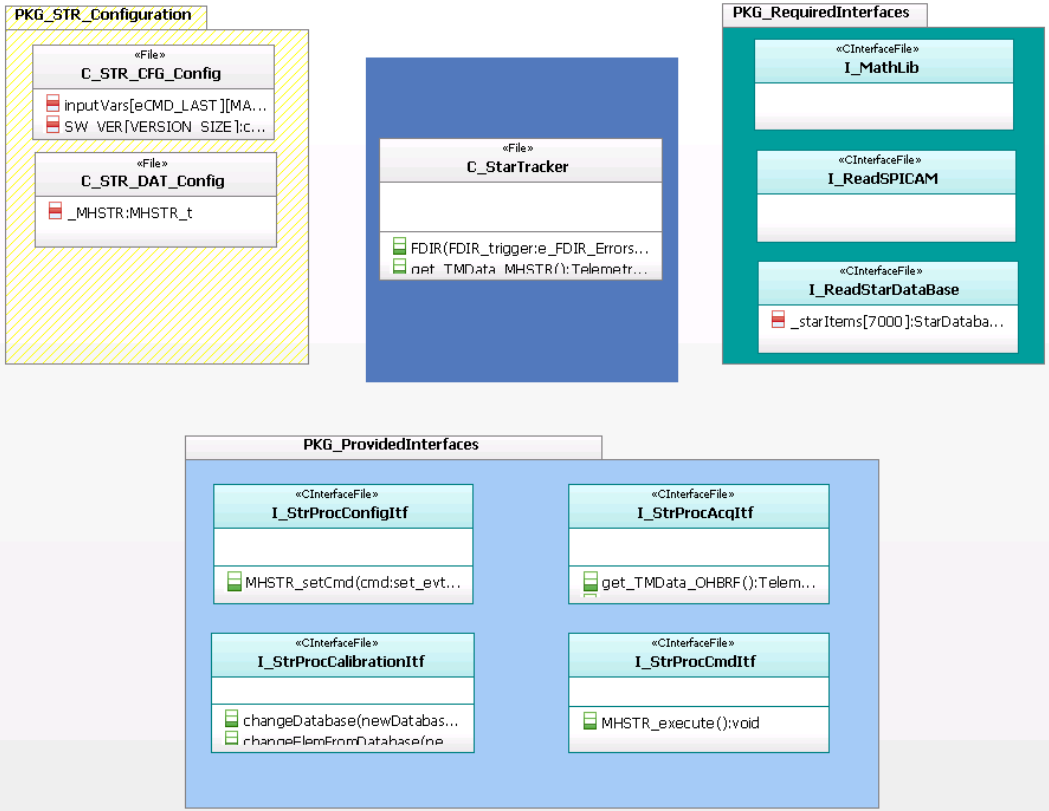
- Command the STR SW

StrProcConfigIf

- Exposes operations related to configuration parameters needed for the processing

StrProcCalibrationIf

- Set-up calibration of database



Software Architecture: Internal Interfaces

I_Centroiding

- Interface for centroiding routine

I_LostInSpace

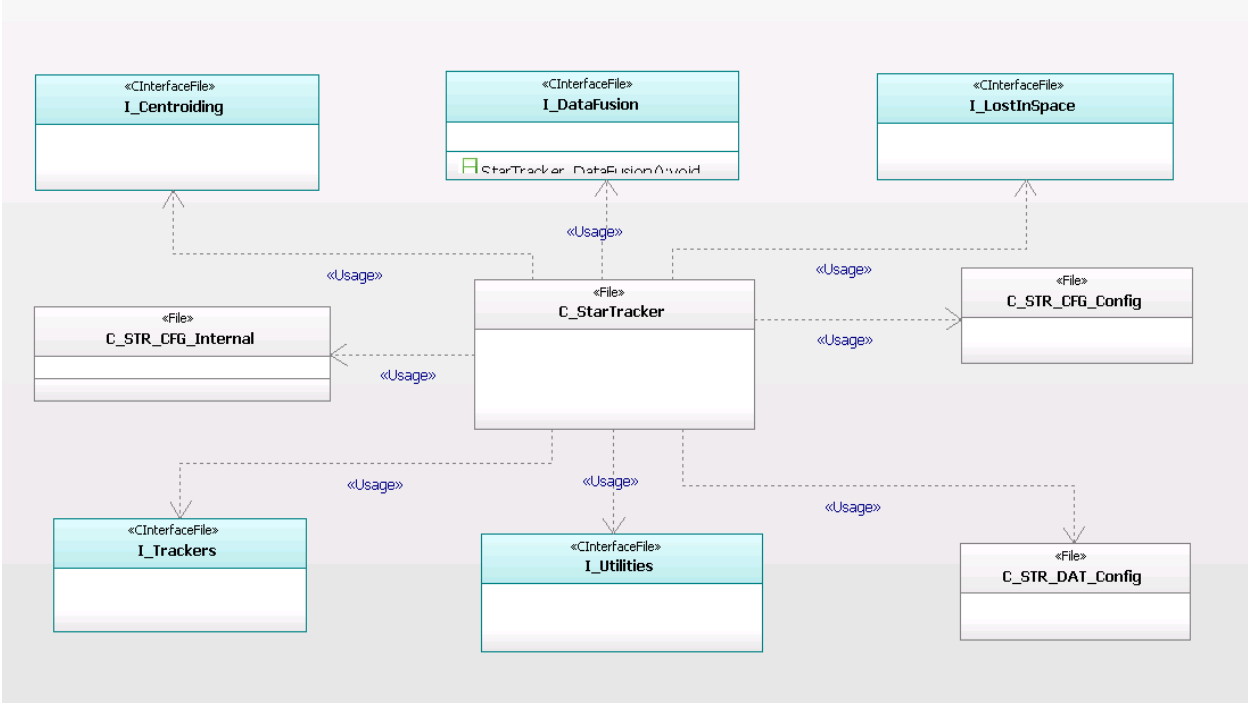
- Interfaces for distance transform, database and lost in space routines

I_Trackers

- Interfaces for matching and AIM routines

I_DataFusion

- Interface for the Kalman filter



Software Architecture: Internal Interfaces

I_Utilities

- Interface for various functions like multiplication of matrix or quaternions

C_STR_CFG_Internal

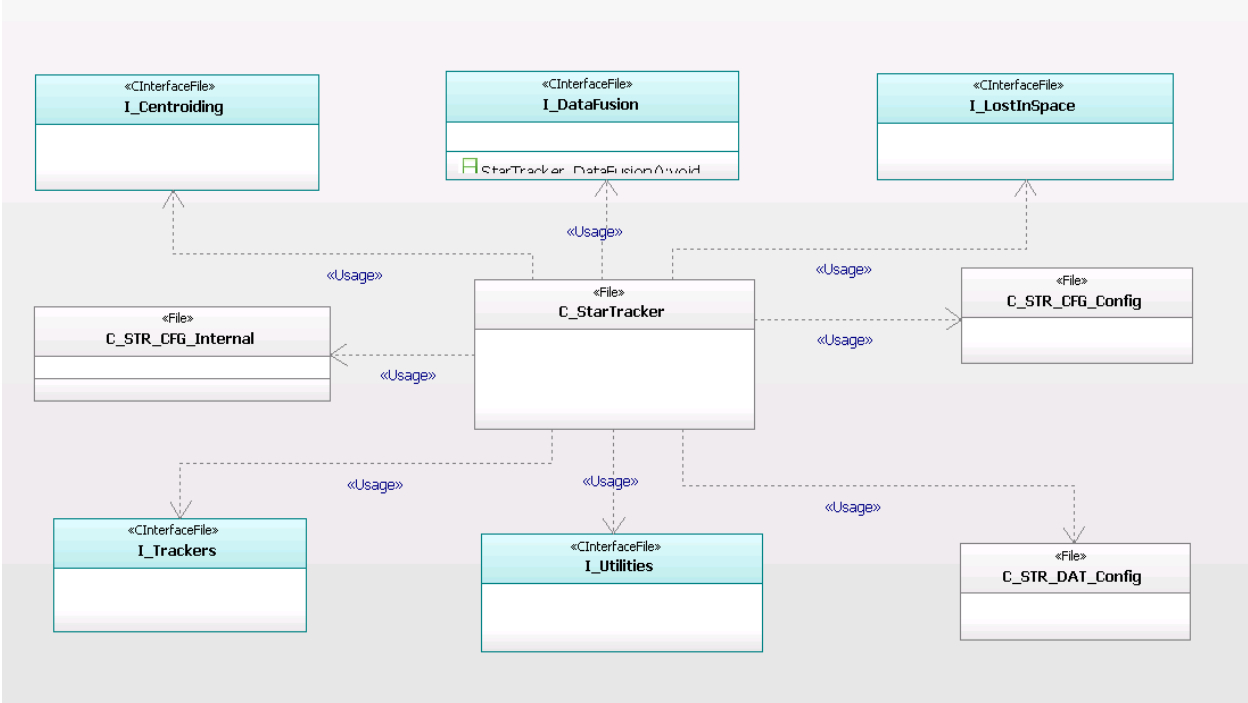
- Contains buffer lengths, distance map maximum size

C_STR_CFG_Config

- Contains all structures used to define commands

C_STR_DAT_Config

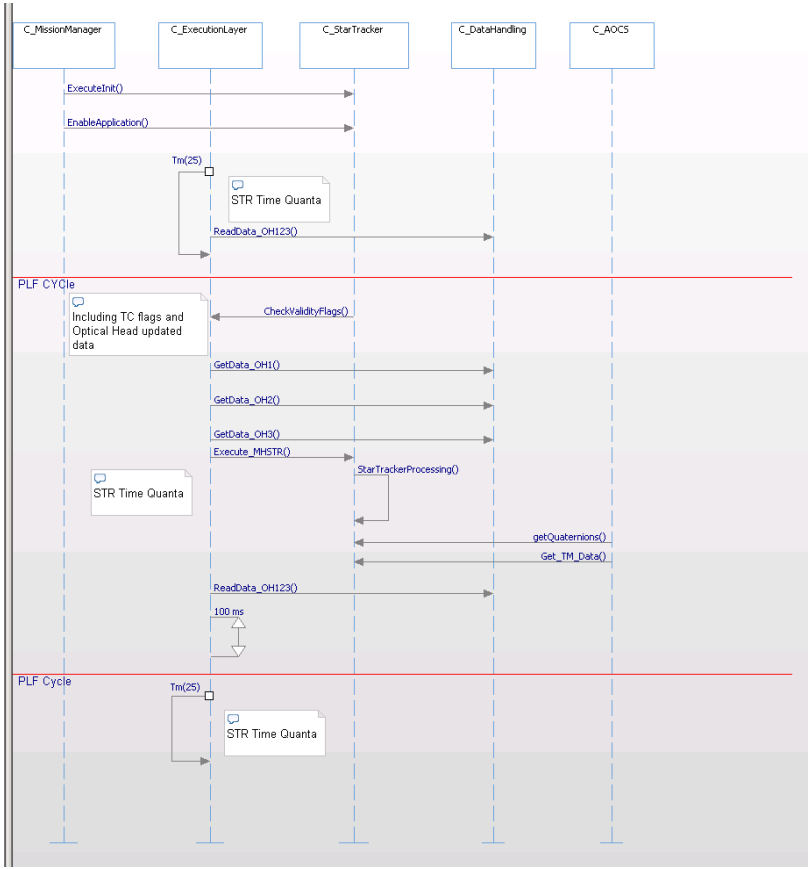
- Contains all structures regarding telemetry data



Typical Call Sequence

This sequence diagram represents the overall behavior of the Star Tracker Component:

- The execution layer will get image data from OH parsed by the data handling layer
- The image data is sent to STR SW and the processing phase is called
- After processing phase the telemetry data are extracted and sent to ground station



STR-SW Memory analysis

Constraints in memory budget (RAM & EEPORM) prompted in depth memory analysis

Tools used:

- *Valgrind* + *massif* tool
- Size command in Linux
- Checked with *–fstack-usage* parameter from *gcc*

Results:

- Stack Usage:
  - 392.384Kb (computed using sparc-gaisler-rtems5-gcc)
  - 302Kb (output of Valgrind and massif)
- Segment size:
  - ***Text***: 5.8 Kb; ***Data***: 141 Kb; ***BSS segment***: 27.5MB

STR-SW Performance analysis

Performed with perf tool from Linux utilities

- The bank of registers for the x87 instructions set was monitored
- Application was running in QEMU
- **taskset** used to set QEMU core affinity

Command line instruction:

taskset --cpu-list 1 sudo perf stat -e r530110 -e r531010 -e r532010 -e r534010 -e r538010 ./run_qemu.sh

- r530110 – the code for the x87 registers
- r531010 – the code for the SSE floating point packed double registers
- r532010 – the code for the SSE floating point scalar single registers
- r534010 – the code for the SSE packed single registers
- r538010 – the code for the SSE scalar double registers

STR-SW Performance analysis cont'd

Performance analysis results LISA

- **Unoptimized** version yielded 1.6 billion x87 instructions for three LISA iterations
- Not feasible to run on LEON3 in given platform cycle timeslot
- High resource usage prompted further investigation

```
Performance counter stats for './run_qemu.sh':

    1.601.397.481      r530110
                0      r531010
                0      r532010
                0      r534010
                5      r538010

    253,948229650 seconds time elapsed
```

LISA floating point operations

Performance analysis in Tracking

- The Star Tracker software was initialized with a valid quaternion for all OH and the LISA routine was bypassed

```
Performance counter stats for './run_qemu.sh':

       49.401.224      r530110
                0      r531010
                0      r532010
                0      r534010
                5      r538010

    88,544534271 seconds time elapsed
```

Tracking floating point operations

STR-SW Profiling

Callgrind was used for profiling

- The MHSTR test app compiled under Linux using gcc
- Majority of the CPU resources were used in ***generateDatabaseImages(..)***
- Approximately 50 million *atan2* calls needed
- Routine needed to be optimize by TSR

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000000c30	ld-2.23.so
100.00	0.00	1	_start	main
100.00	0.00	1	_dl_runtime_resolve_sse	ld-2.23.so: dl-trampoline.h
100.00	0.00	1	(below main)	libc-2.23.so: libc-start.c
100.00	0.17	1	main	main: main.c
92.89	0.00	4	MHSTR_execute	main: C_StarTracker.c
92.87	0.00	1	MHSTR_running	main: C_StarTracker.c
92.45	0.00	3	performLostInSpaceMode	main: C_StarTracker.c
92.37	0.16	3	LISASpeedy	main: C_LostInSpace.c
87.38	3.65	3 999	generateDatabaseImage	main: C_StarDatabase.c
52.22	3.95	24 873 780	isInFieldOfView	main: C_StarDatabase.c
43.73	4.64	24 877 776	toSphericalCoordinates	main: C_Utilities.c
35.85	3.86	49 755 552	atan2	libm-2.23.so: w_atan2.c
31.99	31.99	49 755 552	__ieee754_atan2_sse2	libm-2.23.so: e_atan2.c, fenv_pri…
24.42	6.13	24 873 780	rotateVector	main: C_Utilities.c
18.29	9.10	24 873 783	toDCM	main: C_Utilities.c
13.46	5.44	3 317 588	<cycle 1>	ld-2.23.so
9.19	6.30	24 873 798	getNormalized	main: C_Utilities.c

Function call details for UoL implementation

STR-SW Optimization

The *generateDatabaseImages()* routine has two major objectives:

- Determining DB stars in camera field of view for a given OH attitude quaternion
- Converting stars in camera field of view (FoV) from Cartesian to focal plane

TSR proposed algorithm for checking if a vector is in field of view which reduces the use of trigonometric functions.

- Vector dot products are used to check if a point is in OH FoV.
- At most three dot products are computed, in the worst case scenario.

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000000c30	ld-2.23.so
100.00	0.00	1	_start	main
100.00	0.00	1	_dl_runtime_resolve_sse	ld-2.23.so: dl-trampoline.h
100.00	0.00	1	(below main)	libc-2.23.so: libc-start.c
100.00	0.93	1	main	main: main.c
60.04	0.00	4	MHSTR_execute	main: C_StarTracker.c
59.94	0.00	1	MHSTR_running	main: C_StarTracker.c
57.56	0.00	3	performLostInSpaceMode	main: C_StarTracker.c
57.44	0.89	3	LISASpeedy	main: C_LostInSpace.c
41.95	30.54	3 309 590	<cycle 1>	ld-2.23.so
35.72	27.04	3 145 728	_IO_vfscanf <cycle 1>	libc-2.23.so: vfscanf.c, scratch_bu…
29.87	27.66	3 999	generateDatabaseImage	main: C_StarDatabase.c
8.80	6.68	3 856	transformDatabaseImage	main: C_LostInSpace.c
8.77	8.77	19 741	__GI_memset	libc-2.23.so: memset.S
8.58	0.01	15 710	setDBMaxSize	main: C_StarDatabase.c
6.88	2.59	7 852	selectSmallFOV	main: C_LostInSpace.c

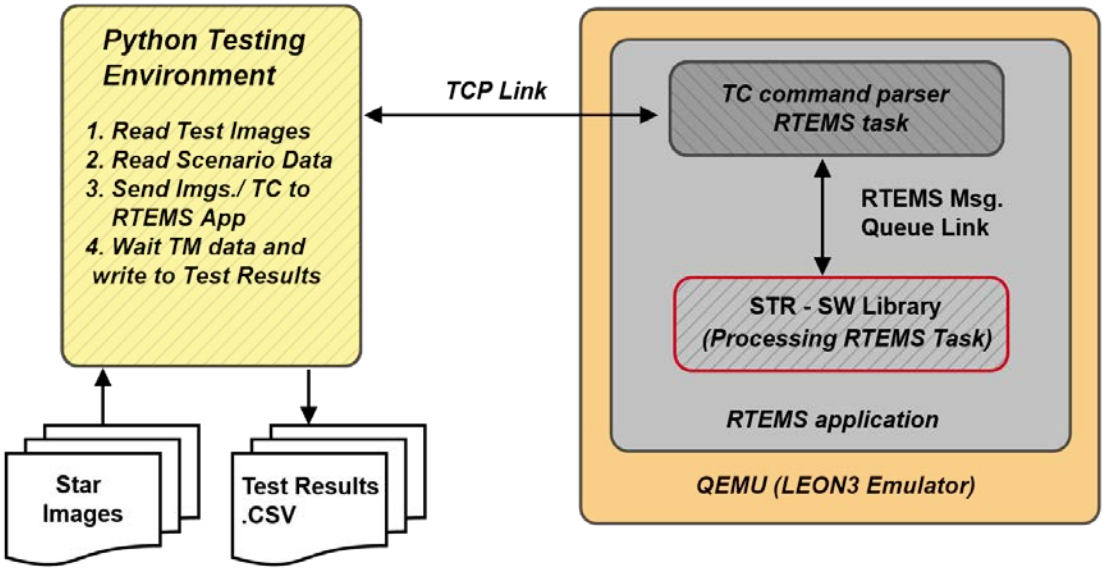
Function call details for TRS implementation

```
Performance counter stats for './run_qemu.sh':

       64.473.505      r530110
                0      r531010
                0      r532010
                0      r534010
                7      r538010

    109,490540579 seconds time elapsed
```

LISA optimized: floating point operations

STR-SW Precision Test Environment

- QEMU emulator which is emulating the LEON3 CPU (initial HW baseline)
- RTEMS command parser task used to receive TC/Images.
- The STR SW (.c code) is called within a RTEMS OS task .
- Python script controls the test execution and saves the test result in a .csv file



STR-SW Precision Test Results

Tested on SPICAM datasets provided by TAS-F

- Scenarios include various maneuvering conditions
- Static scenarios to test attitude determination availability for various attitudes
- Dynamics scenarios which test robustness at various angular rates.
- Shadowing scenarios if attitude determination is still possible when one OH is partially or fully blinded.

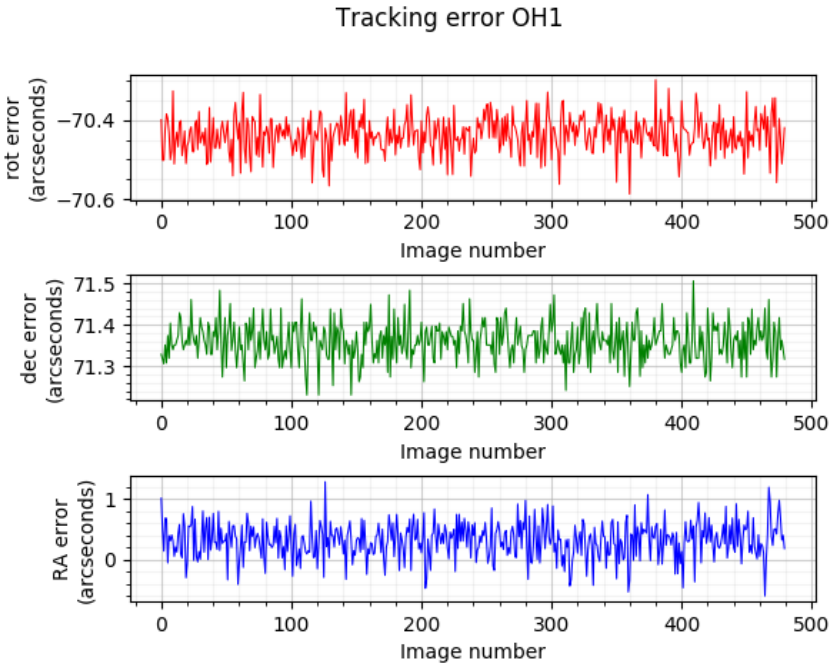
Following aspects where evaluated:

- Noise errors for single OH and multiple OH configurations
- Bias errors for single OH and multiple OH configurations
- Angular rates determination precision

STR-SW Precision Test Results

Example: Static Scenario (1/3)

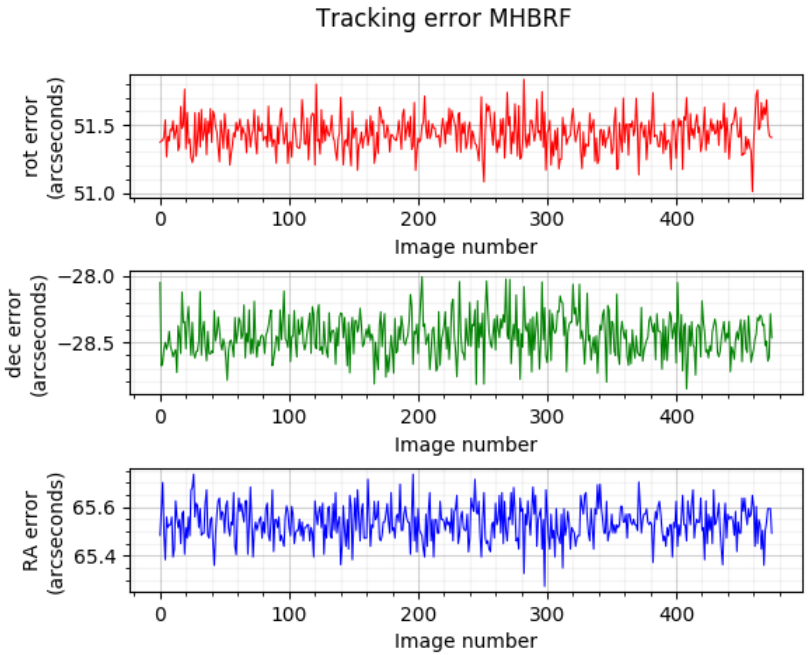
- Absolute measurement error (**AME**) for **IRF to OHBRF attitude quaternion**. Angular error between frames expressed using Euler angles.
- Attitude quaternion determined by each individual OH is compared with ground truth from scenario.
- Target:
  - Bias errors (3σ): < 20 arcsec
  - Noise errors (3σ): x,y < 20 arcsec
    z < 50 arcsec
- Conclusion:
  - Stable determination but with small bias (open issue)...



STR-SW Precision Test Results

Example: Static Scenario (2/3)

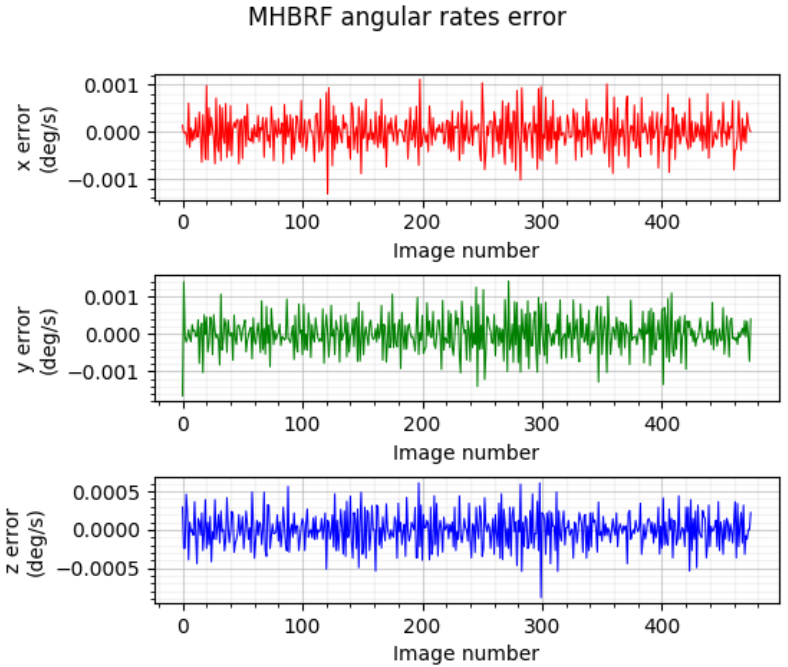
- Absolute measurement error (**AME**) for **IRF to MHBRF attitude quaternion**.
- Attitude quaternion determined after data fusion step is compared with ground truth from scenario.
- Target:
  - Bias errors (3σ): < 20 arcsec
  - Noise errors (3σ): < 5 arcsec
- Conclusion:
  - Similarly, stable determination but with small bias (open issue)...



STR-SW Precision Test Results

Example: Static Scenario (3/3)

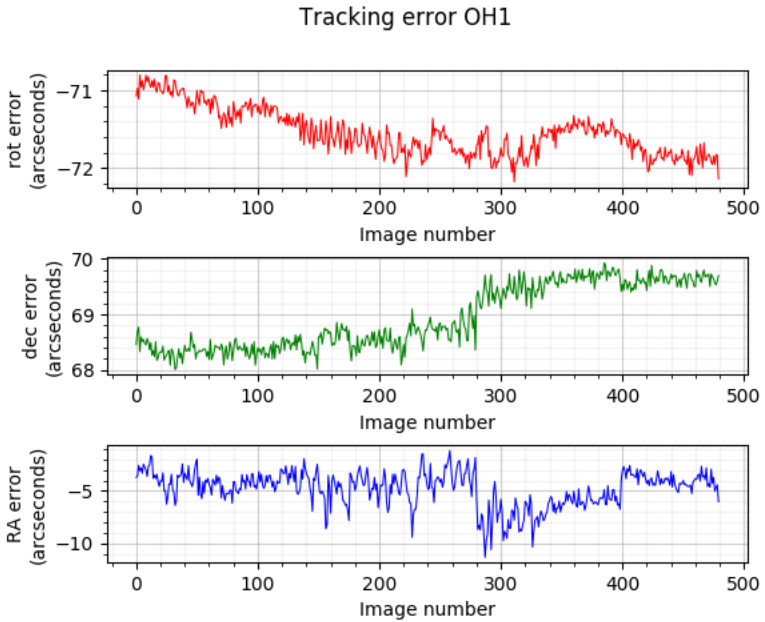
- **Angular rates** estimation error. Rates are expressed in MHBRF frame.
- Target:
  - Error < 0.01 deg/s in tracking mode,
  - Error < 0.05 deg/s in other modes.
- Conclusion:
  - Angular rates are properly estimated, errors below 10^-3



STR-SW Precision Test Results

Example: Dynamic Scenario (1/3)

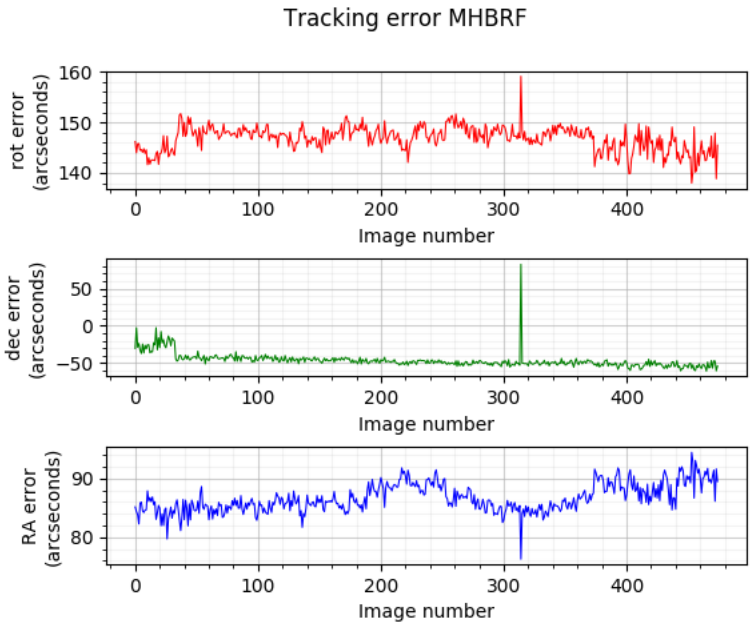
- Absolute measurement error (**AME**) for **IRF to OHBRF attitude quaternion**. Angular error between frames expressed using Euler angles.
- Target:
  - Bias errors (3σ): < 20 arcsec
  - Noise errors (3σ): x,y < 20 arcsec
    z < 200 arcsec
- Conclusion:
  - Stable determination but with small bias (open issue)...



STR-SW Precision Test Results

Example: Dynamic Scenario (2/3)

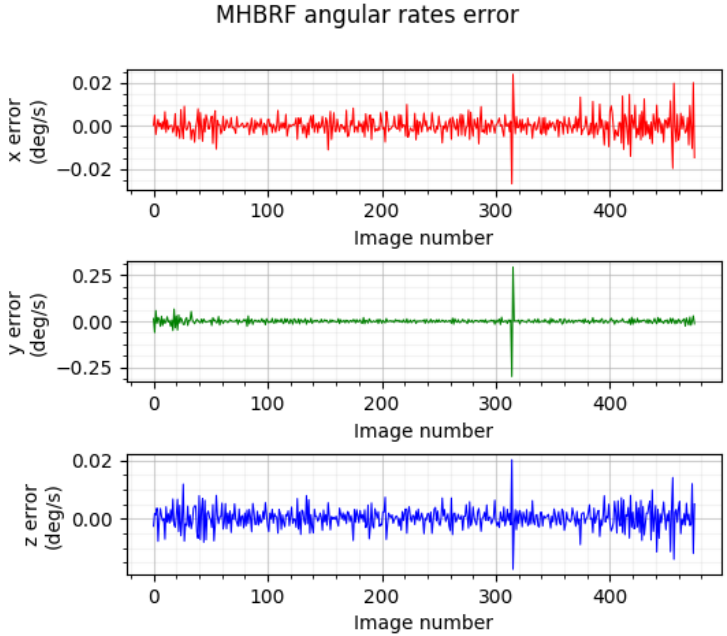
- Absolute measurement error (**AME**) for **IRF to MHBRF attitude quaternion**.
- Target:
  - Bias errors (3σ): < 20 arcsec
  - Noise errors (3σ): < 20 arcsec
- Conclusion:
  - Similarly, stable determination but with small bias (open issue)...



STR-SW Precision Test Results

Example: Dynamic Scenario (3/3)

- **Angular rates** estimation error. Rates are expressed in MHRF frame.
- Target:
  - Error < 0.01 deg/s in tracking mode,
  - Error < 0.05 deg/s in other modes.
- Conclusion:
  - Angular rates are generaly properly estimated



Lessons Learned

- TSR team has expanded it's knowledge about the inner workings of Star Tracker software
- TSR acquired the necessary competences and skills to develop space-qualified software and how to apply the adequate ECSS standards
- TSR gained knowledge about LEON3 architecture, it's advantages and limitations as well as emulation of said architecture in QEMU
- TSR understands the workflow and has the competence to test and validate further Star Tracker implementations

CCN1 Activity overview

The main activities/objectives :

- TSR develop space qualified SW
- TSR acquire knowledge about emulating LEON3 target on FPGA
- TSR development of serial drivers (UART & SPI) for LEON3 target
- TAS-E define SW requirements
- TAS-E integration tests for SW provided by TSR

CCN1Activity overview cont'd

For dev environment TSIM used to emulate LEON3 :

- GRLIB mono-core LEON3 IP minimal configuration
- GRLIB APB-UART
- GRLIB SPICTRL

For testing environment FPGA LEON3 emulated CPU:

- On target validation of correct functionality of:
  - Driver layer (correct peripheral initializations and overall interactions)
  - Message protocol layer

CCN1 Activity summary

Lessons Learned

- TSR acquired in-depth knowledge about development environment based on soft-core FPGA
- TSR understood the specifics LEON3 architecture & serial peripherals

Conclusions

- TSR successfully implemented flight ready software
- Software implementation validated

Conclusions

- Successfully ported the STR algorithm in C (from C++ & MATLAB code provided by UoL)
- Extended algorithm to support three optical heads
- Developed a integration test environment using Python, QEMU & RTEMS for STR-SW
- Still room for improving the precision as well as the performance (tuning still needed)
- Zynq may be a more suitable platform (much more processing power available)

Future Work

- Qualify software for Cat C criticality (provide full documentation, perform PA activities, etc.)
- Integration with real optical heads
- Run software on real target platform not only on emulator

Thank you!