The CaNaPy RTC: towards pre-correction of the LGS beacon

RTC4AO 2023

CaNaPy RTC status Report / 06.11.2023

David Jenkins

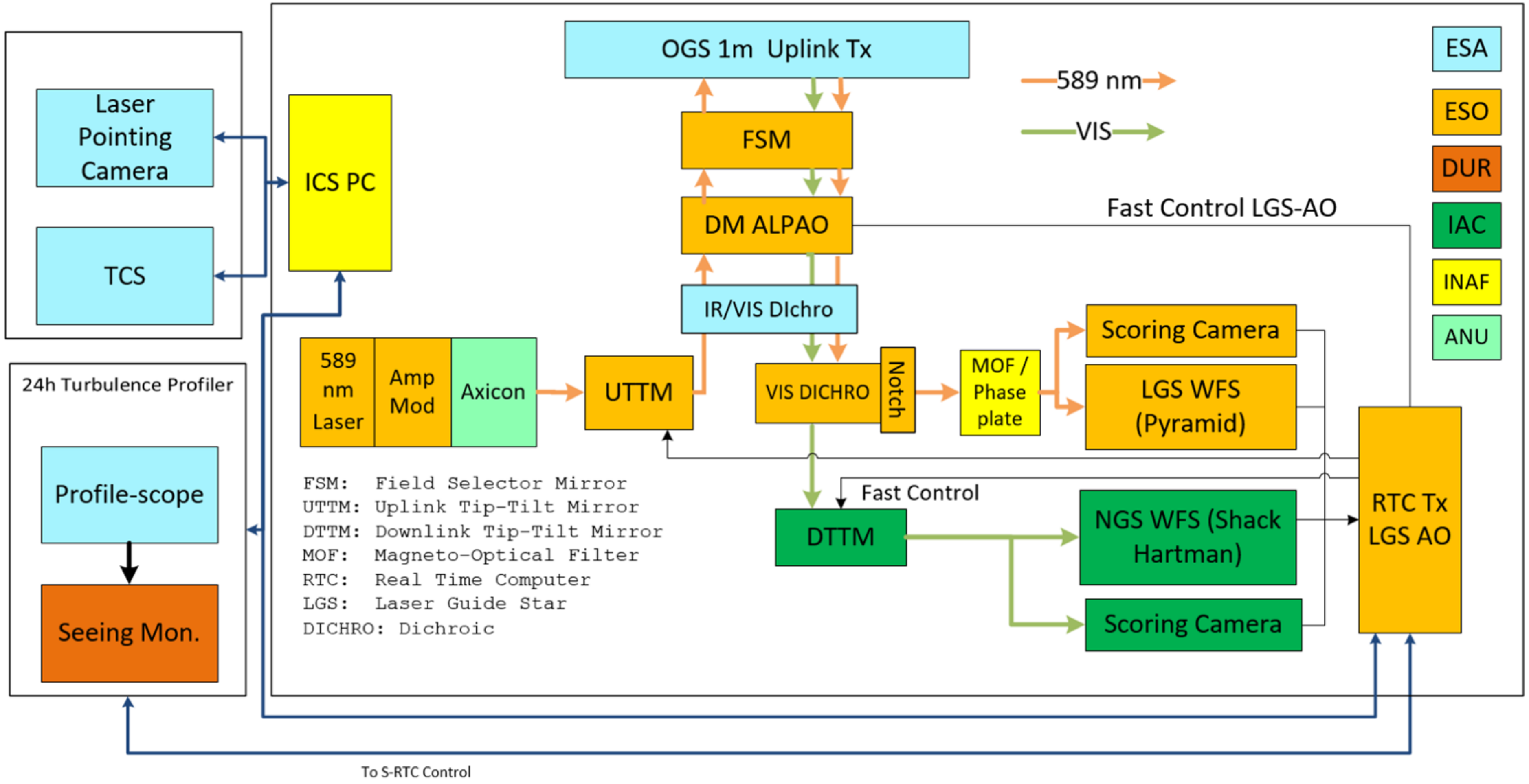
CaNaPy RTC developer ESO

What is CaNaPy?

- Demonstrate and Optimize LGS uplink pre-compensation, for the smallest LGS size (implies pulsed/chopped laser)
- Close the LGS-AO loop with Pyramid WFS, in monostatic configuration
- Optimize for operation in the visible, demonstrate performance
- Have an experimental facility for advanced LGS-AO R&D experiments (agreement with ESA done)

- So far foreseen CaNaPy experiments within the collaboration with ESA:
  - Demonstrate operation and control also in non-favourable seeing conditions (including daytime)
  - Test the time delay method (Ragazzoni, 1999) for the measurements of tip-tilt from the LGS
  - Test the candle-light method to have and use the sodium profile and its centroid during operations
  - Evaluate the advantages of the uplink pre-compensation in monostatic mode, vs more standard bistatic LGS-AO configurations

CaNaPy Concept



Project: CaNaPy	Title: System Block Diagram	DBC	Version: 4.0	Date: 27th July 2021

CaNaPy AO System

- LGS AO system with one high order ALPAO 97 DM and one LGS uplink jitter mirror
- A monostatic launch, laser is launched from the main telescope
- The laser is chopped (pulsed) synchronously with the WFSs shutters to avoid blinding the cameras during the propagation
- LGS Pyramid WFS with 40x40 pixels per pupil (4x4 oversampling) for High Order control
  - Using the OCAM 2S at 2kHz

CaNaPy AO System

- NGS Shack-Hartmann with 12x12 sub-apertures for Low Order tip, tilt and focus
  - Using the OCAM 2K also at up to 2kHz
- ALPAO 97-15 DM, provides high order modal correction with up to ~90 modes
- Downlink TT mirror in front of the NGS SH-WFS only
- Uplink LGS Jitter Loop mirror to keep LGS pointing
- Each WFS has a "scoring" camera to look at the PSFs
  - LGS-WFS uses an EVT HB-1800-S, NGS-WFS uses a Hamamatsu ORCA

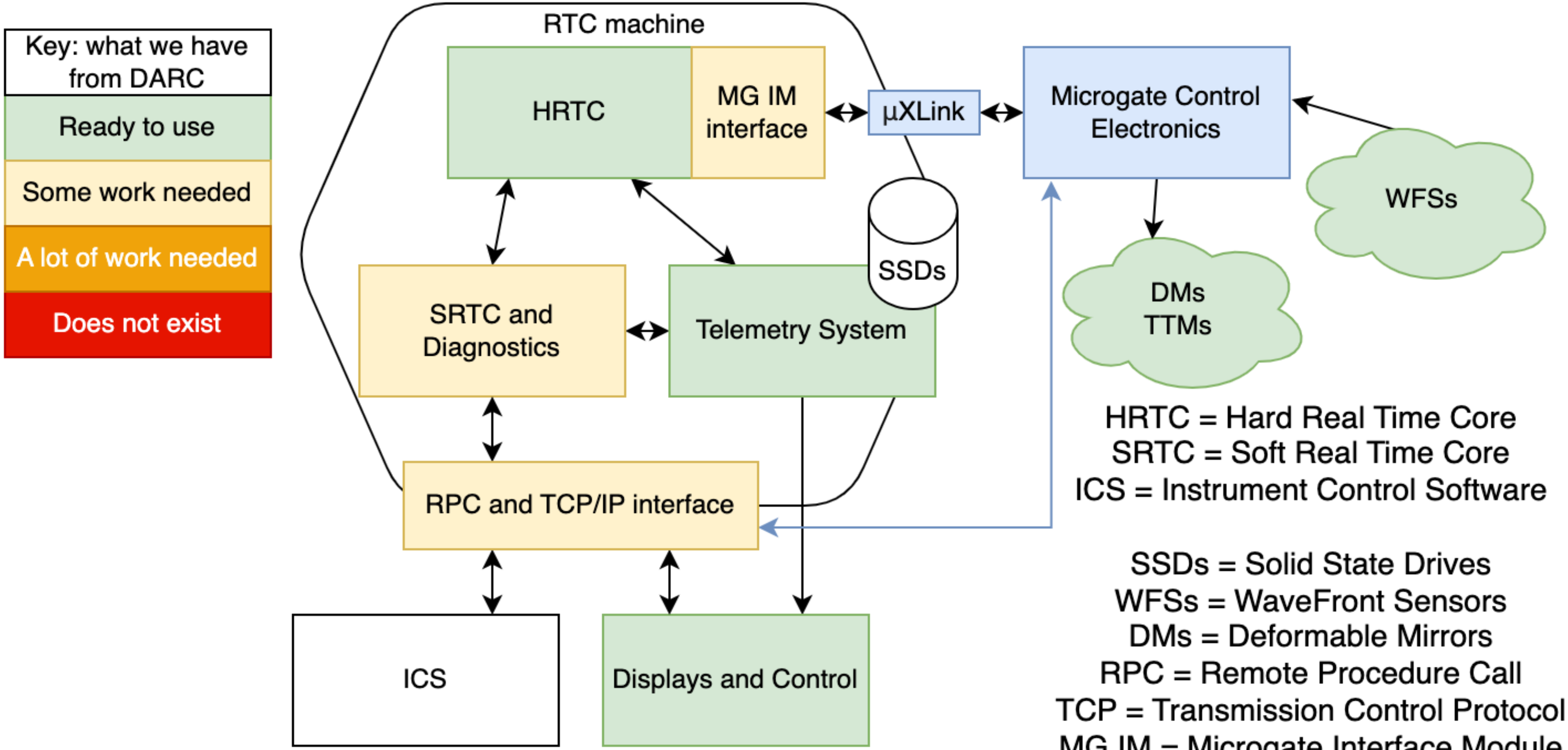
ALASCA upgrade with Microgate

- Converts CaNaPy into an optical feeder link experiment for Satellite communications
- Introduces an IR Tx and Rx path
  - The Rx path includes an IR Py-WFS using a CRED 2-lite, otherwise a clone of the LGS Py-WFS
- Upgrades the AO hardware interfaces to the RTC
  - Microgate hardware streams pixels directly into CPU memory
  - Actuator command are read directly from CPU memory and sent to the hardware
  - Reduces the load on the CPU to concentrate on AO reconstruction
- The IR WFS uses the satellite downlink as a guide star

CaNaPy RTC

- The RTC processing is done on a COTS Dell server running CentOS 7.5 (legacy due to kernel requirements for PCIe ALPAO interface)
- CaNaPy is a small-scale experimental AO system
- CaNaPy RTC sub-system developer, operator and maintainer: David Jenkins (ESO) with support from Microgate for hardware interfaces

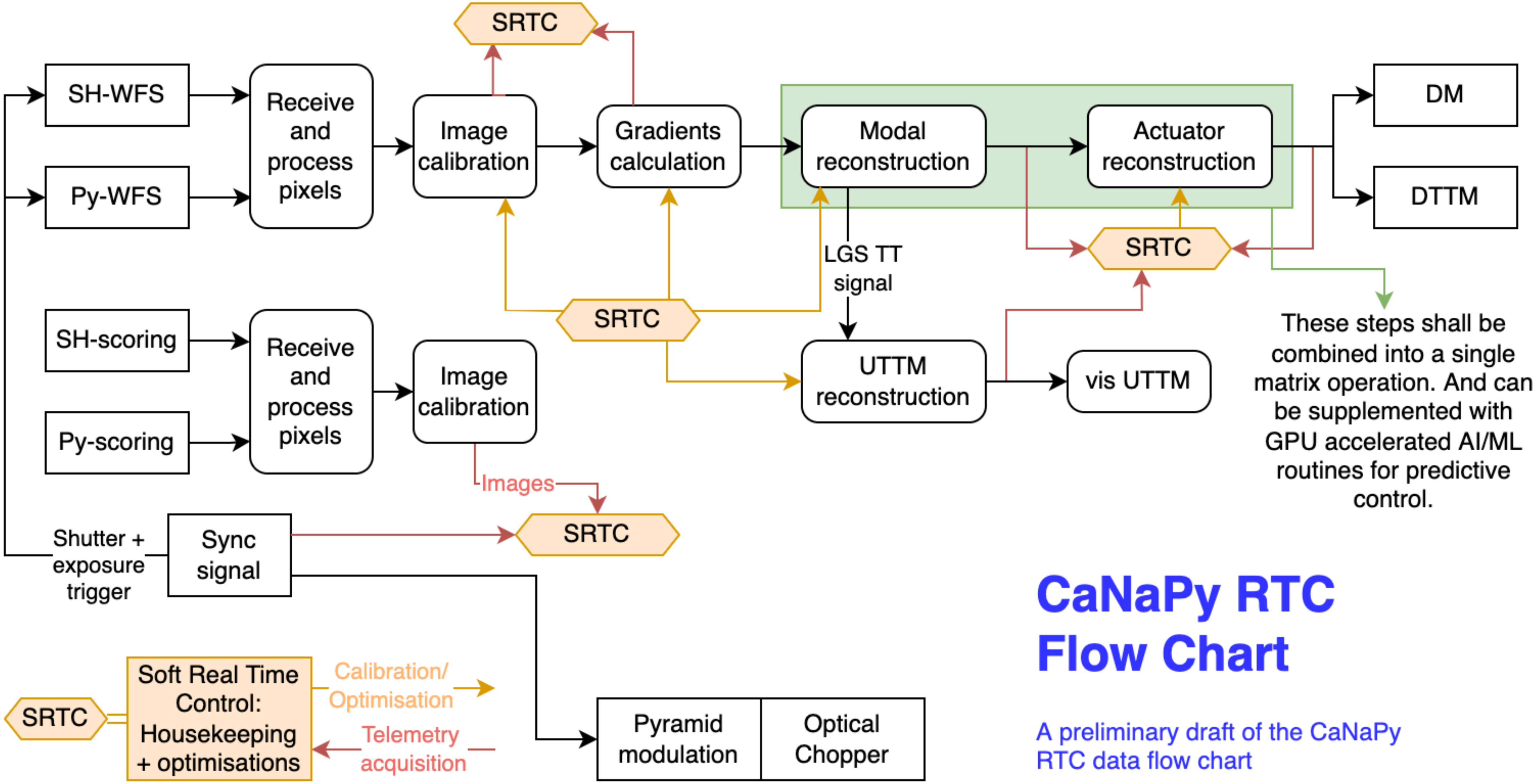
CaNaPy RTC Status



HRTC = Hard Real Time Core
SRTC = Soft Real Time Core
ICS = Instrument Control Software

SSDs = Solid State Drives
WFSs = WaveFront Sensors
DMs = Deformable Mirrors
RPC = Remote Procedure Call
TCP = Transmission Control Protocol
MG IM = Microgate Interface Module

CaNaPy RTC Flow Chart



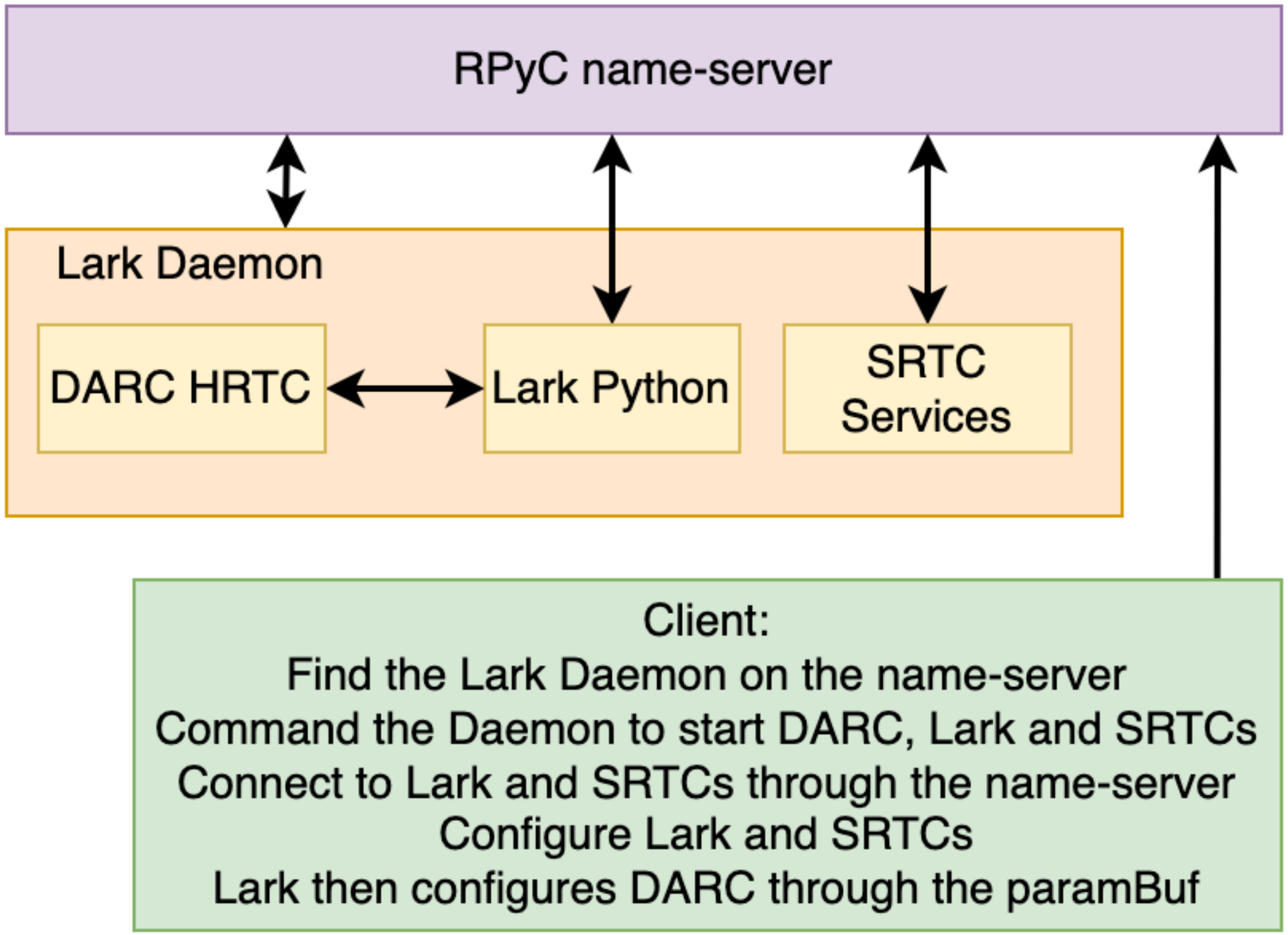
CaNaPy RTC Flow Chart

A preliminary draft of the CaNaPy RTC data flow chart

Software Architecture

- HRTC uses DARC c code
  - DARC is multithreaded C with shared memory parameter buffer and circular buffers for telemetry
  - ”Horizontal” multi-threading, each thread processes a portion of pixels through to the partial actuators
  - For CaNaPy only a single thread is needed, small problem size
  - New camera and mirror libraries to interface with the Microgate hardware
- HRTC control and SRTC uses Lark, pure Python and Python C-Extensions
  - Python code can apply parameters to the HRTC through the shared memory parameter buffer
  - It can read telemetry from the circular buffers
  - Custom Python C-extensions use the DARC shared libraries directly
  - Only implements the required functionality
- Systemd services are used to launch the DARC core (with root privileges) and to launch Python RPyC services for control and SRTC (with user privileges)
  - All process are run in the background, Python services spawn new processes
  - Uses RPyC, zeromq, and shared memory for communication

Lark Concept

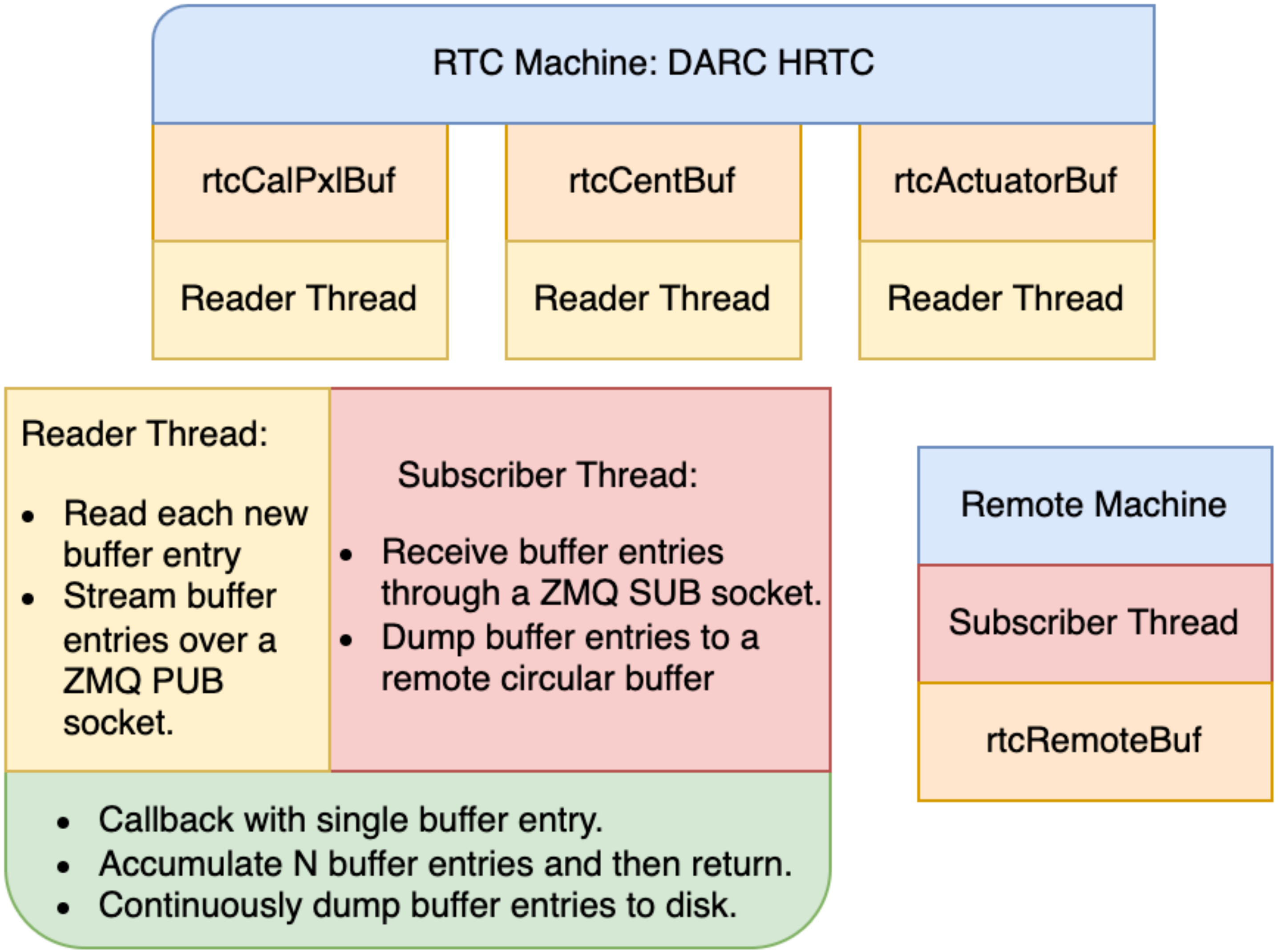


Modes:

- A mode is a collection of DARCs/Larks and Services that are launched together.
- Used for configuring an observing mode
- These can be launched by a client by processing the mode configuration and command the Daemon to start the processes

Lark Telemetry System

- Each telemetry buffer has a reader thread implemented in the Python C extension
- The reader transfer telemetry as needed
- Saving to disk has been implemented in a novel way
    - Empty .cfits or .npy files are created on disk by Python code
    - Files are queued up to ensure continuous operation
    - Memory regions are mmap'd by the C extension with the correct offsets
    - Data is copied directly to the file region
    - The .cfits extension is used to distinguish files with little endiannes



SRTC Services and Plugins

- Background processes for calibration and optimisation functions and loops
- Each function or loop is implemented as a Plugin
- Plugins are registered to a SRTC Service
- Uses runtime introspection to display functionality in a GUI

SRTC Service:

- Can register Plugins
- Holds parameters for all Plugins
- Initialises and Configures Plugins
- Execute Plugins
- Access Results
- Report Plugin Status
- Interface for Displays

- Not only for Soft Real Time Processes
- Can also be used to implement general background process functionality.
- e.g. for OCAM iPort Daemon and command sending

SRTC Plugin:

A python class that implements methods to run specific tasks:

- Configure: to update parameters
- Init: first time setup
- Setup: setup per run
- Acquire: get data
- Execute: main functionality
- Check: verify resilts
- Finalise: cleanup and store result
- Apply: optional send new data to RTC
- Result: return the last result

High Level:

- run: calls the above functions in order with option for calling Apply
- start: run every N seconds in a python thread, Setup is called once at beginning
- stop: cancel the thread

The Background Tasks for CaNaPy - SRTC

Each WFS
- OCAM frame-rate, exposure time, gain
- OCAM image calibrations, dark, flat, background
- WFS-DM misalignments
- NCPA
- WFS-modes matrix

Pyr-WFS
- Modulation rate and magnitude
- Modulation and exposure synchronisation
- Pyramid quadrant detector offset
- DM + UTTM interaction matrix
- DM + UTTM control matrix

SH-WFS
- SH sub-aperture positions
- Centroiding, thresholds, correlation
- DM + DTTM interaction matrix
- DM + DTTM control matrix

Telemetry needed
- Decimated WFS images
- Pyramid flux values (add quadrants)
- SH subaperture and image flux
- Wavefront gradients
- Wavefront modes
- Actuator values

Scoring cameras
- OCAM frame-rate, exposure time, gain
- OCAM image calibrations, dark, flat, background
- Image stacking?
- Image processing, FWHM, SR, centroid

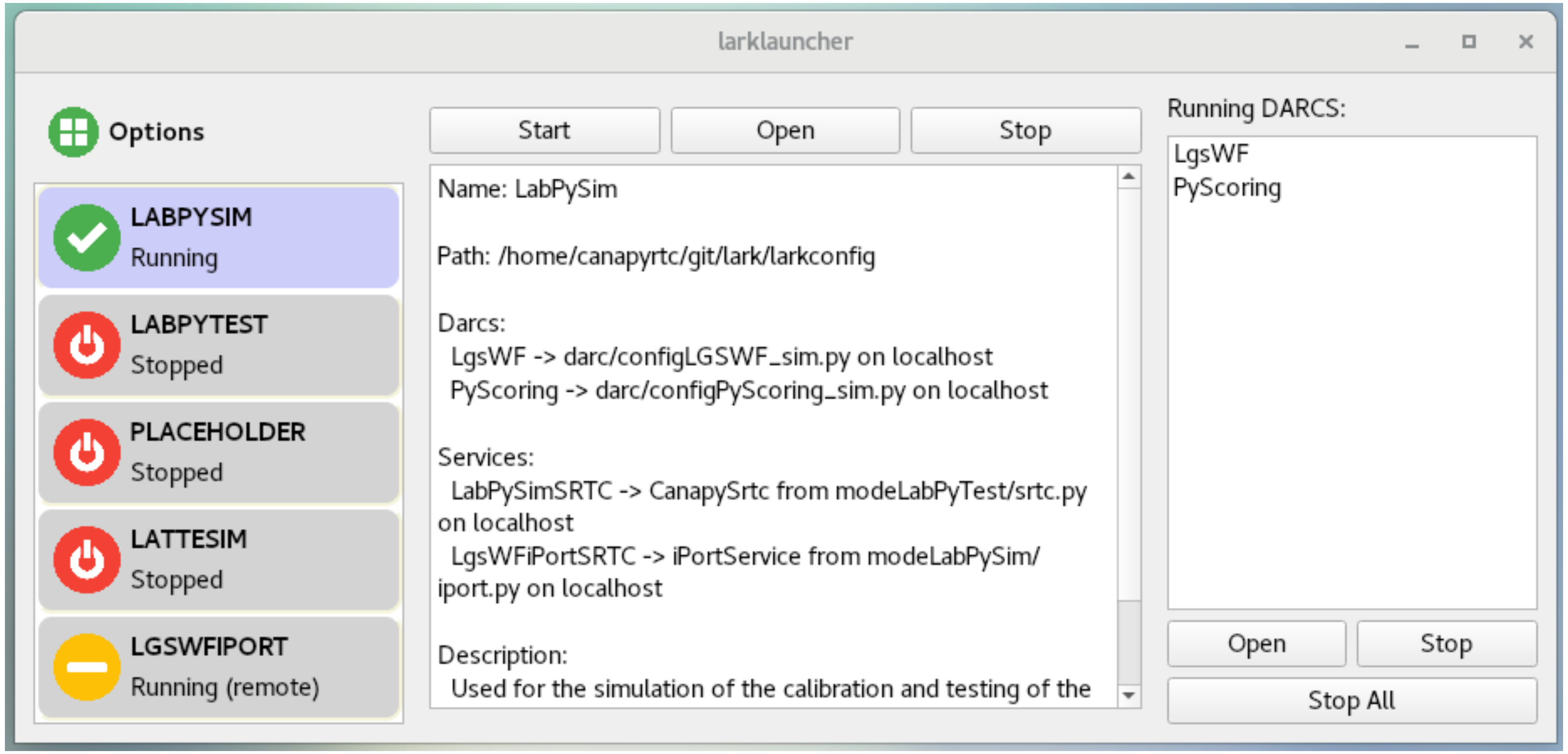
DM, UTTM, DTTM
- Modes-DM control matrix
- Actuator scaling, clipping

Reconstruction and control
- Gains
- Noise covariance matrices

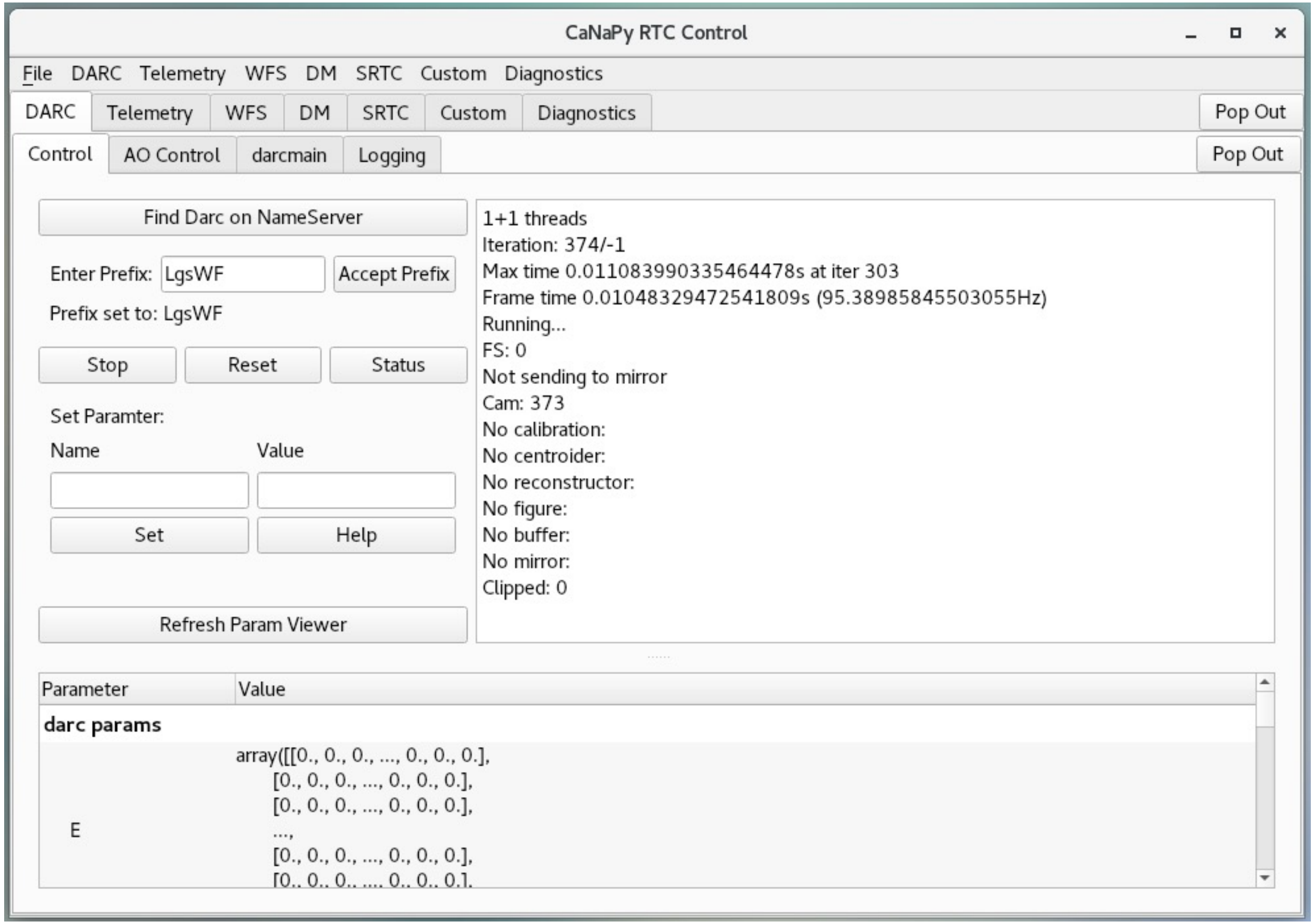
Key
- Continuously updated based on telemetry
- Calibrated once and used on every frame
- Telemetry needed for calibrations on every frame
- Telemetry used for visualisation only

Lark Launcher

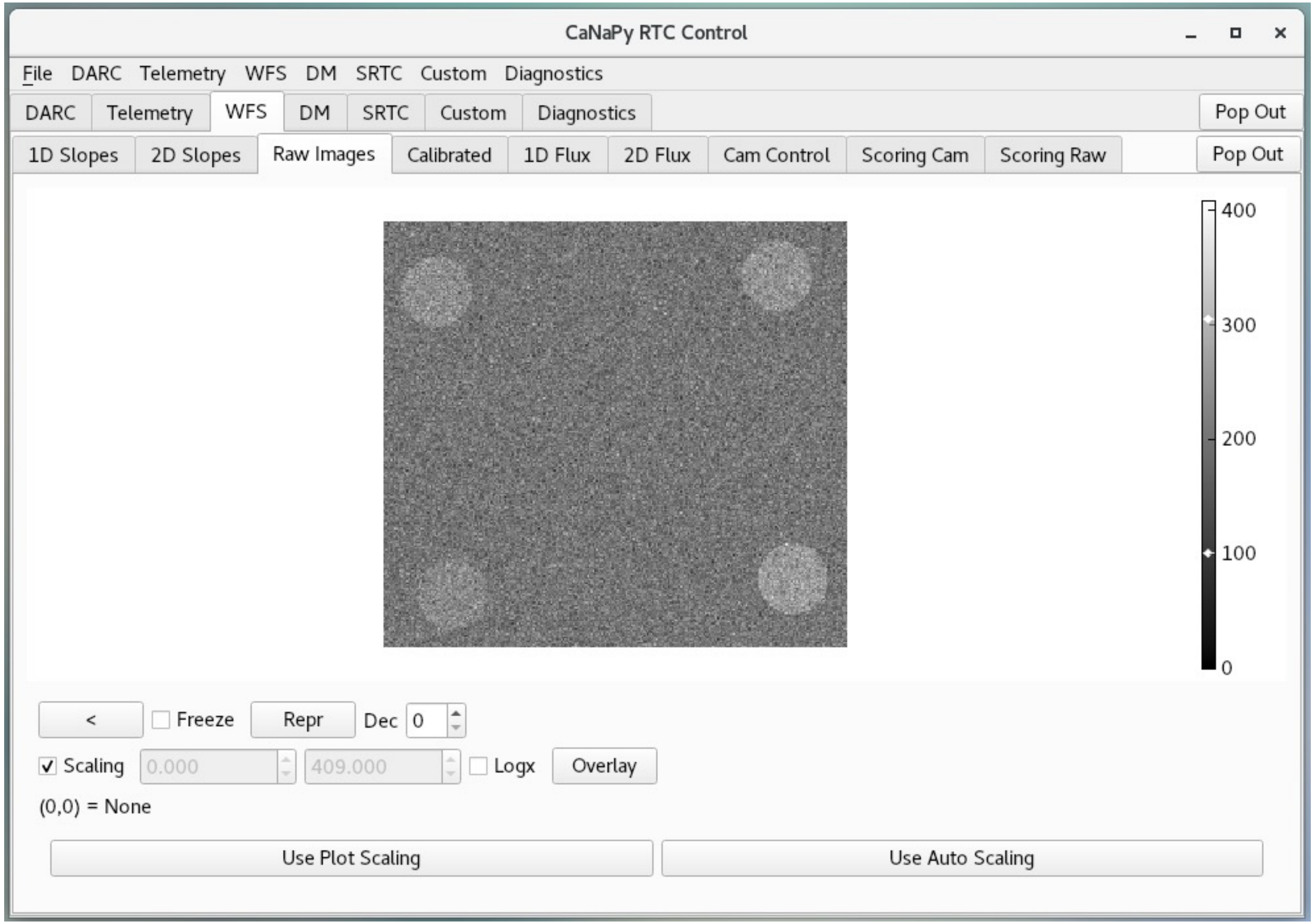
- It finds modes in the config directory and loads the information
- Each mode has 3 basic commands, start, open and stop
- Open is used to open a mode specific display
- The running DARCS are displayed, and the basic Lark Plot GUI can be opened for each
- The options menu allows selecting a config directory and to reset the Lark Daemon to kill all running processes and start fresh



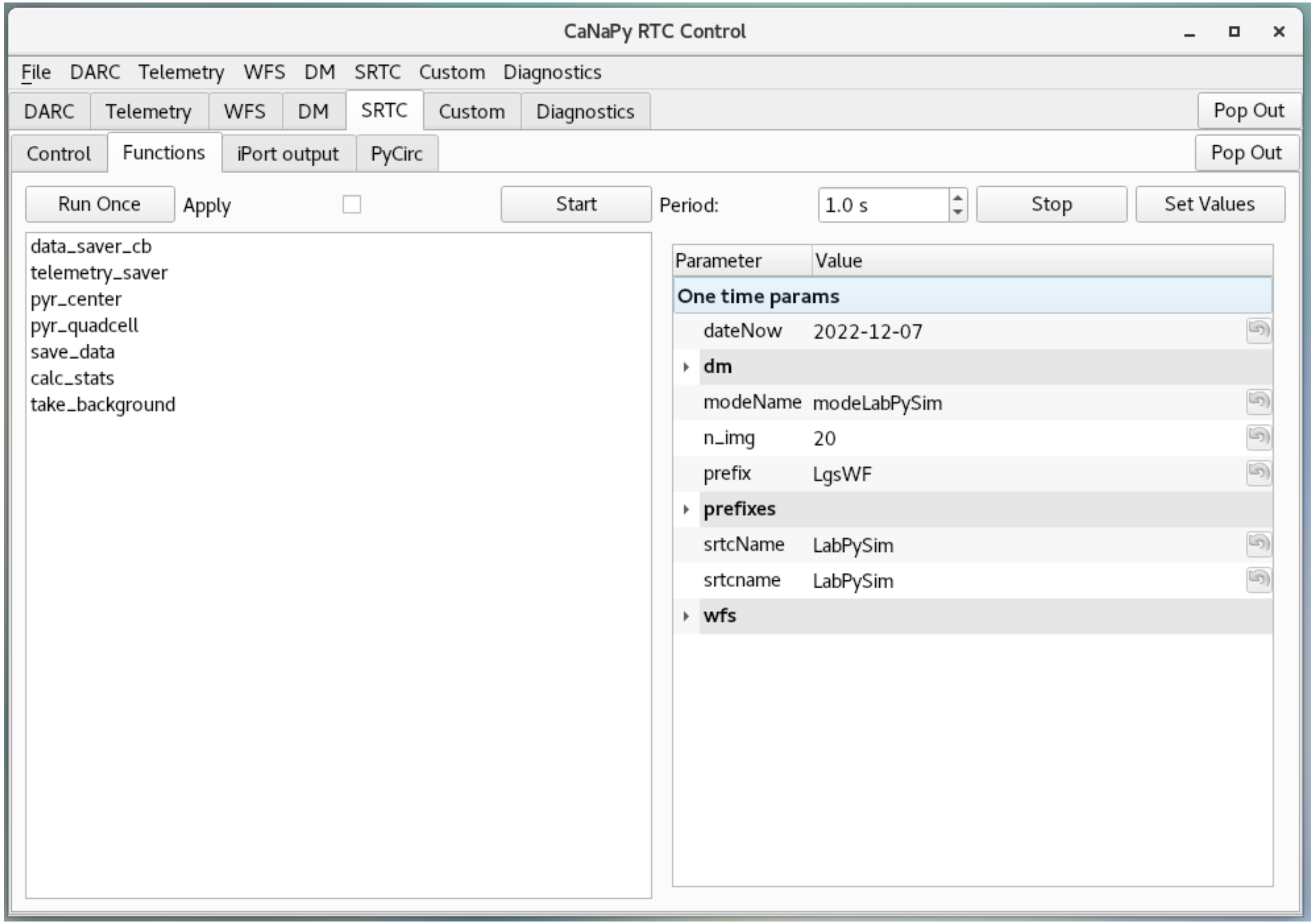
Lark Custom Displays



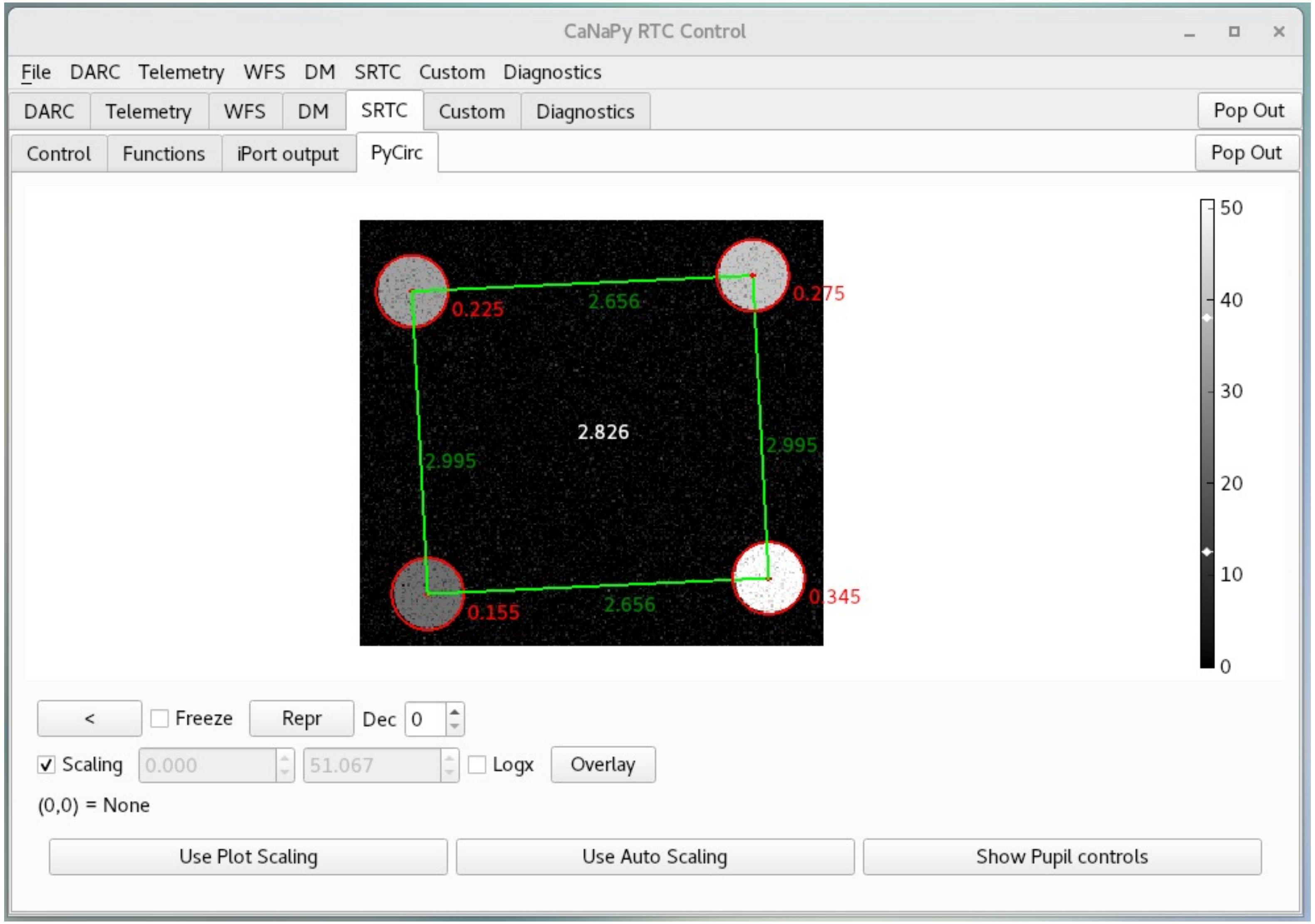
Lark Custom Displays



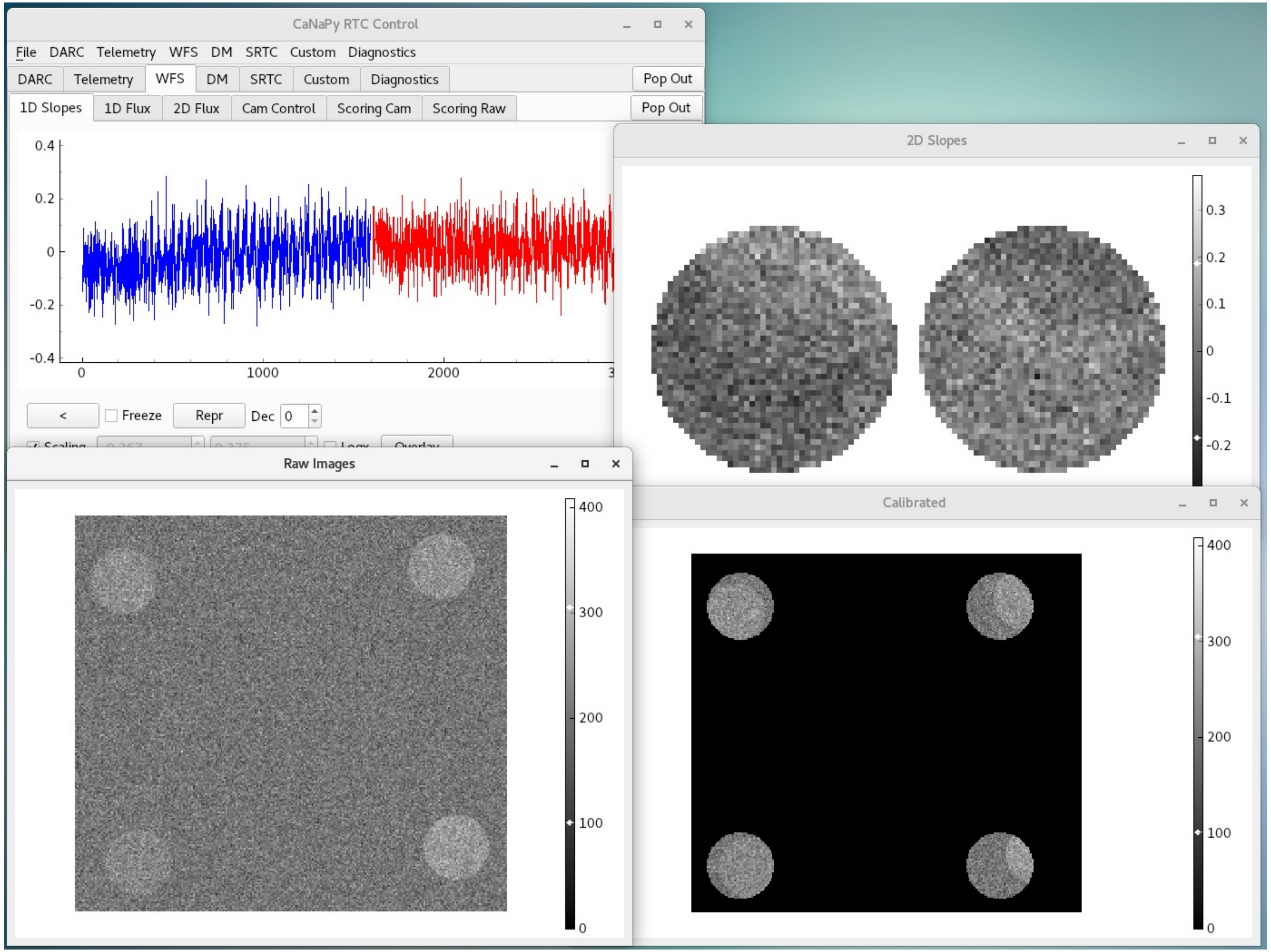
Lark Custom Displays



Lark Custom Displays



Displaying Multiple Plots



Current Status of CaNaPy

- Assembled and tested in laboratory conditions in engineering mode
  - Closed loop operation achieved with the Pyramid WFS and the ALPAO DM using a telescope simulator installed in the lab
- Currently being installed at the OGS on Tenerife
- September/October commissioning run delayed (Tenerife forest fires)
- Phase A commissioning planned for November 18-26, 2023
- Initial plan is to commission the NGS SH-WFS and the Laser uplink
- LGS Py-WFS closed loop planned for Spring 2024
- RTC development and testing will continue alongside the commissioning of CaNaPy/ALASCA

Questions?

- *CaNaPy is a test facility also for future further experiments with the community and with ESA*
- *CaNaPy is coordinated by ESO and carried out jointly with AO expert groups from three ESO member states: Durham University (UK), INAF (OAA and OAR - Italy), IAC (ES); and in synergy with ESA, within the frame of the ESO-ESA collaboration and implementation agreements.*