Synchronome Project Design Documentation

Samuel Gfrörer

This project is an exercise in realtime embedded engineering. An embedded computer connected to a camera periodically takes pictures of a ticking clock and saves them to persistent memory (Flash). The program outputs one image each second, each showing a still image of the seconds hand in one stationary position. (or one image each 1/10th of a second if the clock displays 1/10ths of a second). In order to achieve that, the process of taking/picking pictures needs to be carefully synchronized with the external clock.

Contents

1 Synchronome Project - Requirements [1]

1.1 Minimal Objectives

Input: Images acquired from stationary digital camera facing an analog clock with hour, minute and second hands (or a stop watch with 1/10th hand *)

Output: selection of camera frames for which the following conditions hold:

- Frames encode a time-lapse of the clock at 1Hz (/10Hz *)
- Bijection (1:1 relation) between distinguishable clock hand positions and selected frames
  - Every frame shows a unique state of hands on the analog clock
  - There is a frame for every distinguishable stationary position of clock hands
- Skipped, blurred or repeated frames are considered an error
- All frames not-blurry (all hands stationary)
- 1800+1 frames (frame 0…1800) that means at 1Hz runtime = 1800s = 30min (at 10Hz: runtime 180s = 3min *)
- Resolution >= 320x240
- Frames stored to disk, where
  - Every frame stored as separate file
  - Every frame stored as Portable Anymap (.ppm P3/P6 or .pgm P2/P5)
  - Every frame contains a timestamp in the file header or in the image
  - Every frame contains system info from `uname -a` in the file header or in the image

Additional objective: compute and plot average jitter and drift

1.2 Target Objectives

One or more of the following features:

- Program saves frames at 10Hz and runs stable (glitches are allowed)
- Image processing or frame compression
- Continuous streaming or automatic periodic download of frames, so that the service never runs out of flash memory
- another image processing feature that can be toggled during runtime

[1] This is a summary of the documents in <./doc/resources> given as requirements and grading criteria of the course.

1.3 Stretch Goals (Full Credit)

- Program acquires frames at 20Hz
- Program outputs frames at 1Hz or 10Hz without glitches and with continuous download
- Program runs glitch-free with an an external digital clock that updates 1/10th of a second
- Any other frame-by-frame image processing at 1Hz and 10Hz
- Switching single or mutliple features on/off during runtime does not create stability issues (jitter, drift, monotonicity)

2 Synchronome Project - System Design

2.1 Platform Specification

Platform:

- OS: linux
- Board: Raspberry Pi Model 4B
- Camera: C270 HD WEBCAM

2.2 Implementation Details

- Implementation in C
- Frame acquisition from the camera uses the Video for Linux ver.2 API (V4L2) and uses the streaming I/O method aka *memory mapping*
- Frames should *never* be copied but shared/accessed via pointers for performance reasons

2.3 Design Overview

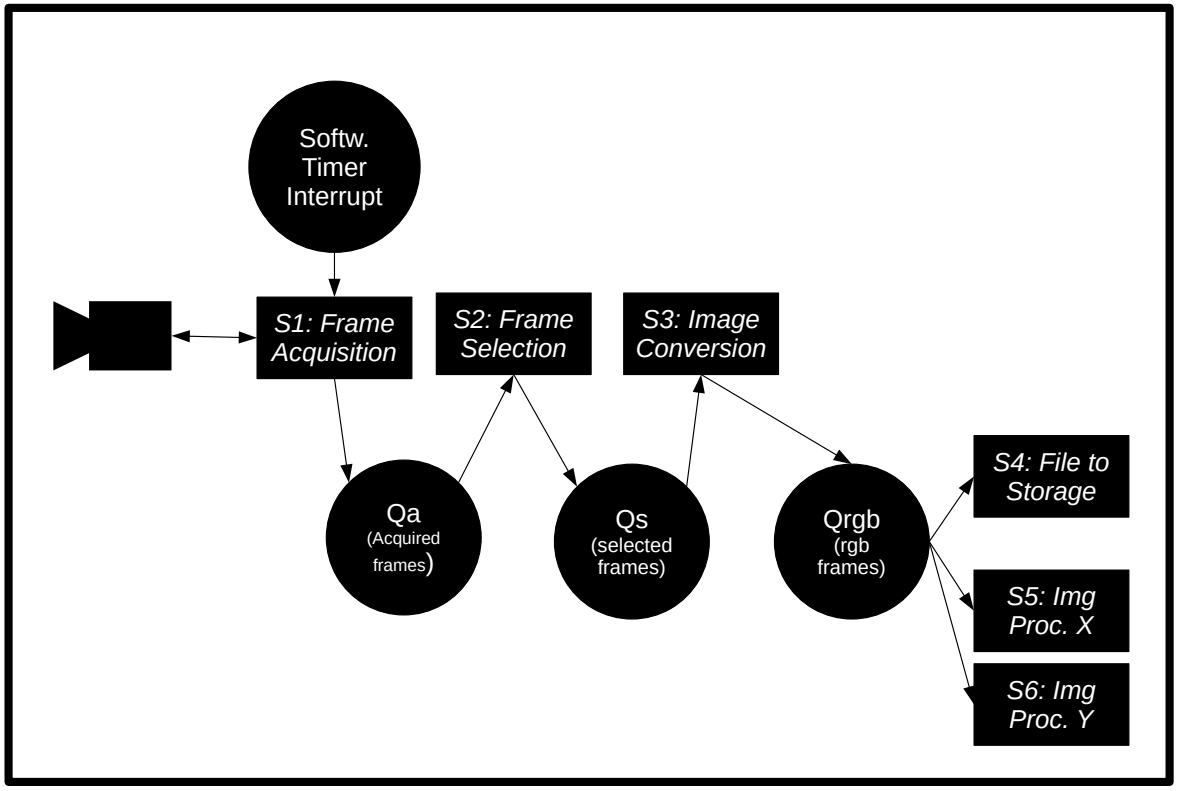


Figure 1: System Components Overview

2.4 Resources

Queues are used to decouple read and write access. These are implemented as ringbuffers in user space. A read operation blocks on a semaphore, until the next element is available.

- Camera: repeatedly sampled to acquire frames
- Select State:
- Qa: buffer of frames acquired from the camera with timestamps (from the linux system clock) attached
- Qs: buffer of selected frames
- Qrgb: buffer of selected frames in rgb format

2.5 Services

Definitions:

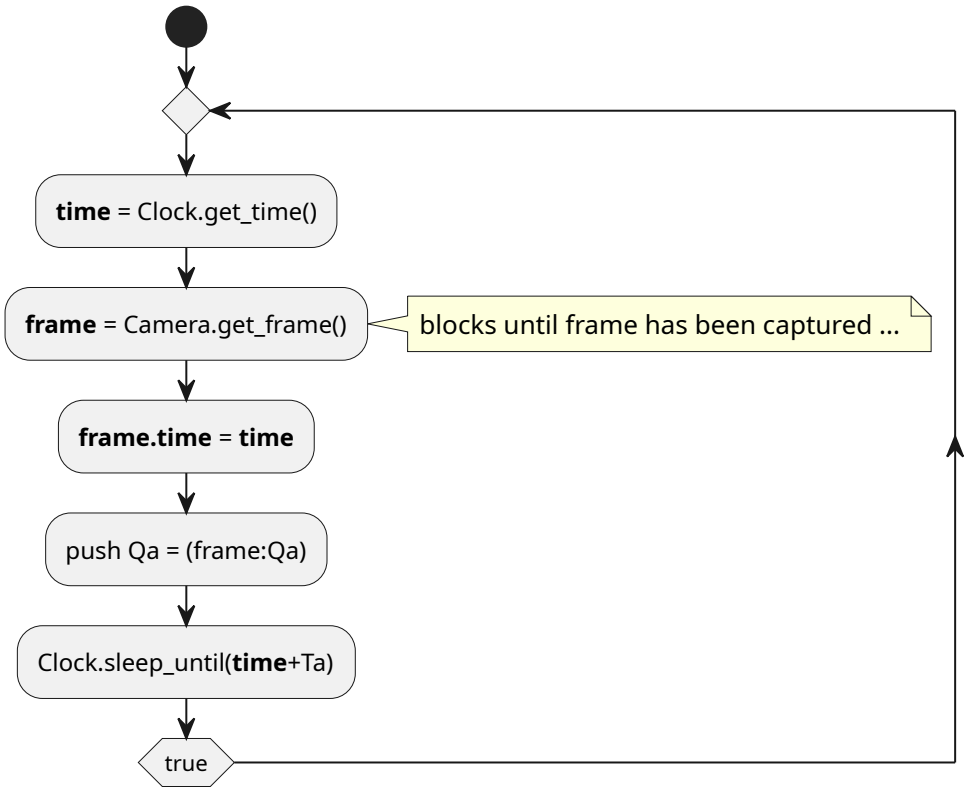
- Tc := “tick interval of external clock” (= 1s or 0.1s for target/stretch goal)
- Ta := “acquisition interval” should be an integer multiple >3 of the external clock interval. Let `s_res` be the factor between Ta and Tc:

  s_res * Ta ~ Tc, s_res >= 3, s_res Integer
- We can then define the “internal tick interval” Ti:

  Ti = s_res * Ta

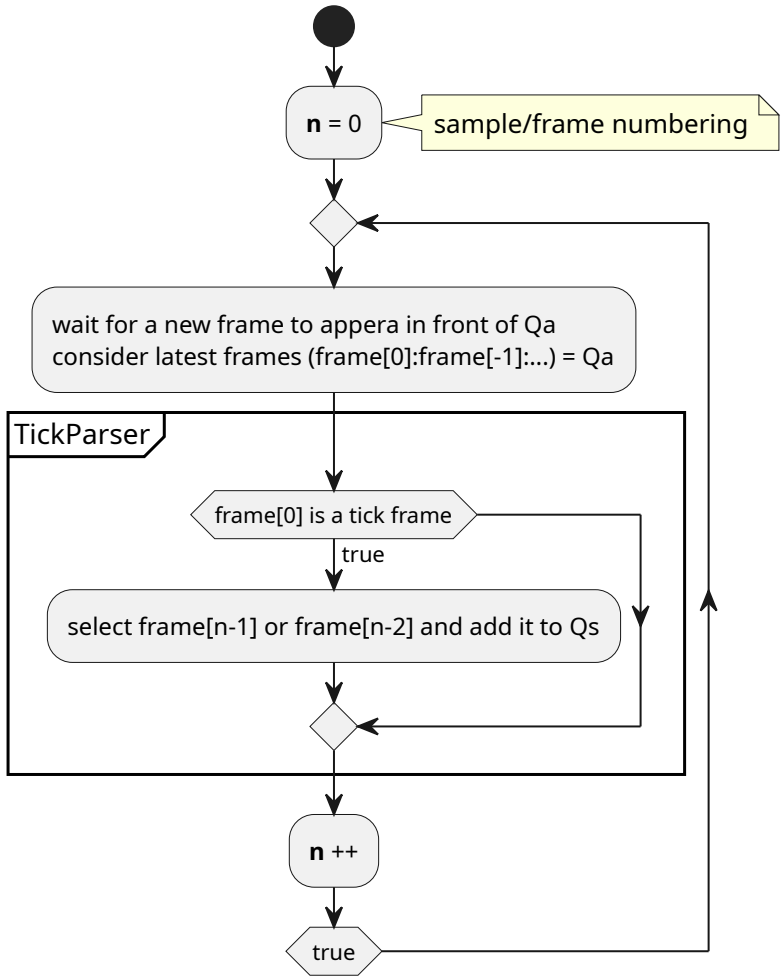
2.5.1 S1: Frame Acquisition

Idea: aquire frames from camera and attach time stamps

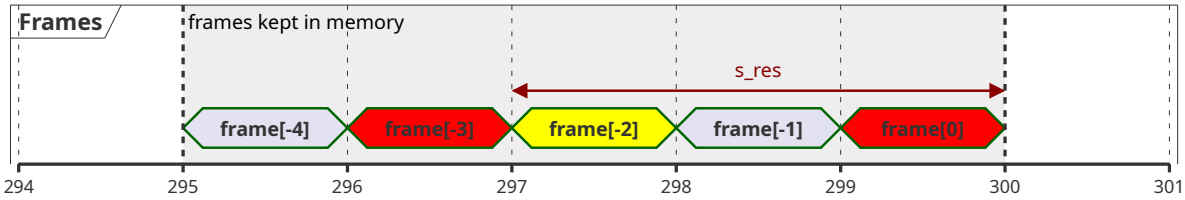


2.5.2 S2: Frame Selection (& Tick Recognition)

S2 is started whenever there is a new incoming frame in Qa. Qa shall then contain the most recent frames up to a number > s_res (older frames will be released):



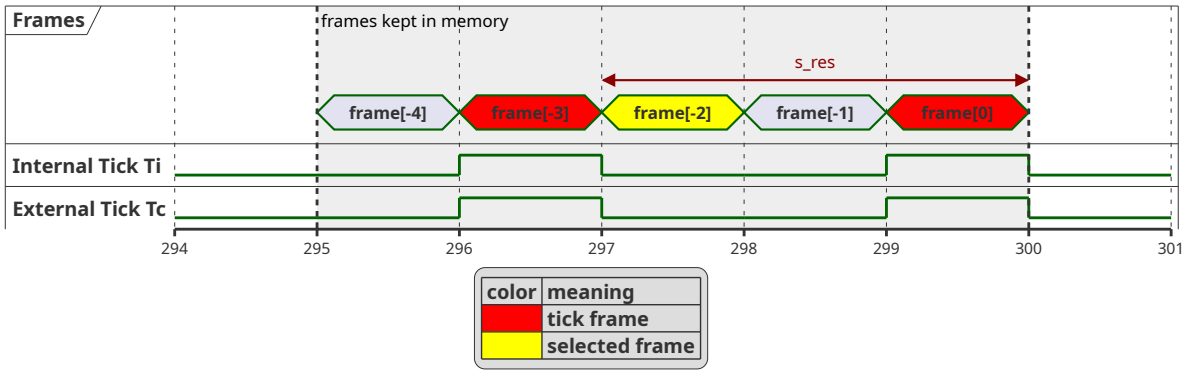
Therefore each cycle, the service can inspect multiple captured frames from the past:



Whether frame[0] should be considered a tick frame is to be discussed:

2.5.2.1 Case a) S2 in Sync / Stubborn Mode

If internal and external clocks are in sync, the situation is as follows:



In this case, the system could simply run on the internal clock. There would be no need to detect external ticks. It would increase the counter n for every incoming frame and assume every frame where the counter is a multiple of s_res to be a tick frame. Whenever a new tick frame arrives, frame[-2] (the frame after the previous tick) would be selected and send to Qs.

In reality, the situation is more complex.

2.5.2.2 Tick Detection & Drift Correction

Assumptions:

- The internal clock Ti may slowly drift against external clock Tc
- Drifting happens sufficiently slow, such that there is always a period of multiple samples, where the internal clock is only 1 sample ahead or behind the external clock

Idea:

- detect ticks by calculating the sum of pixel-wise difference between adjacent frames
- match every detected external tick frame with a corresponding internal tick frame (at multiples of s_res)
- recognize drift by reacting to multiple detected ticks being earlier/later than the internal tick
- counteract drift by adjusting the sample number n (+1 or -1)

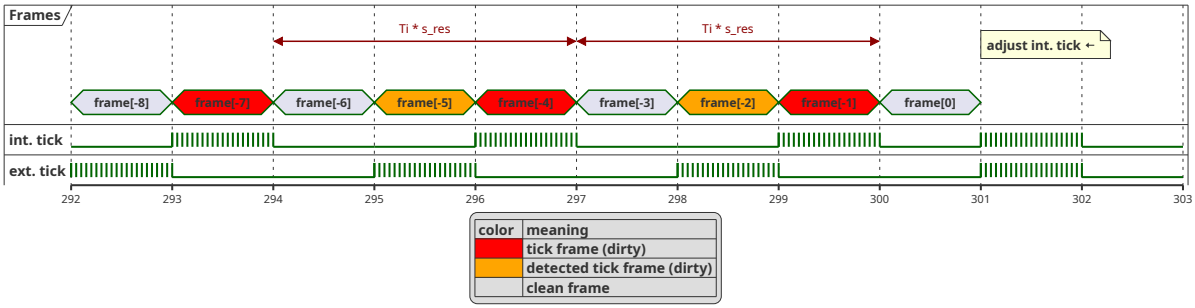
In this more advanced version of the TickParser, we will delay tick selection until 1 frame after the latest internal tick (301 instead of 300, in the diagrams).

2.5.2.2.1 Case b): int. tick behind ext. tick:

if the TickParser has registered to be 1 sample behind the external tick:

- tick selection should lean “left” (as in stubborn mode)

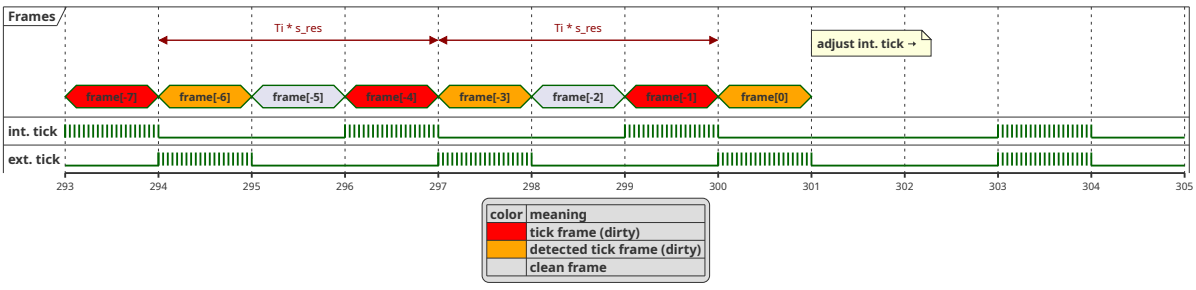
- the internal clock has to be adjusted “left”, that means n −−
- to be robust against occasional jitter or tick detection errors, adjustment should only happen after the drift has been confirmed some nr of samples



internal clock Ti **behind** Tc

2.5.2.2.2 Case c) int. tick ahead of ext. tick: Ff the TickParser has registered to be 1 sample behind the external tick:

- tick selection should lean “right”, to avoid selecting a “dirty” frame during a tick
- the internal clock has to be adjusted “right”, that means n ++
- to be robust against occasional jitter or tick detection errors, adjustment should only happen after the drift has been confirmed some nr of samples



internal clock Ti **ahead of** Tc

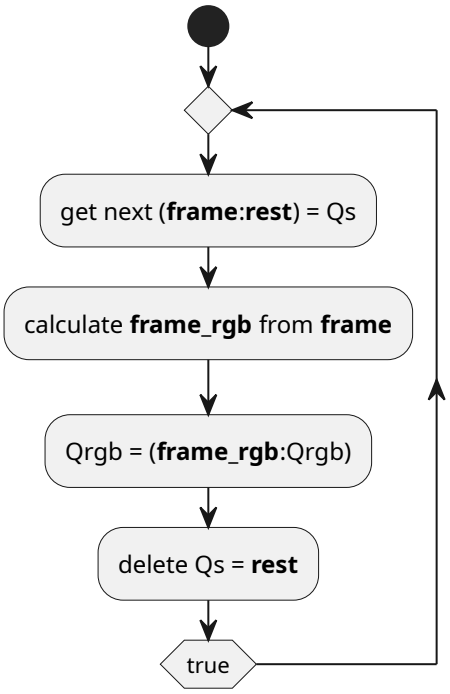
2.5.2.3 Robustness to Tick Detection Errors Tick detection may be unreliable for short periods (additional ticks, missing ticks).

This is recognized by the TickParser, whenever the 1-to-1 correspondence of internal and external ticks cannot be maintained. The system then goes into stubborn mode a), ignores detected ticks and frame selection is then based on the internal ticks. As soon the tick detection becomes reliable, the system will again take into account detected ticks for synchronization.

2.5.3 S3: Image Conversion

Idea:

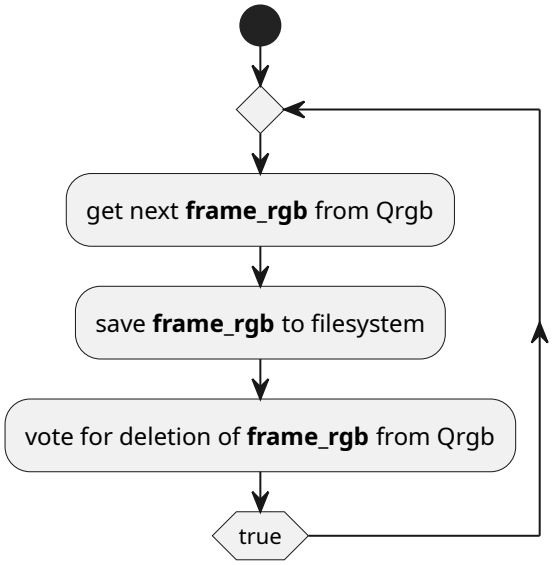
- Keep it simple



2.5.4 S4: File To Storage & S5,...: Other Image Processing

Idea:

- Keep it simple



Other image processing tasks S5,… have a similar control flow with a different action instead.

Notes:

- deletion of frames from Qrgb is postponed until all Services (S4,S5,…) have agreed.

3 Synchronome Project - Timing Analyis

Definitions: (same as in previous chapters)

- Tc := “tick interval of external clock” (= 1s or 0.1s for target/stretch goal)
- Ta := “acquisition interval” should be an integer multiple >3 of the external clock interval. Let `s_res` be the factor between Ta and Tc:

  s_res * Ta ~ Tc, s_res >= 3, s_res Integer

The system consists of the following services:

- S1: Frame Acquisition
- S2: Tick Recognition
- S3: Image Conversion
- S4: Files to Storage
- S5, …: Other Image Processing Tasks

3.1 Real-Time Requirements

For the following discussion, we assume that for each service the deadline is equal to its period: D = T. S1, S2 and S3 are considered critical Services. The deadline is determined by the functional requirements: In order to guarantee at least 1 frames between ticks and to fulfill the additional target requirement to measure and control drift, Ta should be less than half of a clock tick, preferrably Ta <= 1/3 * Tc. S4 and others are not critical for synchronization issues and may therefore run as best effort services. In order to prevent data overflow, the period is required to be less than Tc *on average*.

In summary, we get:

Deadline/Period	S1	S2	S3	S4
T	Ta	Ta	Tc	Tc

3.2 Real-Time Analyis and Design

In order to more specifically analyse the timing requirements and give concrete constraints, we need to specify a mapping of services to cpu cores. The following table shows the chosen mapping in combination with the requirements.

Deadline/Period	S1	S2	S3	S4
T	Ta	Ta	Tc	Tc
CPU	2	3	3	4

3.2.1 Feasibility

3.2.1.1 S1 and S4 S1 and S4 occupy one core on their own. Therefore they must fulfill the trivial constraint U < 1.

3.2.1.2 S2 and S3 S2 and S3 share core 3. According to the Liu & Layland paper [1], feasibility is guaranteed for rate-monotonic (shortest-deadline-first) policy, as long as the total utilization below the least upper bound. For 2 processes, this criteria evaluates to:

U_{total} < LUB = 2*(2^(1/2) - 1) = 0.83 = 83%

3.3 Concrete Measurements

The updated timing analysis is based on the program tracing runtime information such as start/end/runtime information via syslog. The following diagrams show the data collected from logfiles for the case Tc = (1/10)s & Ta = (1/30)s. If feasibility can be shown, the requirements for the less challenging case of Tc = 1s, Ta = (1/3)s shall be trivially fulfilled.

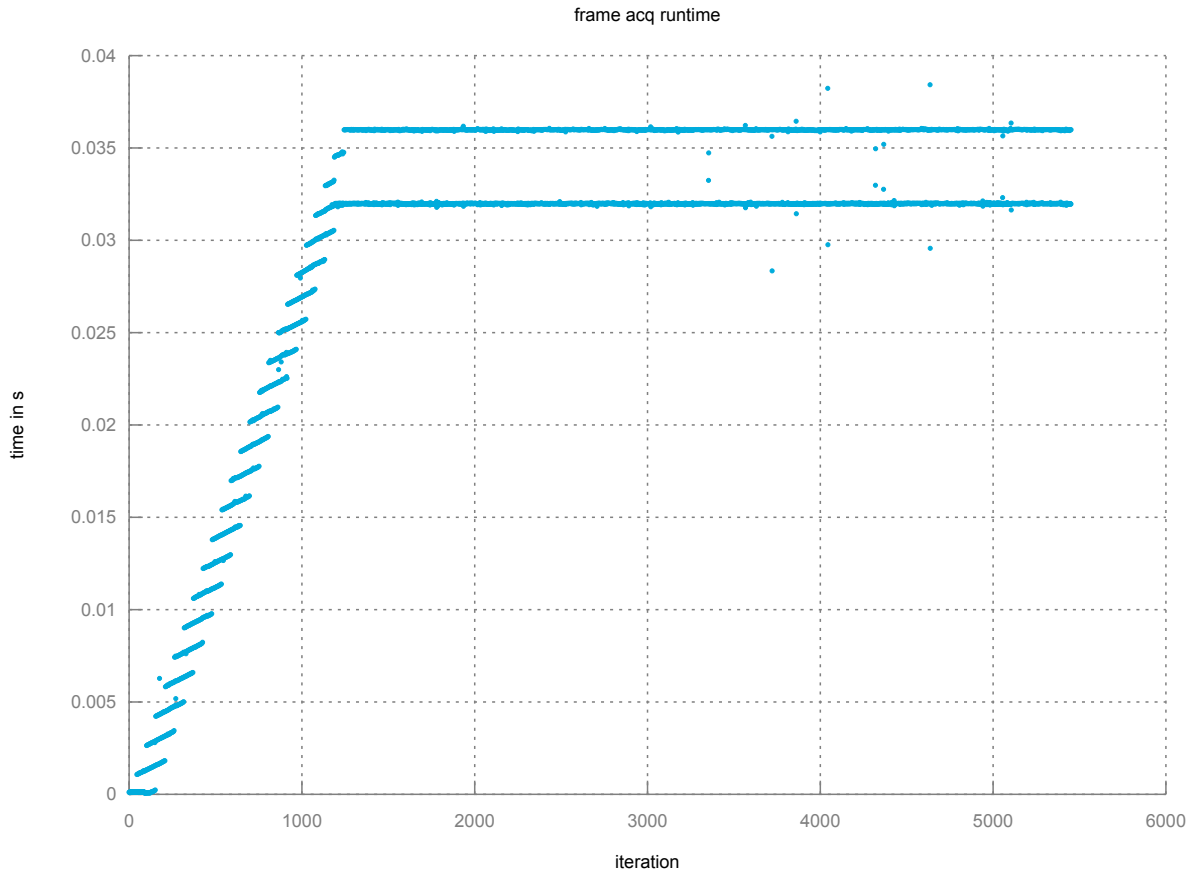


Figure 2: frame_acq runtime

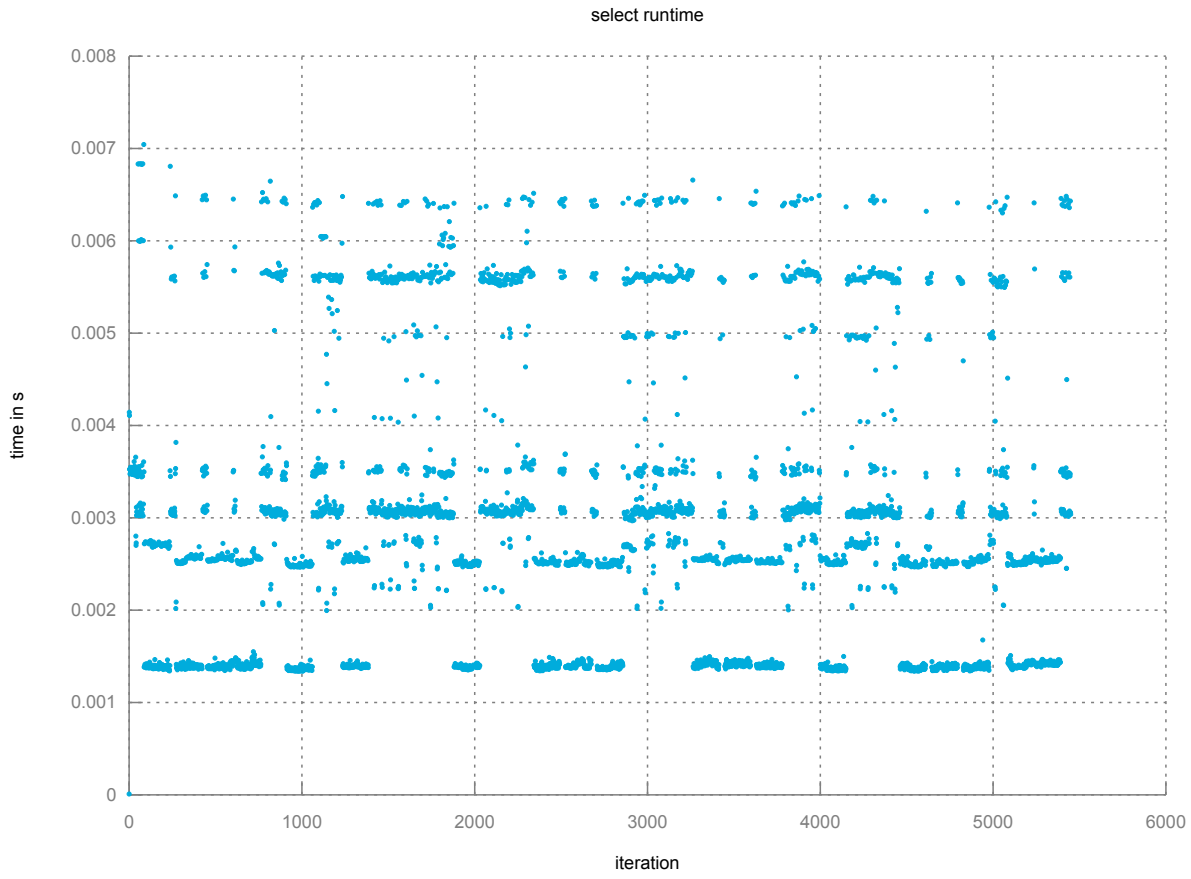


Figure 3: select runtime

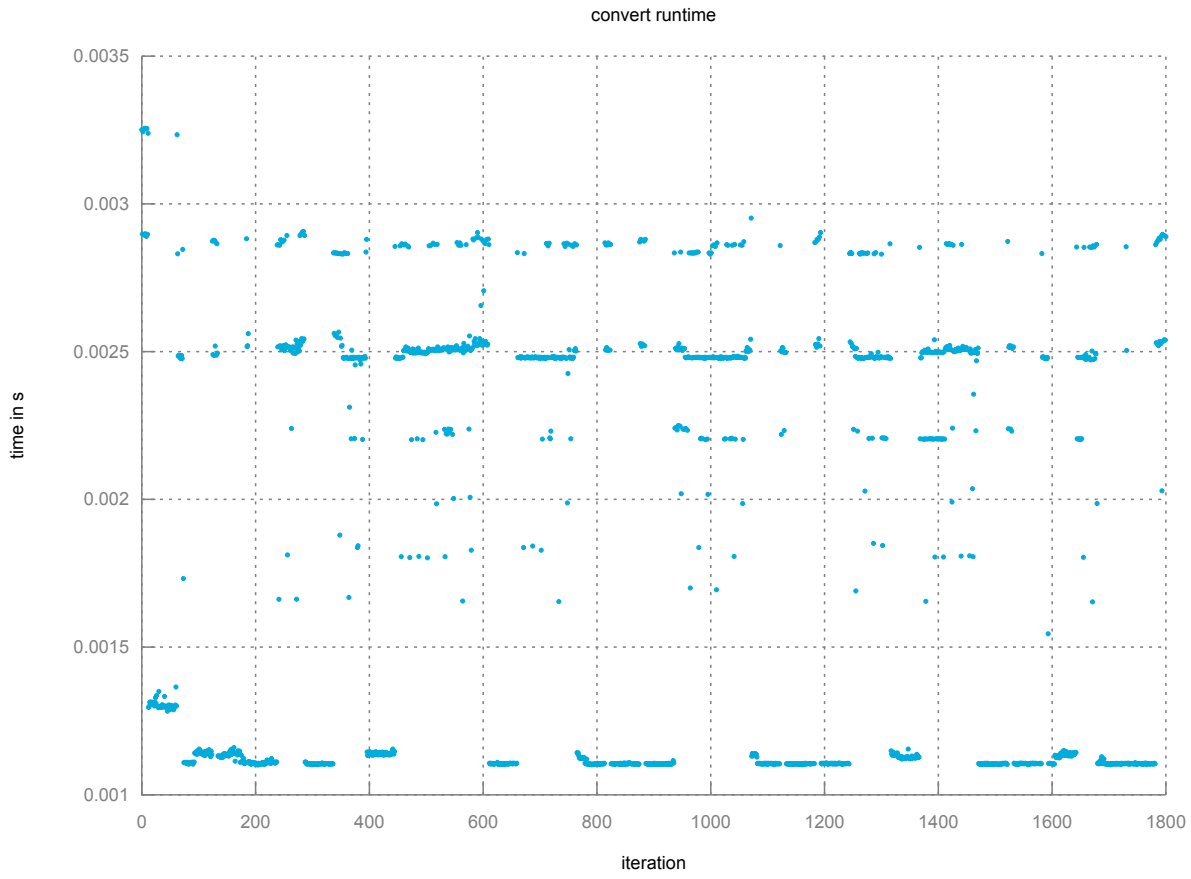


Figure 4: convert runtime

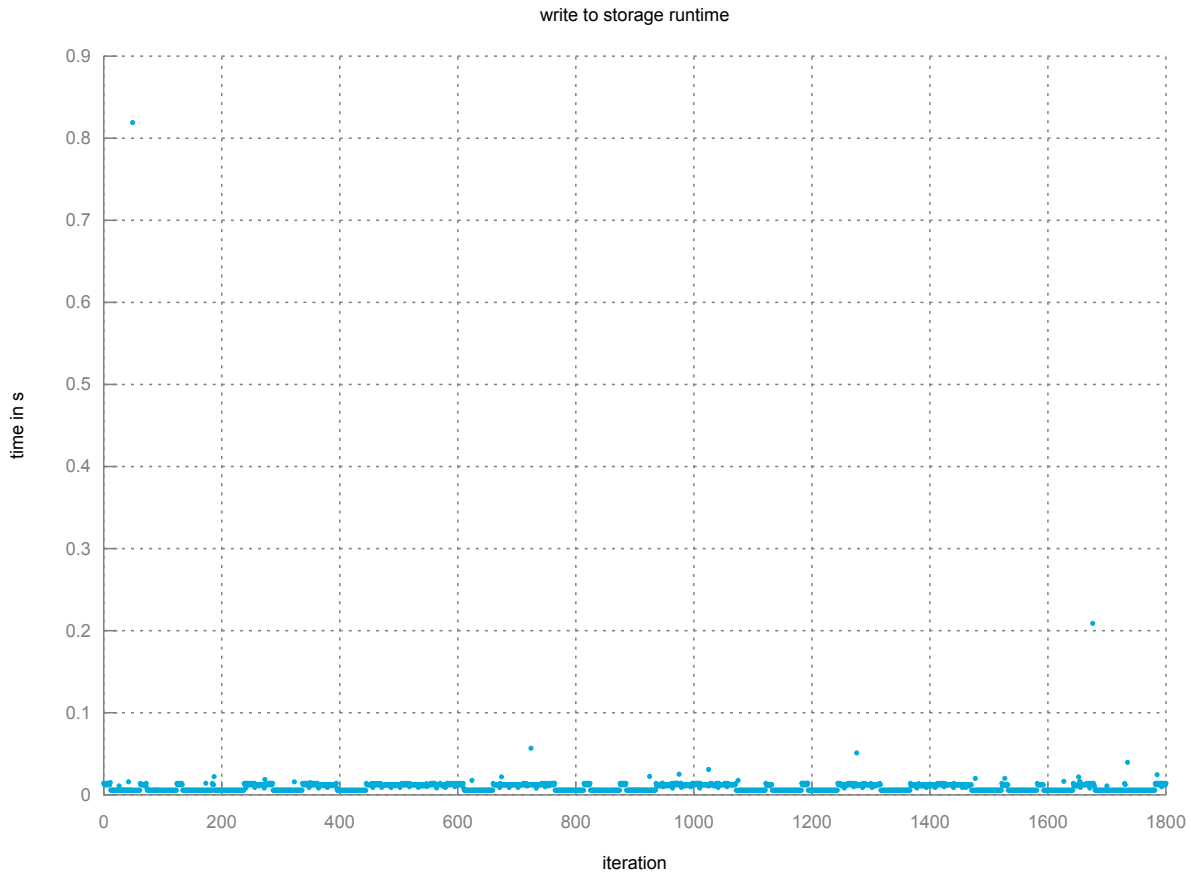


Figure 5: write_to_storage runtime

Diskussion

- frame_acq: The execution time of frame acquisition solely depends on the minimum supported acquisition interval of the camera (the contribution to the runtime of additional code is minimal and should be neglegible). The minimum supported sample interval listed by the V4L2 driver is 1/30. The driver accepts values given as fraction of integers. As can be seen in the diagram, the driver chooses a slightly unequal distribution. This is no problem for this application, as long as the average interval is as requested and the variance is sufficiently slow.
- select, convert: From the diagrams, we can assume that the WCET for both services is < 0.01. Under this assumption, following the discussion above, this is always feasable for Tc = 1s and Tc = 0.1s:
  - Case Ta = 1/3:
    * U2 = 0.01s / (1/3)s = 3%

    * U3 = 0.01s / 1s = 1%
    * Utilization U = U2 + U3 = 4%
  - Case Ta = 1/30:
    * U2 = 0.01s / (1/30)s = 30%
    * U3 = 0.01s / (1/10)s = 10%
    * Utilization U = U2 + U3 = 40%

Timing diagrams for these cases (units are 1s/300):

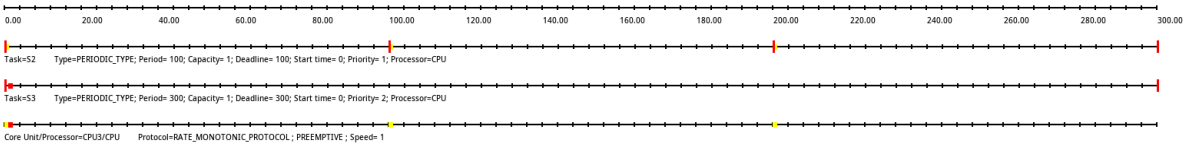


Figure 6: scheduling diagram Ta=1/3, time in 1s/300

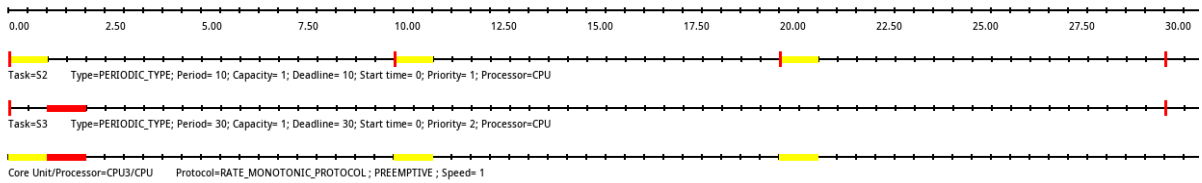


Figure 7: scheduling diagram Ta=1/30, time in 1s/300

- write_to_storage: This service is run as best effort task. The services are decoupled via queues, such that their runtime does not depend on each other. The average runtime must still be be below Tc to prevent the queues from overflowing. The occasional spikes are a result of buffered file I/O. Linux buffers write operations in RAM and occasionaly synchronizes the collected data to flash, which is remarkably slow. This demands the queue lenghts limit to be sufficiently high as well as other tasks involving file system access such as logging to be decoupled as well in order to prevent unintended race conditions for the file system.

3.4 Enforcing Deadlines / Hard Real-Time

In order to proof hard real-time capabilities, a missed deadline in S1, S2 and S3 is considered fatal and the program will fail with a non-zero exit code.

3.5 Drift

The following diagrams visualize drift by plotting the fractional part of start times for each service. The diagram for the frame selection service shows, that the system indeed successfully detects external ticks and counteracts them by adjusting the frame numbering.

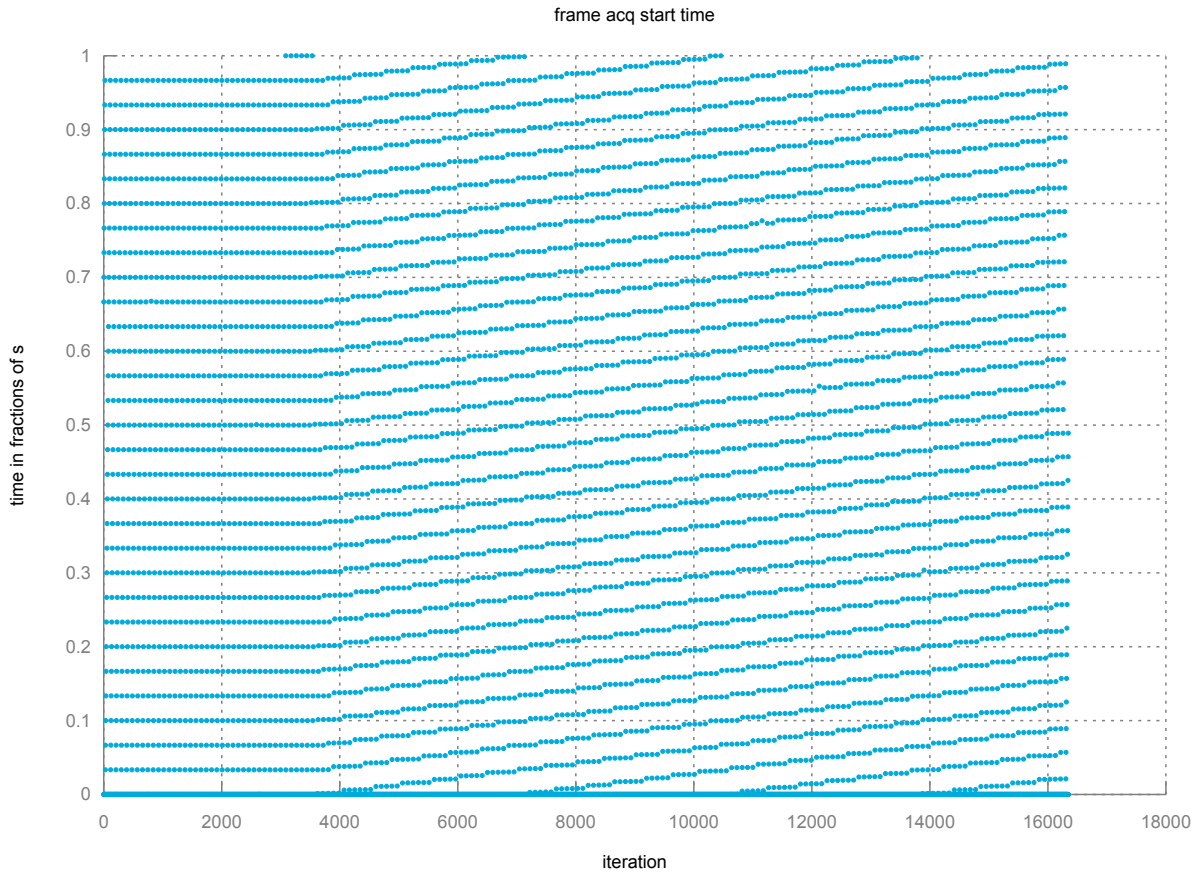


Figure 8: frame_acq start time mod 1s

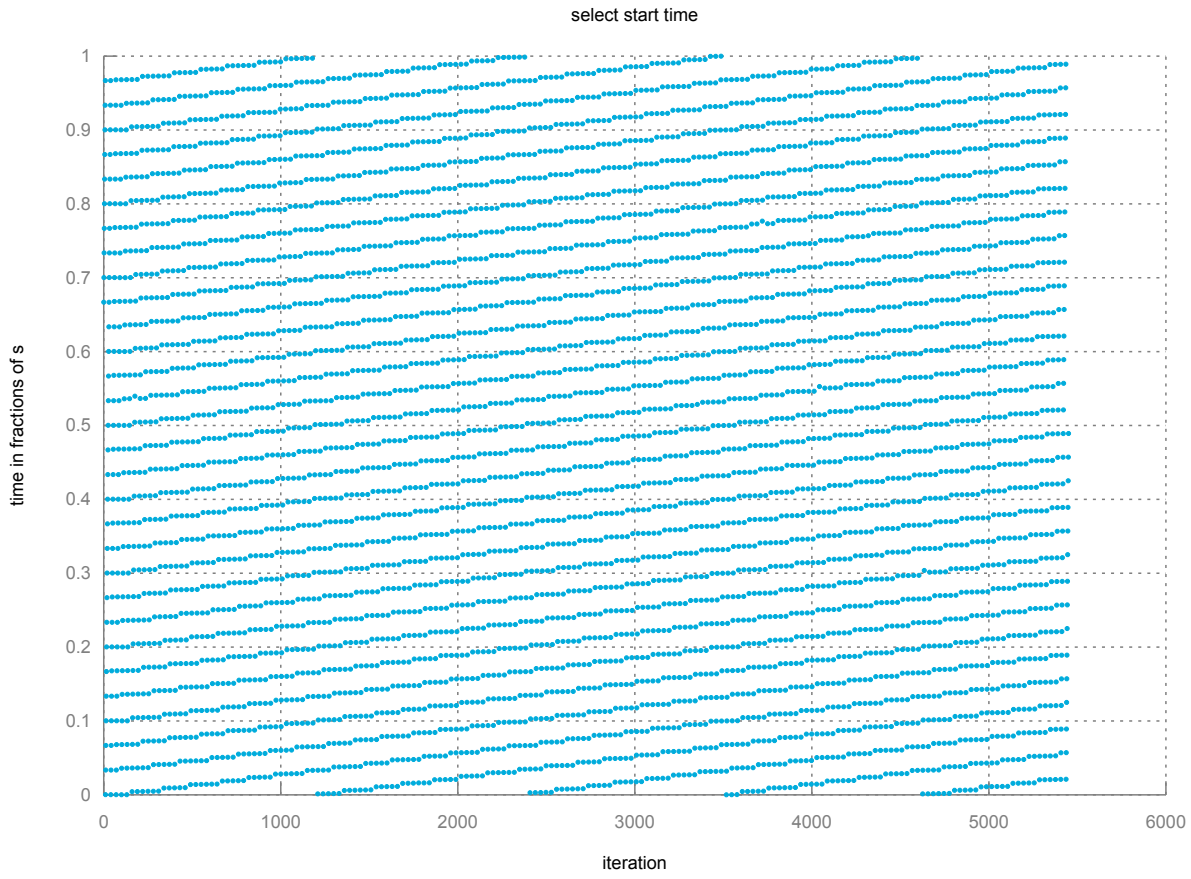


Figure 9: select start time mod 1s

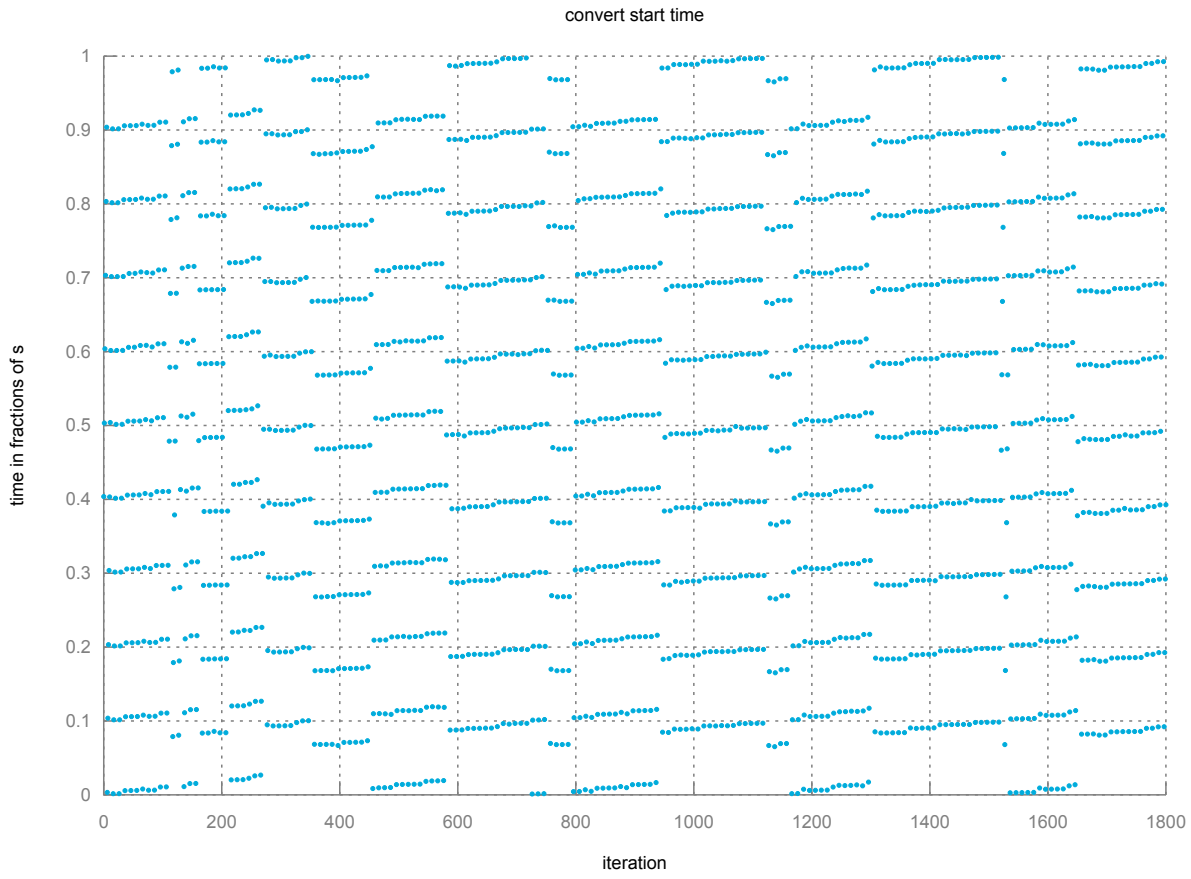


Figure 10: convert start time mod 1s

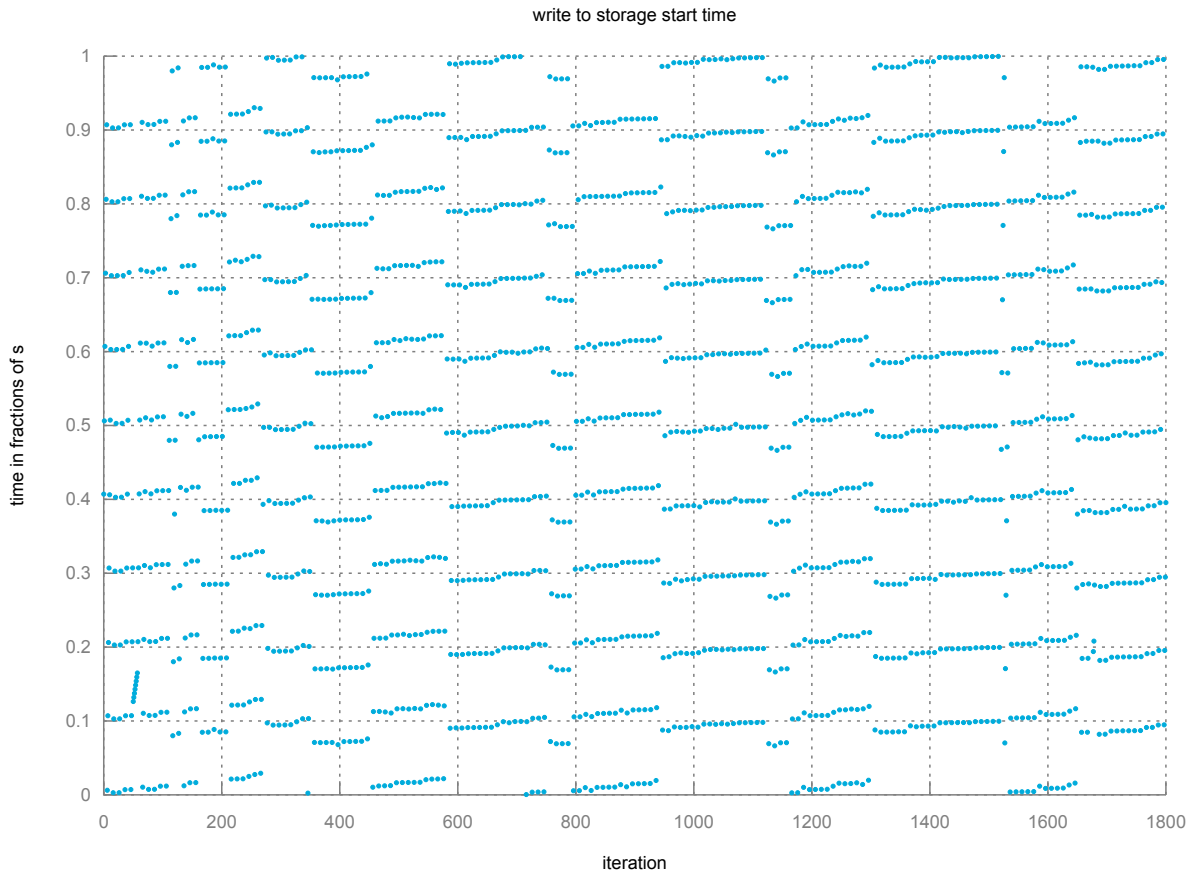


Figure 11: write_to_storage start time mod 1s

4 Preliminary Design & Analysis (for Reference)

Running the `./scripts/run_statistics` on the destination platform prints information about the platform and tests for the worst-case and average runtime for relevant cpu-intensive parts of the services described. These can be used to give a first estimation of WCET without doing an actual test run of synchronome program. Before the program was in a mature state, services including runtime tracing via syslog implemented, the preliminary system design and analysis was based on these estimations. These measurements are given here for completeness:

service: task	320x240	640x480	1280x960
S1: camera_get_frame	max: 0.068028s avg: 0.066803s	max: 0.067995s avg: 0.066800s	max: 0.143959s avg: 0.141613s

service: task	320x240	640x480	1280x960
S2: image_diff	max: 0.001505s avg: 0.001478s	max: 0.006652s avg: 0.006592s	max: 0.041315s avg: 0.041092s
S3: image_convert_to_rgb	max: 0.001124s avg: 0.001094s	max: 0.015134s avg: 0.005367s	max: 0.031482s avg: 0.018423s
S4: image_save_ppm	max: 0.017130s avg: 0.010344s	max: 0.117209s avg: 0.045070s	max: 0.239186s avg: 0.207855s

For the minimal objectives Tc = 1s and taking into account the measured WECT for 320x240, we get:

Parameters	S1	S2	S3	S4
T in s	1/3	1/3	1	1
C in s	0.068	0.002	0.001	0.017
U in %	20%	0.45%	0.11%	1.71%

For the target/stretch goals Tc = 0.1s, we get:

Parameters	S1	S2	S3	S4
T in s	1/30	1/30	0.1	0.1
C in s	0.068	0.002	0.001	0.017
U in %	204.08%	4.52%	1.12%	17.13%

The obvious issue of S1 exceeding the maximum period of 1/3 seems to be rooted is rooted in a flawed runtime measuring strategy and doesn't appear in practice (as has been demonstrated above).

4.1 References

- [1] Liu, C. L.; Layland, J. (1973), "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, 20 (1): 46–61, CiteSeerX 10.1.1.36.8216, doi:10.1145/321738.321743

5 Synchronome Project - Conclusion

The Program has successfully been tested running on the specified platform (Linux + Raspberry Pi Model 4B + C270 HD Webcam) at a resolution of 320x240 for Ta = (1/3)s & Tc = 1s and Ta = (1/30)s & Tc = (1/10)s.

Example output and logfiles are in the `./doc/examples/` directory.

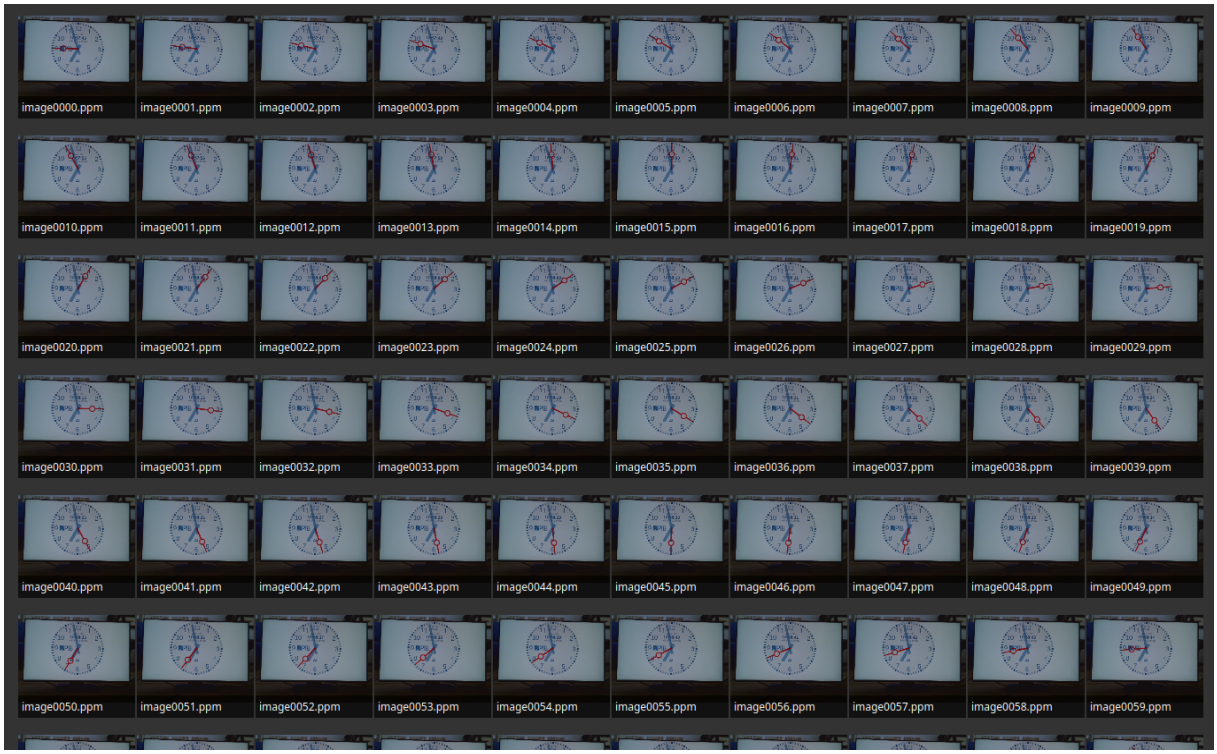


Figure 12: Screenshot of saved images for 1Hz

Figure 13: Screenshot of saved images for 10Hz

5.1 Results and Outlook

- File I/O to the Flash drive is the main bottleneck
- Logging and tracing to the file system introduces competetion for the file system. Decoupling logging into a seperate thread is decisive to prevent erratic blocking of real-time services when logging

6 References

- [1] Liu, C. L.; Layland, J. (1973), “Scheduling algorithms for multiprogramming in a hard real-time environment”, Journal of the ACM, 20 (1): 46–61, CiteSeerX 10.1.1.36.8216, doi:10.1145/321738.321743

6.1 Software Development Stack

Editor:

- (neo)vim editor
- ALE plugin for vim
- vimspector integrated debugger for vim

Documentation in PDF format uses the following tools:

- pandoc universal document converter
- Eisvogel Template Creating uml diagrams from descriptions:
- plantuml
- pandoc-plantuml

Rendering diagrams from runtime traces:

- gnuplot
- fish
- awk