Smart Camera Systems in the Context of Mobile Service Robots - Latest Results

TAMS Oberseminar SoSe 2013

Hannes Bistry

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Technical Aspects of Multimodal Systems

11 July 2013

Outline

My research work on one slide

Overall Goal:
Evaluate Smart Camera Systems for usage on Service Robots

- create a software architecture for integration
- implement image processing functions and scenarios
- use an existing Smart Camera as a testing platform
- evaluate the performance, figure out the benefits and drawbacks
- draw the conclusions in terms of
  - assess usefulness of current camera system
  - define requirements on future types of intelligent camera systems

Enhancing Image Acquisition with Smart Cameras

Definition of a smart-camera:

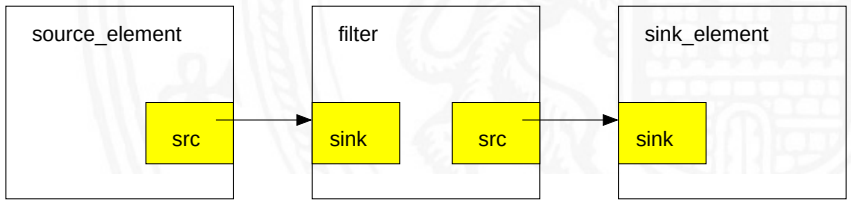
- digital cameras with integrated computing capabilities
- integration of CPU, DSP or programmable hardware
- standard Ethernet interface - no need for special hardware

Advantages:

- image processing directly on the camera
- transmit image information / regions of interest instead of image data
- reduce amount of data
- reduce load on control PC of the robot

Modular Software Architecture

- each processing step is implemented as one element (back-end: GStreamer)
- efficient development and testing of processing strategies
  - plugging together elements instead of low-level-programming
  - exchange of single elements possible (portability)
  - elements can be reused in different context
- timing analysis (all systems synchronized by NTP)
- TCP-elements allow splitting pipelines across network
- system can be applied to
  - Smart Camera systems
  - object detection/robot grasping
  - Cloud Computing



Results of prior research

Benchmarks on Basler eXcite

- slower than a desktop PC
- an appropriate distribution of functions can nevertheless lead to improvements concerning latency and CPU load on the host system

ROI transmission based on face localization

- smart camera is used to scale down images
- extracting ROIs on smart camera reduces network load and latency

Distributed Object detection

- feature vector (about 50 kB) can be sent to many systems in the working environment of a robot
- speedup by searching for many objects in parallel

Why a ROS GStreamer interface is necessary

- using camera with GStreamer drivers in ROS
- use features of GStreamer (compression, network data transfer)

For this work, it is especially important to:

- integrate smart cameras in ROS
- use implemented algorithms for object detection in ROS
- compare efficiency of developed method to that of ROS

ROS Integration

- existing ROS-GStreamer interface “gscam“
- implemented as a standalone ROS-Node
- GStreamer Pipeline is configures in the environment variable “GSCAM_CONFIG“

Drawbacks of the current integration:

- restricted to RGB, 24 bpp format
- fixated to ROS-topic
- inefficient due to unnecessary copy operations
- no timestamps exchanged
- supports only one direction: GStreamer $\rightarrow$ ROS
- run-time control of parameters not possible

Thus the ROS-GStreamer integration was redesigned from the scratch.

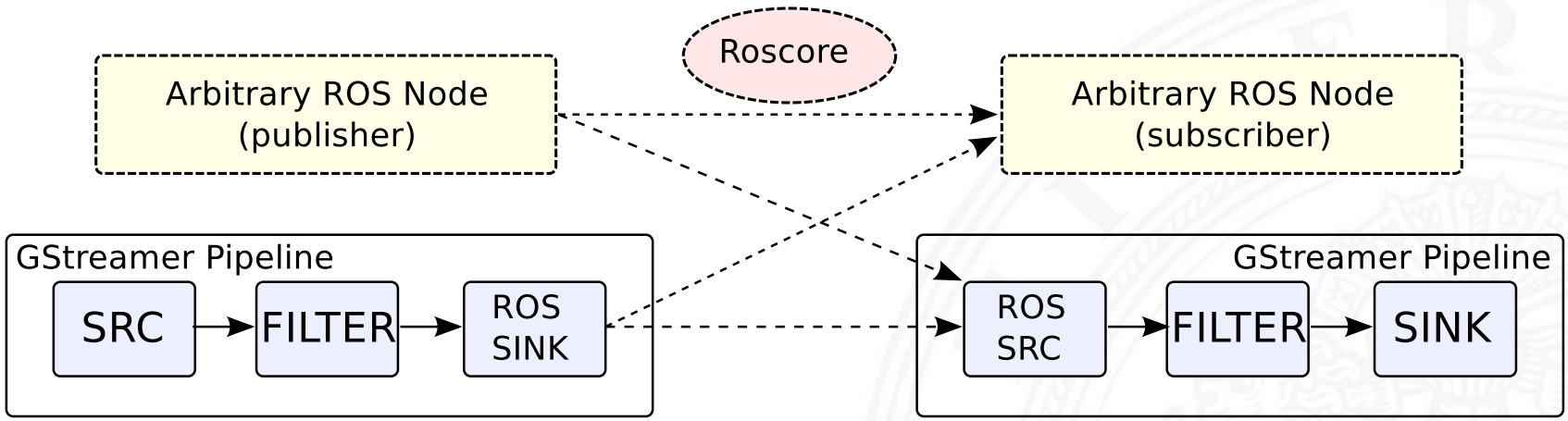
New ROS Integration - Concept

ROS Support implemented as a set of GStreamer plugins:

- **ROSSink:** Publish arbitrary GStreamer video streams in ROS
- **ROSSrc:** Subscribe to a ROS Topic and "publish" images inside GStreamer
- **ROSParam:** Exports parameters of a GStreamer Pipeline to ROS parameter server
- **ROSSiftfolder:** SIFT-based Object detection, publishes transformations in ROS

Wherever possible, memory buffers are shared between GStreamer and ROS (Zero-Copy).

New ROS Integration - Concept



ROSSink

Publish arbitrary GStreamer video streams in ROS.

```
gst-launch videotestsrc ! capsfilter caps="video/x-
    raw-rgb,bpp=24" ! rossink topic=testvideo
```

Will generate the following topics:

```
/testvideo/camera_info
/testvideo/image_raw
/testvideo/image_raw/compressed
/testvideo/image_raw/compressed/parameter_descriptions
/testvideo/image_raw/compressed/parameter_updates
/testvideo/image_raw/compressedDepth
/testvideo/image_raw/compressedDepth/parameter_descriptions
/testvideo/image_raw/compressedDepth/parameter_updates
/testvideo/image_raw/theora
/testvideo/image_raw/theora/parameter_descriptions
/testvideo/image_raw/theora/parameter_updates
```

ROSSink (cont.)

Use cases:

- use arbitrary GStreamer compliant camera in ROS
- integration of smart camera
- publish preprocessed video
- use camera on remote system, using encoders from GStreamer (h264)

Format support: RGB, YUV,gray,jpeg
Timestamp and metadata conversion

ROSSrc

Subscribe to a ROS Topic and “publish“ images inside GStreamer:

```
gst-launch rossrc topic=/testvideo/image_raw !
    ffmpegcolorspace ! ximagesink sync=0
```

Use cases:

- use ROS compliant cameras in GStreamer (including virtual cameras)
- record ROS image topics (with compression)
- drop-in replacement for Image_View (ROS)
  - Display VGA 60 Hz on Core i5:
  - GStreamer: 19 % CPU-load
  - Image_View: 57 % CPU-load

ROSSiftfolder

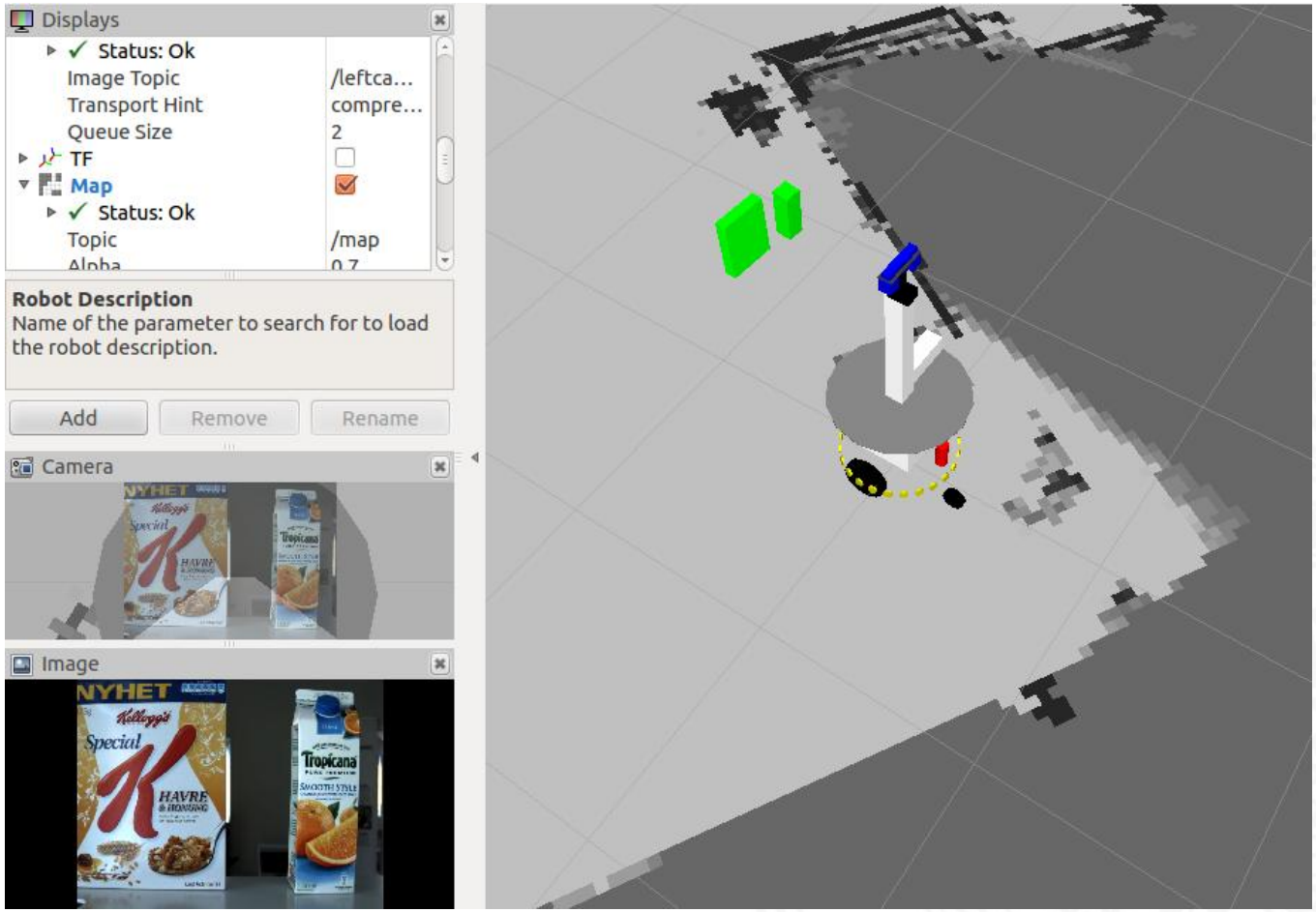
SIFT-based Object detection, publishes transformations in ROS

- all image files in a folder are detected
- Input: feature-vector
- currently, one dimension is encoded in the filename (like box100.jpg)
  - planned feature: database access
- if $>$ 4 matches are found, the 3D pose is calculated

Published topics:

- ROS Vizualization Markers
- tf-Messages from camera frame to object frame

Example: Displaying Objects with RViz



Overhead of modular software frameworks

Aim of the following experiments:

- test the GStreamer-ROS interface
- compare the concept of integrating smart cameras to the "state of the art" methods of implementing interacting software modules
- answer the question, whether it makes sense to completely switch over from GStreamer to ROS (especially on X86 based cameras)

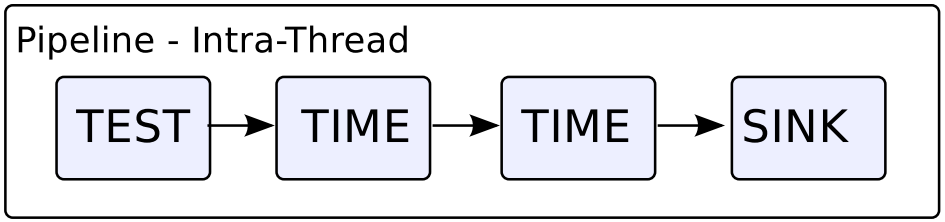
Background - Communication between software modules

If different modules exchange data, we must distinguish between different Modes.

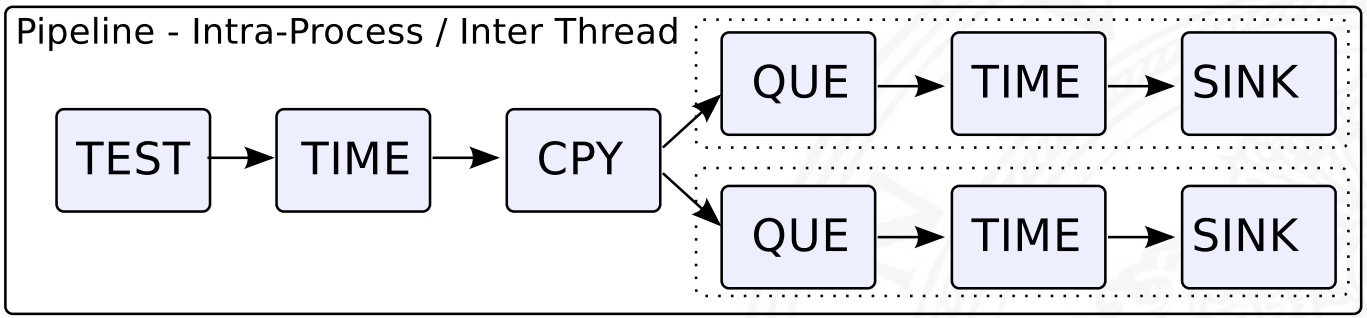
1. Intra-Thread Communication (fastest, only 1 to 1)
2. Intra-Process-Inter-Thread Communication (fast, 1 to N, only single process, only local)
3. Inter-Process Communication (1 to N, only local)
4. Inter-Process Communication using network protocols (1 to N, distributed systems)

Benchmarking different communication modes

1

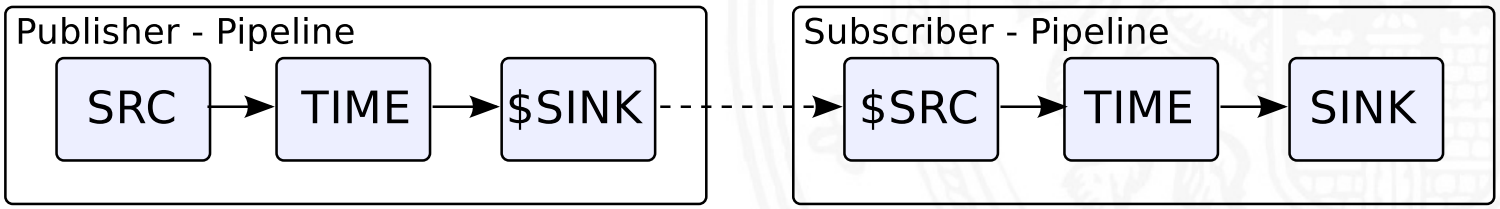


2



Inter-Process Communication - local or on distributed systems
using arbitrary methods (ROS, GStreamer GDP, Shared Memory)

3/4



Background - Communication between software modules

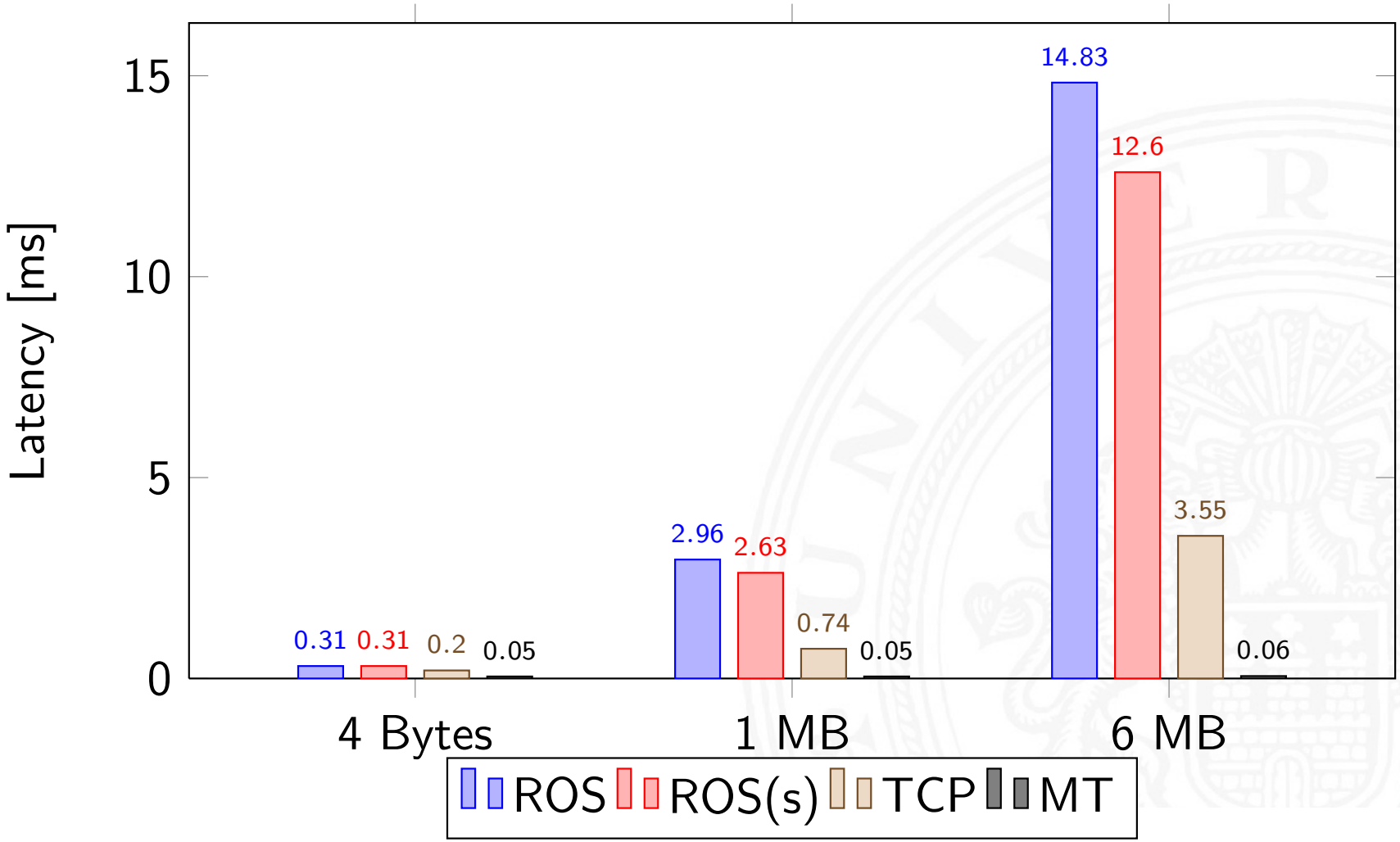
Hardware: Intel Core i5 - 3570
First test: Intra-Thread Communication FullHD image (6MB)
Latency=0.001 ms

Thus:

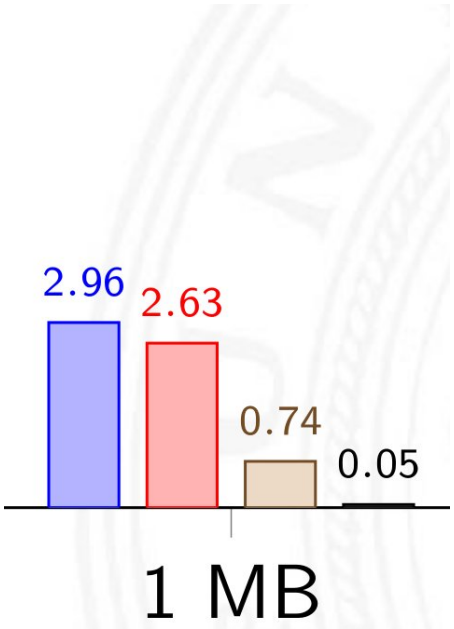
- Overhead of Intra-Thread communication in GStreamer can be neglected in further tests.
- The different methods can be tested “inside“ of the pipelines in the last slide

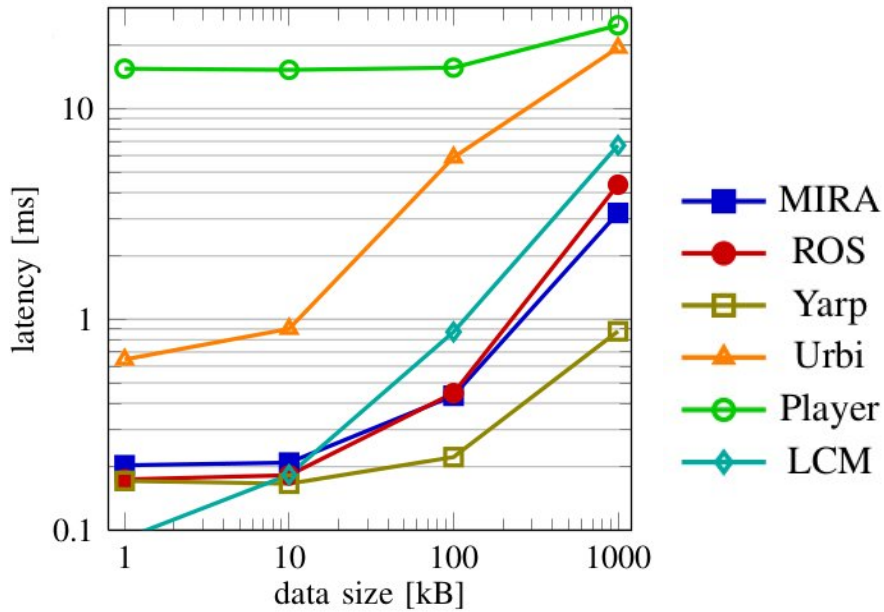
In the next test: ROS , ROS (shared memory), GStreamer TCP, GStreamer Multi-Thread

Benchmarking different communication modes



Benchmarks - Comparison to results from another party





Einhorn et al.,*MIRA - Middleware for Robotic Applications* , IROS 2012

- Similar Results on Core i7 test system for ROS
- GStreamer TCP protocol can probably outperform the state of the art frameworks
- GStreamer multi-threading is extremely fast

Benchmarks - Results

- state of the art frameworks induce overhead
- where possible, multi-threading should be used
- problems:
  - starting and stopping “Nodes“ not possible by default
  - only local communication
  - (imho: debugging a standalone executable is easier than debugging a part of a big system)

Question: Is there a more efficient way of inter-process communication?

Proof of Concept:

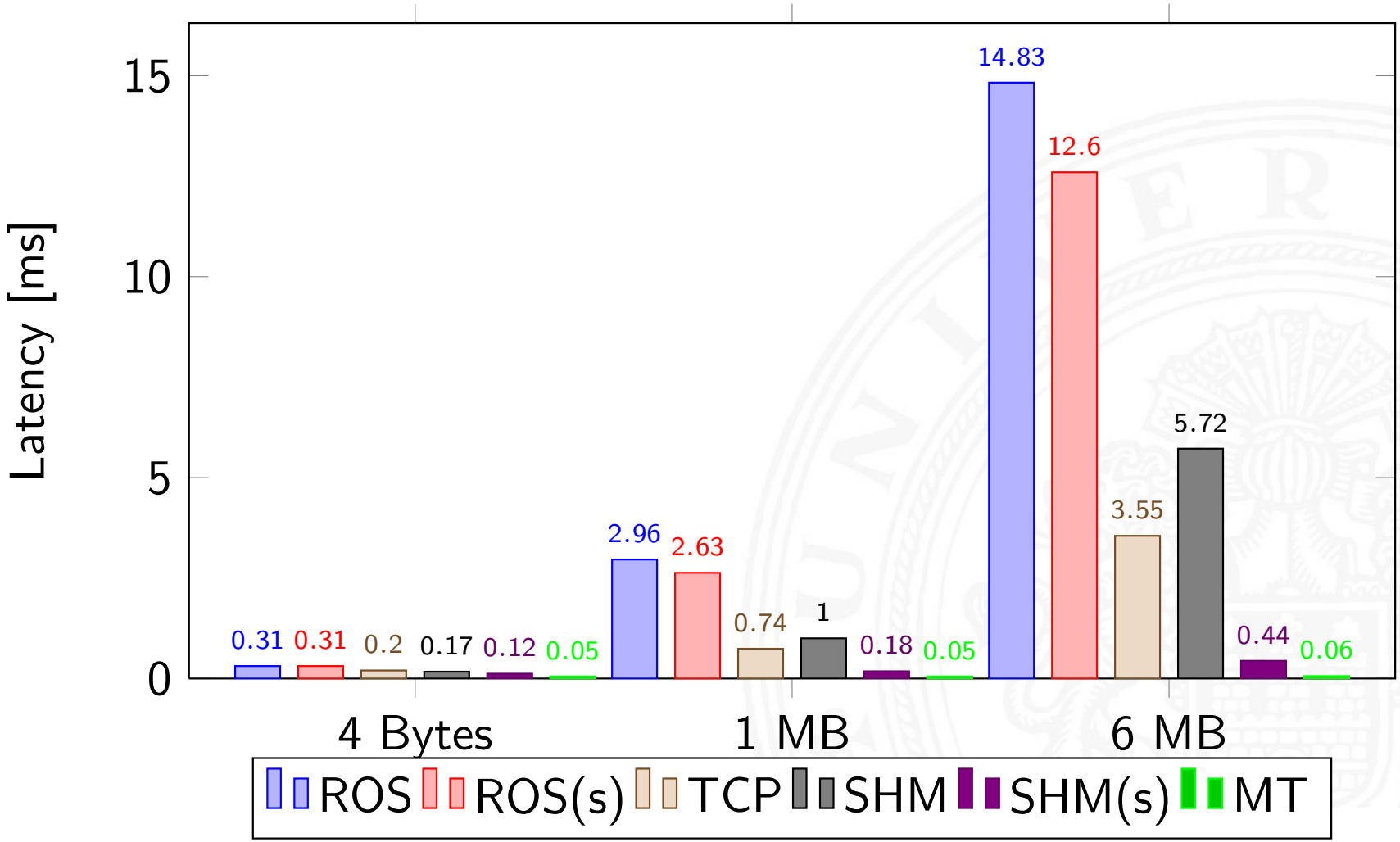
Inter-process Communication using shared memory:

- background: Each process has its own address space
- shared Memory is a mechanism of modern operating systems
- one memory regions can be mapped from different processes
- synchronization by mutexes and semaphores

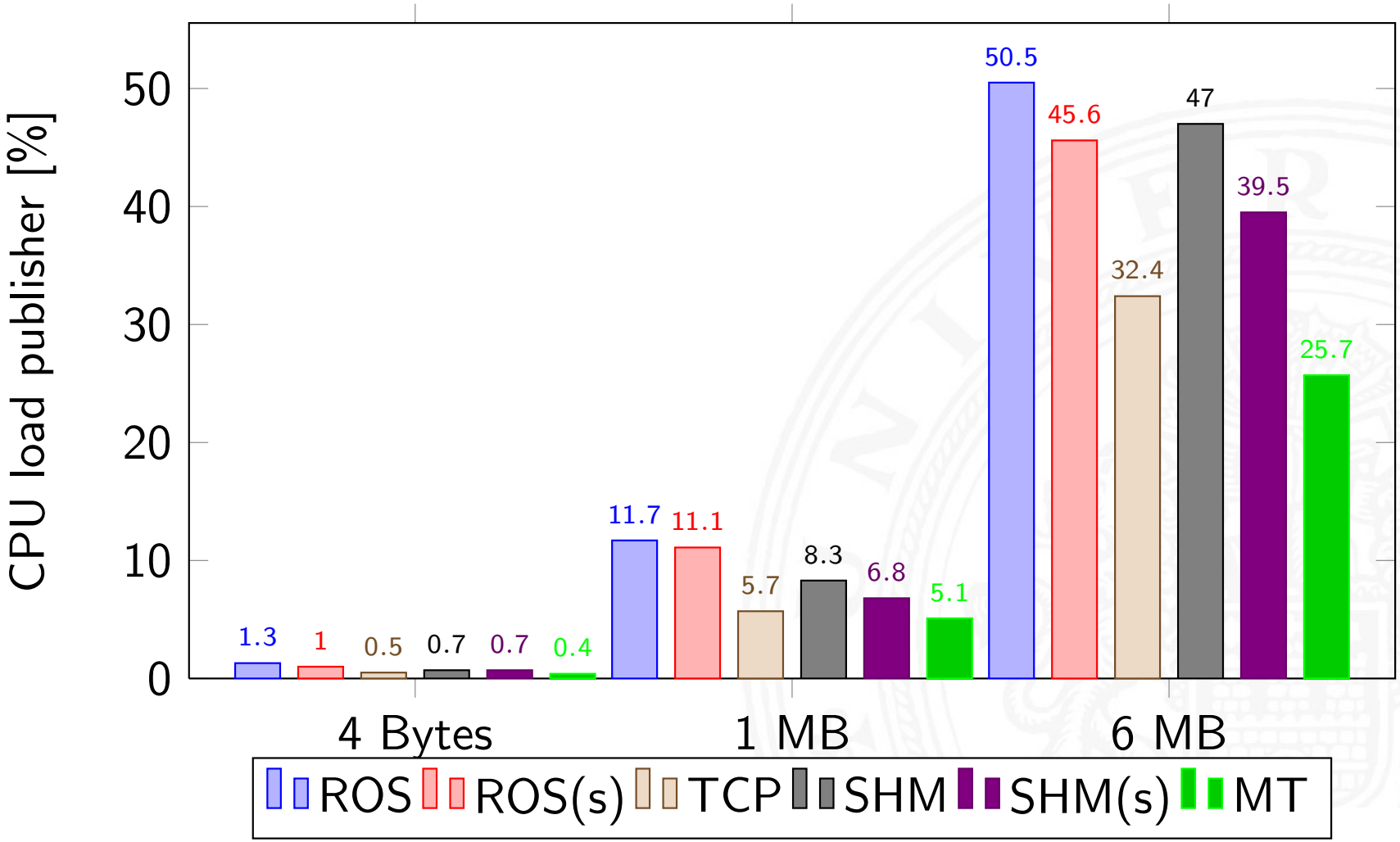
Implemented GStreamer Elements SHMSrc + SHMSink

- “downstream buffer allocation“
- the previous element in the pipeline writes directly into shared memory

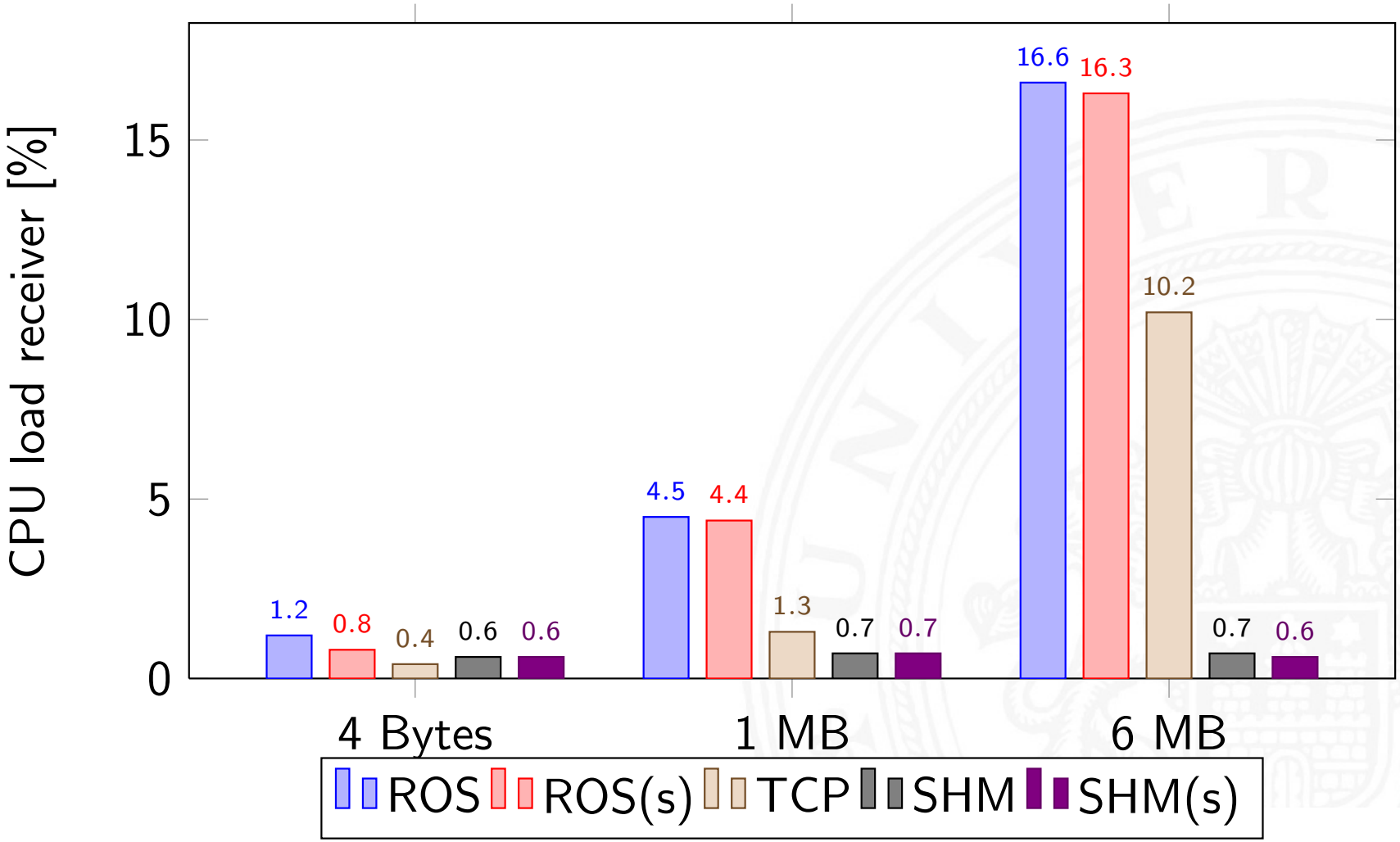
Benchmarking different communication modes



Benchmarks - CPU-load publisher (base load 25%)



Benchmarks - CPU-load receiver



Additional Test: Intel Atom

Test on Intel Atom N270 platform:
Sending a 6 MB FullHD image inside ROS: 132 ms

This leads to unacceptable delays/CPU loads.

Alternatives inside the ROS-Universe

ROS also supports Shared Memory:

- by defining a new datatype
- no compatibility to existing nodes

ROS introduced so called “Nodelets“:

- zero copy pointer-based operation
- started within one “Node“
- no standalone operation - no inter-process communication - no compatibility to existing nodes

In GStreamer, only the element for data transport needs to be exchanged to support new methods.

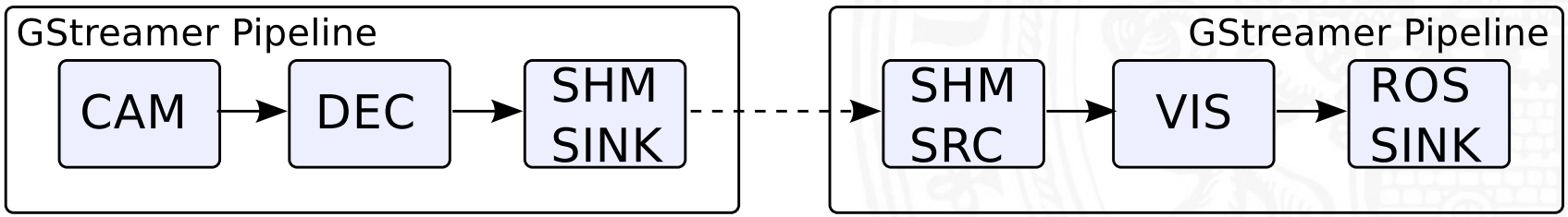
Consequences

- ROS offers great functionality / support for robot hardware
- drawbacks are not that critical for low bandwidth data

No reason to propose “Yet another robot framework“.
Suggestion:

- process high-bandwidth data outside of ROS
- publish the results in ROS

This way, the advantages of both frameworks can be combined.



GPU - Processing

Prior results show that current smart camera systems are to slow to accomplish complex image processing tasks.
Drawbacks:

- embedded processors are slower than high performance desktop PCs
- limited size / heat dissipation capabilities

One way of solving these problems would be to implement parallel computing architectures into smart cameras.

Can GPU based computing be integrated into the developed modular concept?

GPU - Processing

May GPU based processing have advantages over CPU-based processing?

- high performance computing
  - many shaders work in parallel
  - extremely high memory bandwidth
- different ways of programming GPU hardware
  - GLSL (Low-Level Programming of Shaders)
  - CUDA (Nvidia)
  - OpenCL (Apple, Intel, Nvidia, AMD)
- OpenCL is chosen as it is an open standard and widely supported

GPU - Processing

General Drawback of GPU-based programming:

- Data need to be uploaded / downloaded:
- host memory $\leftrightarrow$ video board memory
- but the bandwidth is quite high
- download could be skipped, if data is displayed
- only suitable for algorithms that can be parallelized without a lot of data dependencies

GPU - Processing

Integration into the GStreamer framework:

- Upload and Download within each element, no option to pass data to next element.
- Code for GPU from external .cl file
- new algorithms can be used without installation
- Elements:
  - GstCL (arbitrary functions on gray-scale images)
  - Clrgbafilter (arbitrary functions on color images)
  - clcolorspace (colorspace conversion from YUV to RGBA)
  - clundistort (correction of distortion, including YUV to RGBA conversion)
  - clremap4yuv (panorama stitching)

Ximea CURRERA G

- AMD x86-APU (2 CPU cores 1.6 GHz, GPU-Part 80 Shaders 500 MHz)

Ximea CURRERA G (cont.)

Features:

- Windows Embedded + Linux
- 2 GB RAM
- sensors WVGA up to 5 MPixel
- Gigabit Ethernet + other interface types
- available mid 2013
- **OpenCL compliant**

Ximea CURRERA G (cont.)

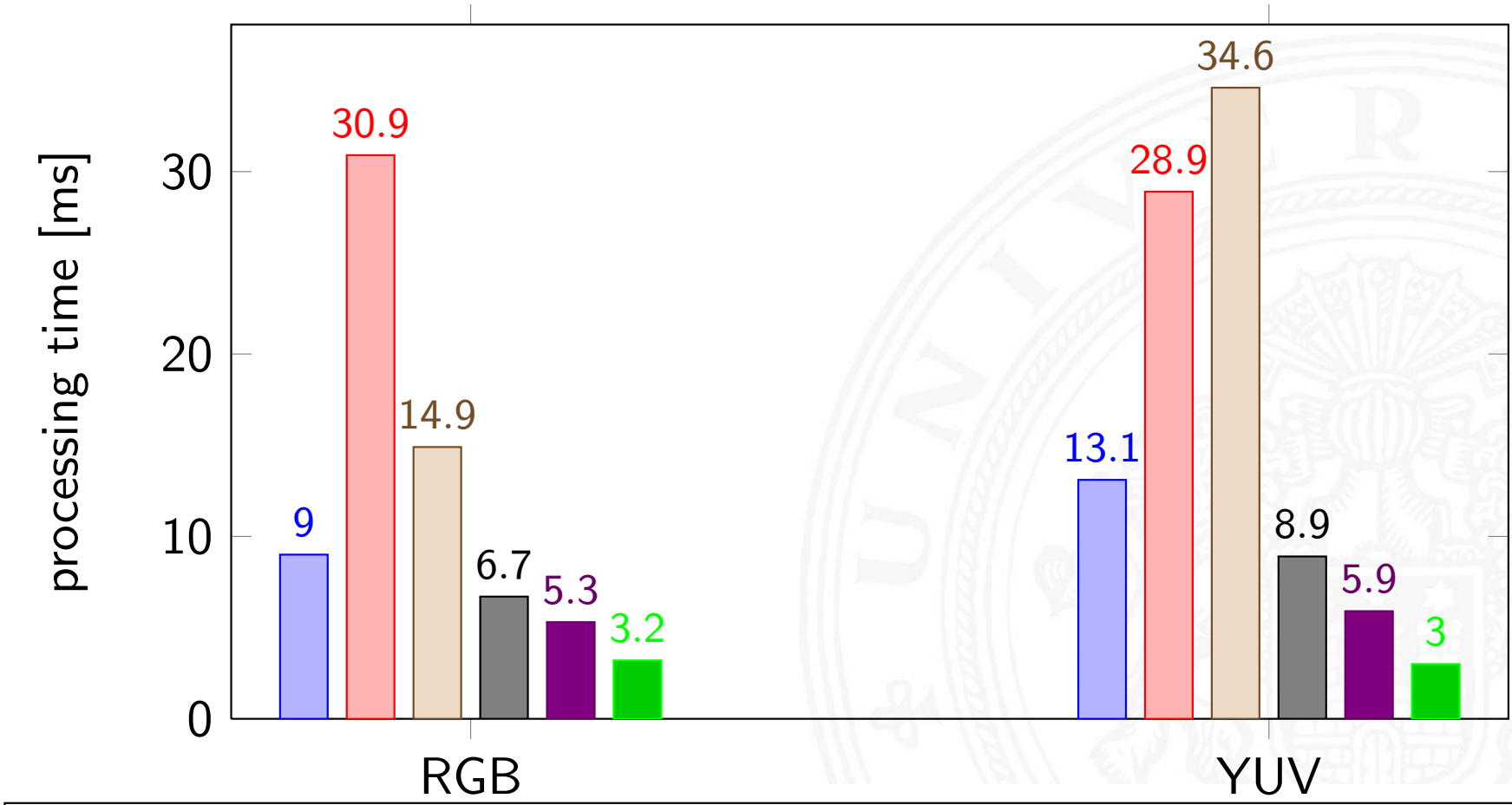
Unfortunately no sample available: Therefore this camera is simulated in the tests using an AMD GPU.

- Radeon 5450
- 80 shader @ 400 MHz

This allows a rough estimation of the processing power to expect from the Ximea CURRERA G.

GPU - Processing

Results for correction of lens distortion, 1384x1032:



Results of GPU-Computing

- middle class GPU can outperform a fast CPU
- CURRERA G will be slower than state of the art hardware
- but it will provide usable framerate
- for comparison: Basler eXcite needs >600 ms for this operation.
- therefore it is usable in additional scenarios
- other tests (Sobel,Laplace) confirm these results

Summary of results

- advanced ROS integration
  - more features
  - better efficiency
- benchmarks on interprocess communication
  - high overhead in ROS
  - provided methods how to solve this problem
- benchmarks on OpenCL based image processing
  - still slower than Desktop hardware
  - usable in scenarios where Basler eXcite is too slow