EyeTracker

Semesterarbeit CAS EBX FS15

Damien Pattier, Cedric Ocaña, Michel Grundmann

Aufgabenstellung

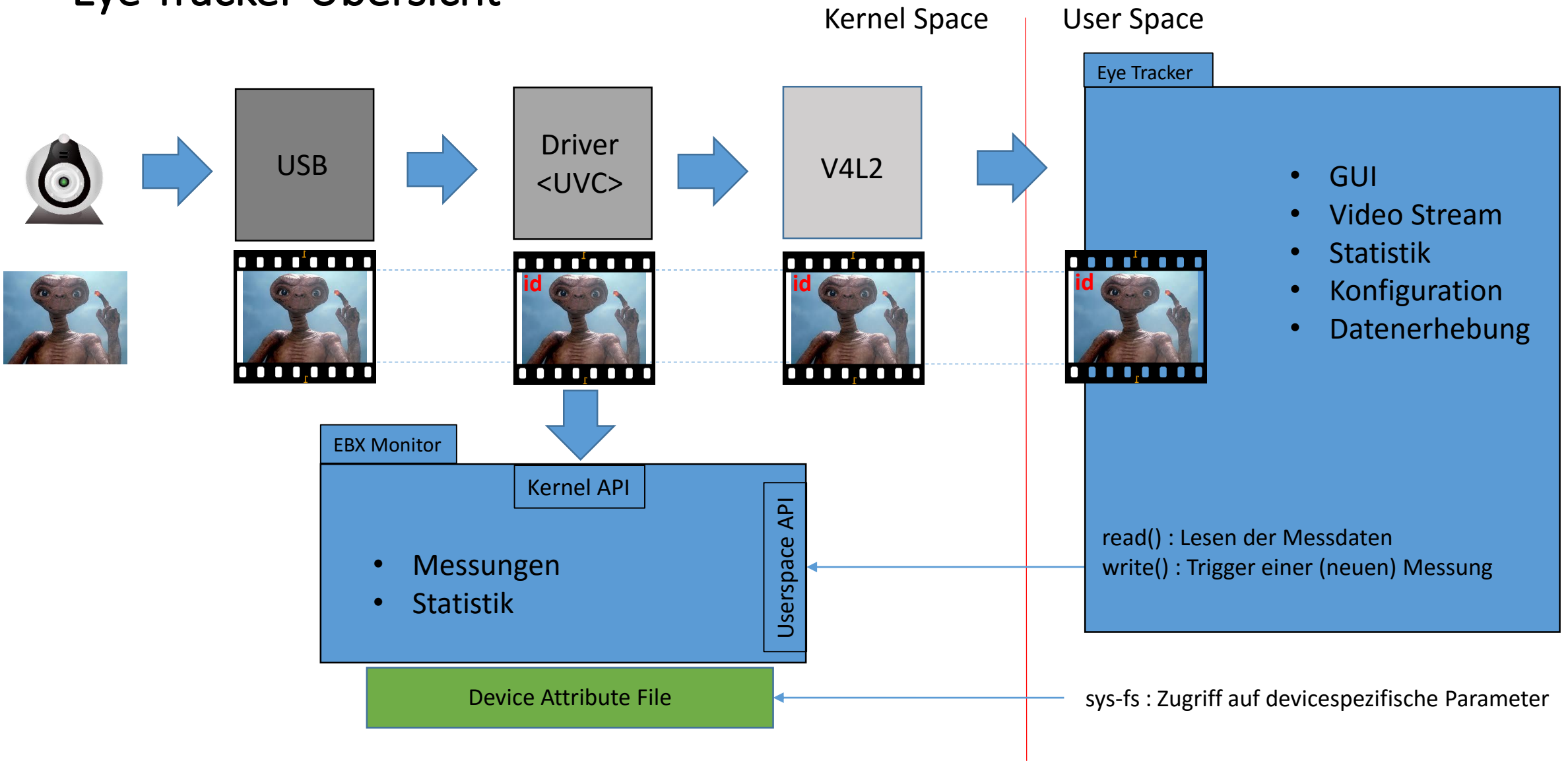
- Basierend auf einem Firmen internen Projekt in dem die Augenpositionen während einer Laseroperation verfolgt werden soll, haben wir uns zum Ziel genommen die Details einer WEBCAM Anbindung in Linux zu untersuchen.

- Während der Untersuchung sollen folgende Punkte erarbeitet werden:
  - Untersuchung der unterschiedlichen Möglichkeiten von USB Kameraanbindungen in Linux.
  - Untersuchung der entsprechenden Kerneltreiber.
  - Messen und erfassen des zeitlichen Verhaltens der vorhandenen Videoinfrastruktur im Kernelspace (Framerate und Latency).
  - Darstellen des Videostreams inklusive der dazugehörigen Messwerte in QT.
  - Darstellung der Messwerte inklusive statistischer Auswertung.
  - Konfigurierbare Parameter aus der EyeTracker Applikation um den Einfluss auf die Messwerte zu identifizieren.

Ablauf der Arbeiten

- Video 4 Linux 2 (V4L2) ist der Standard in Linux
  Es hat sich herausgestellt, dass V4L2 die Standard Video Schnittstelle im (Vanilla) Linux Kernel ist. Aufgrund der begrenzten Zeit war schnell klar, dass andere Schnittstellen auf USB Protokollebene ansetzen und somit den zeitlichen Rahmen sprengen würden. Oder es müsste ein Kamera spezifischer Treiber implementiert werden was eine starke Einschränkung bei der Kameraauswahl bedeuten würde.
- UVC (Kameratreiber) und V4L2 Sourcecode wurden analysiert um Messpunkte zu definieren.
- UVC Kernel Sourcen wurden adaptiert und compiliert um Messungen durchzuführen.
- Definieren des Messvorgangs.
- Design und Implementation des Kernelmoduls EBX-Monitor und der Userspace Applikation EyeTracker.

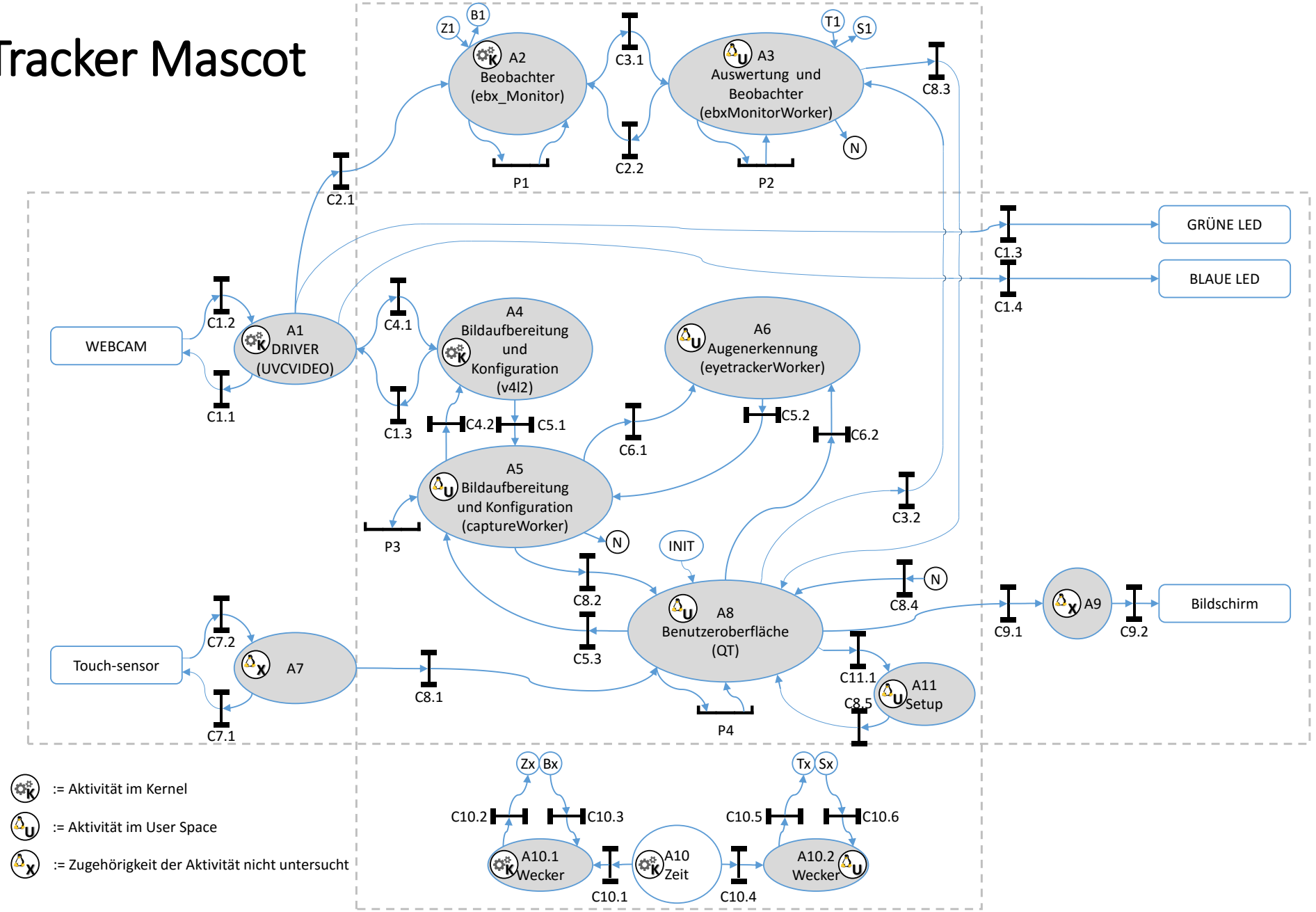
Eye Tracker Übersicht



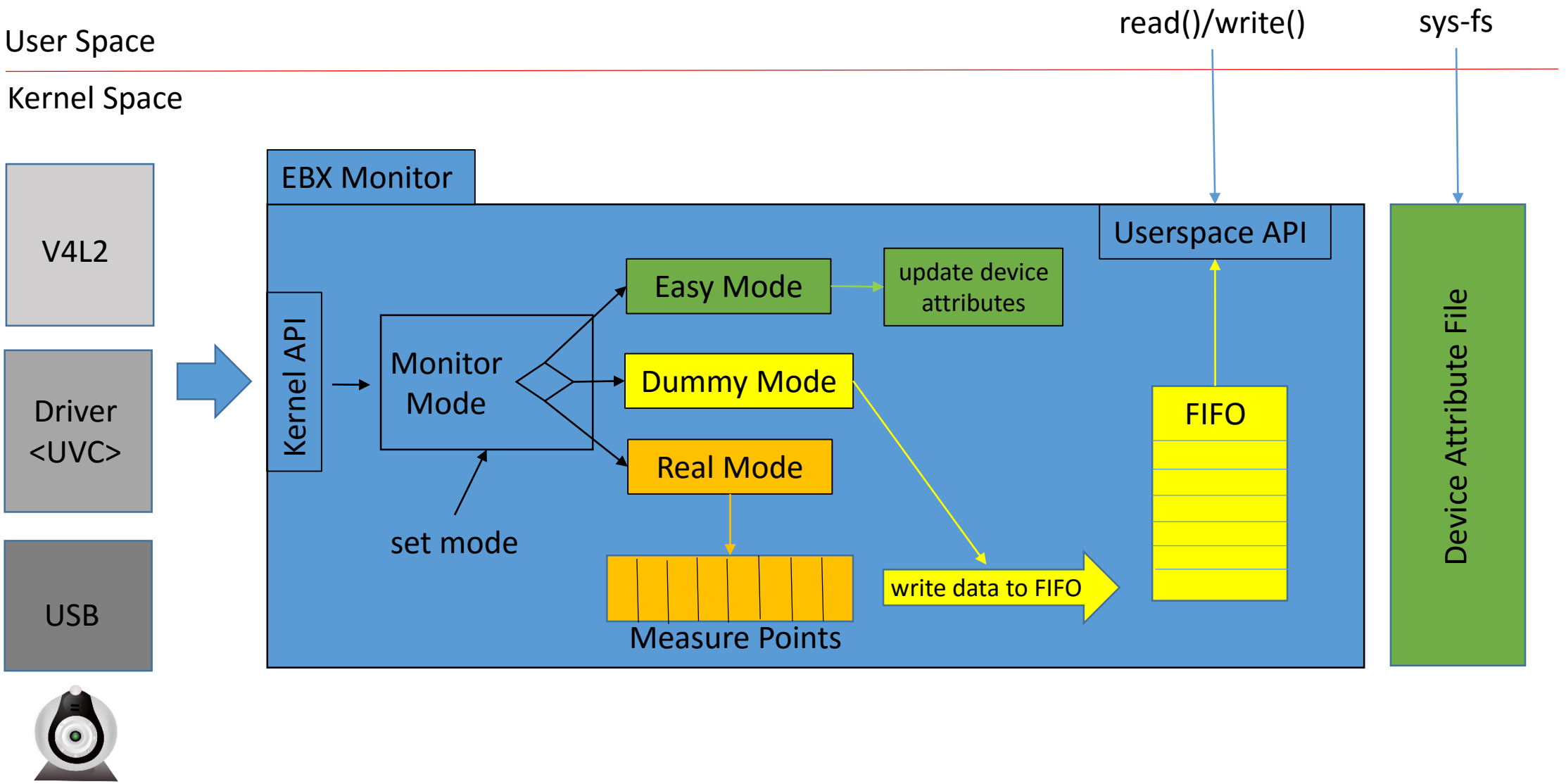
Machbarkeitsstudie (Modifizierung des UVCVIDEO Treibers)

- Es wurde der Quellcode angepasst, welchen wir in der ersten Laborübung benutzt haben. Dadurch wurde der Entwicklungsprozess massgeblich vereinfacht.
- Damit möglichst einfach erkannt werden kann ob der modifizierte Treiber geladen wurden, wird die grüne LED eingeschaltet.
- Damit die Messpunkte an den ebx_Monitor gemeldet werden können müssen die Funktionssymbole vom ebx_Monitor bezogen werden.
  - Falls die Symbole erfolgreich geladen wurden wird die blaue LED eingeschaltet.
- Die Messpunkte wurden an den wichtigsten stellen hinzugefügt und ermöglichen die zeitliche Verfolgung eines spezifischen Bildes.
- Da der original UVCVIDEO Treiber den Zeitstempel beziehungsweise die ID jedes Bildes anpasst, musste vor und nach der Anpassung der ID ein Messpunkt eingefügt werden.
  - Diese zwei Messpunkte werden in der QT Anwendung ausgewertet um die alternative ID zu definieren (Wert in Klammer der ersten Kolonne).

EyeTracker Mascot



EBX Monitor Übersicht



EBX Monitor Beschreibung

- Zeitstempel im Linux Kernel erfassen und ein Userspace API zur Verfügung stellen um die Messdaten zu lesen.
- Nur ein Mode aktiv zur selben Zeit (Daten schützen)
- Monitor Modes
  - Real-Mode: Erfasst Zeitstempel und schreibt die Daten in das «measure point» Array
  - Dummy-Mode: Nach einer abgeschlossenen Messung wird in den «Dummy-Mode» umgeschaltet um die Daten zu schützen. In diesem Mode werden die Messdaten in den Bereich zum Abholen aus dem User API geschrieben. Die nächste Messung ist erst wieder möglich wenn die Messdaten aus dem Userspace vollständig gelesen sind.
  - Easy-Mode: Es werden nur Daten erfasst die über das sys-fs (Device-Attribut-Files) verfügbar sind, z.B. Framerate oder das Delay zwischen den letzten zwei Frames.
- Modul Parameter (modinfo ebx_monitor.ko)
  - nbrOfMeasurePoints (uint)
    Die Anzahl der Messpunkte im EBX Monitor. Der Speicher zum Erfassen der Messpunkte und für den FIFO für das User API wird bei der Modulinitialisierung alloziert. Wenn die Anzahl der Messpunkte null ist (nbrOfMeasurePoint=0), wird der «Easy-Mode» aktiviert. Sind Messpunkte konfiguriert wird der «Real-Mode» aktiviert.
  - deferredFifo (bool)
    Wenn der Parameter auf «Y» konfiguriert wird, werden die Messdaten in einer Work-Queue in das FIFO geschrieben, was einen geringeren Einfluss auf den Videostream hat. Sonst werden die Daten direkt nach der letzten Mesung ins FIFO geschrieben.
- Eine detaillierte Beschreibung ist im Header vom Sourcefile zu finden.

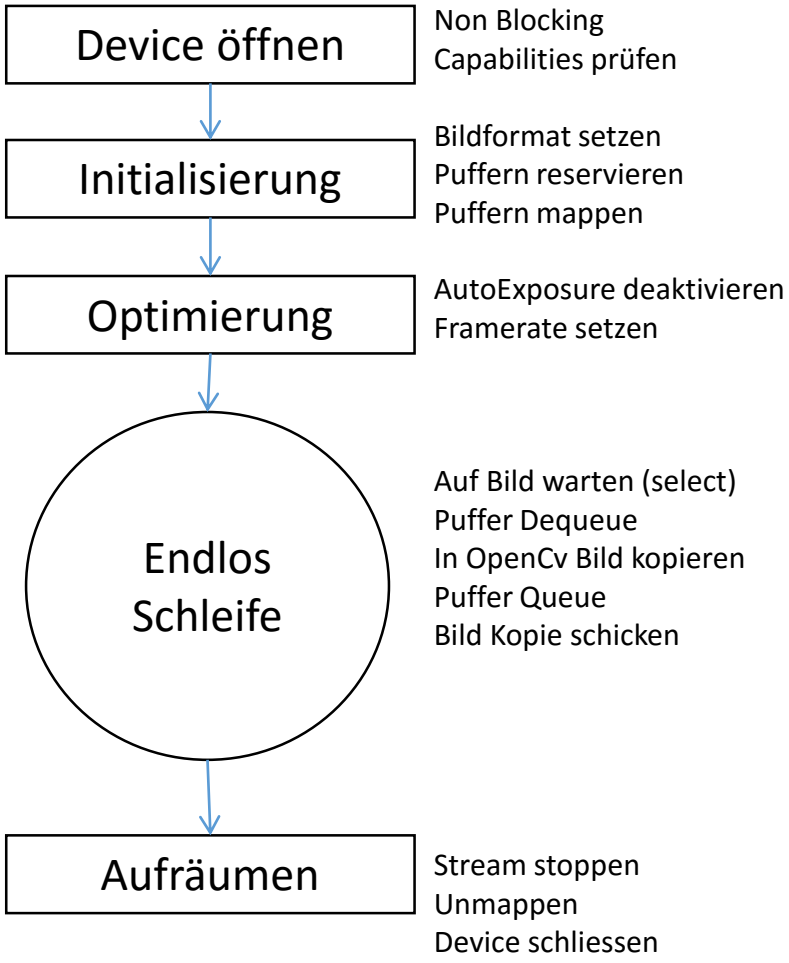
EBX-Monitor API

- Userspace
  - read() blocking und non-blocking.
    Um mit «cat» die Daten zu lesen, kann via Device-Attribut-File «readOneShot» ein non-blocking Zugriff simuliert werden.
  - write()
    Wird das Kommando «start» geschrieben, wird eine Messung (neu) gestartet. Messdaten die nicht gelesen wurden gehen verloren.
  - Device-Attribut-Files: /sys/devices/virtual/misc/ebx_monitor
    - totalFrameCount
    - totalFrameDelta
    - lastFrameDelta
    - averageFrameTime
    - readOneShot

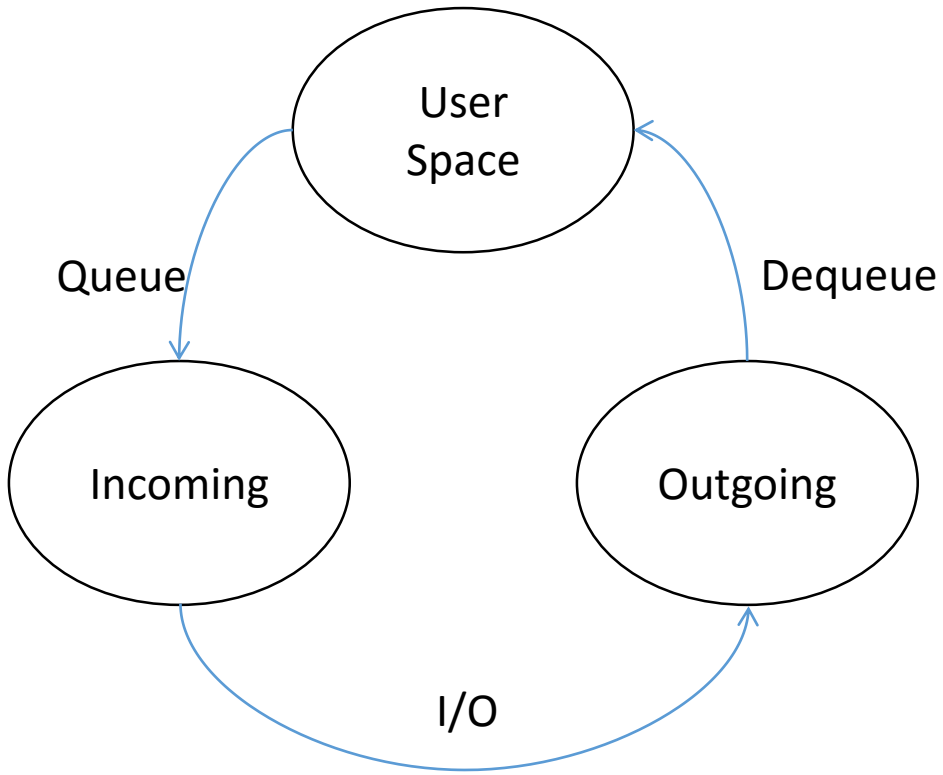
- Kernelspace
  - void ebx_monitor_gotnewframe(const struct timeval* inTimeOfNewFrameP)
    inTimeOfNewFrameP: Initialer Zeitstempel, wird als «id» des Frames verwendet. Der EBX Monitor erfasst zusätzlich den Zeitstempel vom Funktionsaufruf.
  - void ebx_monitor_gotframe(const struct timeval* inTimeOfNewFrameP, const unsigned char inTag)
    inTimeOfNewFrameP: Frame «id», der EBX Monitor erfasst den Zeitstempel vom Funktionsaufruf.
    inTag: Identifiziert einen Funktionsaufruf (maximum Wert = 255)

Details captureWorker (wie wir die Bilder von der Kamera beziehen)

Workflow



State Machine



Resultate

- Ein Frame braucht weniger als 500us von UVC-Video bis in den Userspace.
- Framerate ist, falls keine Datenverarbeitung gemacht wird, je nach Kamera entweder rund 15fps oder 30fps.
- Die Framerate ist bei eingeschalteter Datenverarbeitung nicht stabil weil die Erfassung der Ankunftszeit beeinflusst wird. Die Berechnung müsste in den captureWorker verschoben werden um dies zu optimieren.
- UVC-Video und V4L2 sind nicht das Problem für die niedrige Framerate und zeigen keine erhebliche Latency.
  - Es braucht keine weiteren Messpunkte in V4L2
- Das Problem der niedrigen Framerate ist entweder der USB-Stack oder die Kamera.
- Es müsste ein Test gemacht werden mit einer alternativen Kameraanbindung. z.B. Ethernet Kamera (Bildübertragung via TCP oder UDP)
- Linux «top» der EyeTracker Applikation zeigt die Last der unterschiedlichen Threads.
  - Multicore CPU = bessere Lastverteilung der Userspace Applikation
  - Schnellere CPU = grundsätzlich bessere Leistung der EyeTracker Applikation

Erfahrungen aus der Semesterarbeit

- QT ist für die GUI Entwicklung gut geeignet, benötigt aber einiges an Einarbeitung und CPU Leistung.
- UVC-Video ändert den Zeitstempel der als «id» verwendet wird, was eine eindeutige Erkennung eines Frames vom Kernel bis in den Userspace erschwert.
- Beim Modifizieren von Kernel Sourcecode ist es wichtig darauf zu achten ob man sich im Kernel im Atomic oder im Process-Kontext befindet.
- Versionenkontrolle (z.B. Git) ist auch bei kleineren Softwareprojekten eine grosse Hilfe.

Quellcode verfügbar auf GitHub

- Eye Tracker
  - git clone https://github.com/EBXFS15/EyeTrackerQt
  - Git-Repository Umfang
    - QT-Project
    - Detailliertes Readme (Komplettanleitung)
- EBX Monitor
  - git clone https://github.com/EBXFS15/ebx_monitor
  - Git-Repository Umfang
    - Makefile
    - Kernelmodul BBB
- Modifizierte UVC-Video
  - git clone https://github.com/EBXFS15/uvc-from-bbb-sources
  - Git-Reporsitory Umfang
    - Kernelmodul für BBB
    - Readme