FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Entwicklung eines autonomen Überwachungssystems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Masterstudium Visual Computing

eingereicht von

Adis Buturovic, BSc.
Matrikelnummer 0125423

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Univ.Prof. Dipl.-Ing. Dr.techn. Christian Breiteneder
Mitwirkung: Dr.techn. Dipl.-Ing. Matthias Zeppelzauer

Wien, 08.08.2016

(Unterschrift Verfasser) (Unterschrift Betreuer/in)

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____

Ort, Datum, Unterschrift

Abstract

Video and audio monitoring systems are used across the globe to observe certain events and processes. Usually, such systems are connected to a communications network and electricity grid and are not equipped with any kind of processing logic (“intelligence”) other than data compression. How can such systems be used if there is no electricity nor a broadband communications network available? The aim of this thesis is to develop an autonomous monitoring system built entirely from commercially-available standard components, which can operate independently.

The first step to achieve this is to conduct a pre-evaluation of existing hardware and software components according to various criteria. The next step involves the development of a hardware prototype as well as the implementation of an efficient and modular software framework that can be adapted for different applications. The system should support policy-based recording modes such as modes controlled by input data from defined sensors, like motion and light sensors, that make autonomous decisions about the use of available resources. The software framework should be designed to support the integration of image processing algorithms to allow pre-processing of recorded (video) data on board in order to save bandwidth.

In the final prototype, both the recording of audio and video (among other) data was realized to the desired extent. Using an application, the recording mode can be set easily using a XML configuration file. The power consumption of the build system is, depending on used sensors and executed tasks, between 0.68 to 1.75 watts and lies within the range of comparable commercial solutions, such as “leanXcam”.

Zusammenfassung

Weltweit werden Überwachungssysteme (z.B. mit Kameras und Mikrofonen) eingesetzt, um bestimmte Ereignisse und Umgebungen zu beobachten. Diese werden üblicherweise an ein Kommunikationsnetz und Stromnetz angeschlossen und beinhalten keine eigene Verarbeitungslogik ("Intelligenz"). Wie verfährt man jedoch, wenn weder Strom noch ein Breitband-Kommunikationsnetz zur Verfügung stehen? Das Ziel dieser Diplomarbeit ist es, ein autonomes Überwachungssystem basierend auf handelsüblichen Komponenten zu entwickeln, das möglichst autonom und unabhängig agieren kann und wenige Abhängigkeiten nach außen hat.

In der Arbeit wird zunächst der Auswahlprozess von Hard- und Software anhand verschiedener Kriterien beschrieben. Der Hauptteil der Arbeit umfasst die Entwicklung eines Hardware-Prototypen sowie die Implementierung eines leistungsfähigen und modularen Software-Frameworks, das unterschiedliche Einsatzzwecke ermöglichen soll. Das fertige System soll regelbasierte Aufnahme-Modi unterstützen, beispielsweise gesteuert durch Eingangsdaten von Bewegungs- und Lichtsensoren und basierend darauf autonom Entscheidungen über den Umgang mit den vorhandenen Ressourcen treffen. Das entwickelte Software-Framework soll die Integration von Bildverarbeitungsalgorithmen unterstützen, um eine Vorverarbeitung der Kameradaten im Überwachungssystem selbst zu ermöglichen und Bandbreite zu sparen.

Sowohl die Audio- als auch Videoaufnahme wurde im gewünschten Umfang realisiert. Die Konfiguration des Systems kann bequem mittels einer XML-Konfigurationsdatei durchgeführt werden. Der Stromverbrauch liegt bei dem entwickelten System je nach Aufgabenstellung und den verwendeten Sensoren zwischen 0,68 und 1,75 Watt und liegt somit im Bereich von vergleichbaren kommerziellen Lösungen, wie beispielsweise "LeanXcam".

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Überwachungssysteme werden eingesetzt, um bestimmte Ereignisse und Prozesse festzuhalten und auszuwerten. Solche Systeme bestehen generell aus einer Eingabe-Einheit (z.B. Kamera) sowie einer getrennten Datenverarbeitungseinheit (wie beispielsweise in [Kruse 98] und [Bautista Saiz 14]). Die Kamera wird üblicherweise an ein Kommunikationsnetz bzw. Stromnetz angeschlossen und dient im Falle einer Videoüberwachung rein der Bildakquisition, d.h. hierbei wird die Information entweder nur weitergereicht oder muss noch zusätzlich gespeichert werden. Unabhängig davon, ob ein oder mehrere Bilder aufgenommen werden sollen (Foto oder Video), kann eine Weiterverarbeitung der Daten auf der Kamera selbst nur zum Nutzen der Verringerung der Datenmenge bzw. Bandbreite durchgeführt werden [Sasson 91]. Die aufgenommenen Daten werden dabei auf einem lokalen oder entfernten Speichermedium gespeichert und nachträglich von einem weiteren, unter Umständen zeitlich bzw. geografisch separierten System weiterverarbeitet bzw. analysiert. Im Falle einer geografisch entfernten Verarbeitung ist es notwendig, den entsprechenden Speicherplatz sowie eine entsprechende Übertragungsbandbreite zur Verfügung zu stellen.

Neben dem Aspekt der Datenspeicherung sind die meisten Überwachungssysteme dauerhaft an das Stromnetz angeschlossen und nehmen unabhängig vom Geschehen ununterbrochen auf. Zum einen entsteht dadurch eine sehr große Datenmenge (die es nachträglich zu analysieren gilt) und zum anderen sind diese Systeme auf die externe Stromversorgung angewiesen.

Durch die stetig steigende Rechenleistung sowie immer kleiner werdende Hardware ist erkennbar, dass die Komponenten eines Überwachungssystems zukünftig verschmelzen werden. Diese Überlegungen sind die Motivation für die vorliegende Arbeit, deren Ziel es ist, bestehende, vornehmlich preisgünstige Hard- und Softwarekomponenten zu kombinieren, um die Anforderungen an die Audio- bzw. Videoaufnahme, Analyse und Weiterverarbeitung der Daten sowie an die Erweiterbarkeit durch Sensoren (z.B. zum Anstoßen der Aufnahme-Prozesse) in einem integrierten System zu vereinen. In weiterer Folge soll die prototypische Entwicklung eines solchen Systems durchgeführt werden. Hierbei soll eine möglichst lange Aufnahme- und Standbyzeit, d.h. geringer Stromverbrauch, erreicht werden, um eine Selbstversorgung, beispielsweise über kleinere Photovoltaik-Anlagen, zu ermöglichen. Bei der Analyse der Daten kann es sich um Gesichtsdetektion [Viola 04], Bewegungsdetektion [Cutler 00] und um die Realisierung komplexer Algorithmen, wie Objekterkennung [Viola 01], handeln. Somit könnten die oben angeführten Schritte der Audio- und Videoübertragung und Analyse auf einem entfernten System entfallen bzw. durch eine verteilt stattfindende Vorfilterung unterstützt werden und Bandbreite gespart werden.

1.2 Problemstellung

Existierende Überwachungssysteme benötigen im Allgemeinen eine externe Stromversorgung bzw. eine ausreichend hohe Bandbreite. Wenn dies nicht der Fall ist, sind die angebotenen Lösungen rar. Bei einem Verkehrsüberwachungssystem in Regionen mit schwacher Infrastruktur bzw. limitierten Ressourcen, seien es Strom oder eine limitierte Datenübertragungsbandbreite, wäre ein autonomes, selbstversorgendes, vom Benutzer konfigurierbares System von Vorteil. Dieses könnte durch Bewegungs- oder Ultraschallsensoren getriggert eine Videoaufnahme auslösen, bestimmte weitere Merkmale extrahieren (z.B. Kennzeichen) und nur diese Ergebnisse speichern bzw. zu anderen Systemen übertragen.

Das Ziel dieser Diplomarbeit ist es, das Potential und die Machbarkeit eines solchen möglichst autonomen Überwachungssystems zu evaluieren, Anforderungen zu definieren sowie eine preisgünstige “Proof-of-Concept”-Lösung zu entwickeln. Dies umfasst die Evaluierung der am Markt befindlichen Hardware- und Software-Plattformen, die Entscheidungsfindung und letztendlich die Erstellung und Evaluierung eines Prototypen als “Proof-of-Concept”.

Die Priorität soll vor allem auf der Video-Aufnahmequalität sowie Verarbeitungsgeschwindigkeit bei niedrigem Stromverbrauch liegen. Erwünscht ist eine Videoaufnahme mit einer möglichst hohen Bildwiederholfrequenz in HD-Auflösung, um beispielsweise bei schnell bewegenden Objekten genügend Informationen zwischen zwei aufeinanderfolgenden Bildern zu erhalten. Auch die Konfigurationsmöglichkeit durch den Benutzer muss vorhanden sein. Die genauen Anforderungen sind in Unterkapitel 1.3.1 spezifiziert.

Um die Aufnahme und Auswertung der Daten auf dem System durchzuführen, sollen Standardbibliotheken bzw. Softwarepakete, die Algorithmen für Bildverarbeitung und maschinelles Sehen bereitstellen, verwendet werden, um von zukünftigen Entwicklungen profitieren zu können. Im weiteren Verlauf der Arbeit wird die Untersuchung folgender Fragestellungen und Problemstellungen angestrebt:

Aufnahme und Analyse

- Inwieweit können die Anforderungen der Video- und Audioaufnahme (bezogen auf Abtastrate und Auflösung) erfüllt werden?
- In welchen Video- bzw. Audio-Formaten kann eine Aufnahme realisiert werden?
- Wie lange kann aufgenommen werden bzw. wie kann der Speicherplatz erweitert werden?
- Wie gut kann die Synchronizität zwischen Video- und Audiospuren gewährleistet werden?
- Kann zusätzlich zur Video- und Audioaufnahme eine gleichzeitige Datenanalyse ausgeführt werden?

Entwicklung und Erweiterbarkeit

- Welche Hardware ist notwendig, um die Anforderungen zu erfüllen?
- Wie kann eine einfache Konfiguration des Systems aussehen?

Stromaufnahme

- Mit welchen Hardware-Plattformen ist die niedrigste Leistungsaufnahme zu erwarten?

- Mit welchem Stromverbrauch muss bei der Verwendung mehrerer Eingabequellen sowie den zu erwartenden Hintergrund-Prozessen (z.B. nachträgliche Video-Komprimierung oder Bewegungsdetektion) gerechnet werden bzw. kann dies bei der "Proof-of-Concept"-Lösung eingehalten werden?

Weiters soll ein leicht erweiterbares, unter MIT-Lizenz [Mas 15]) quelloffenes Software-Framework entwickelt werden. Obwohl eine geringe Stromaufnahme eine wichtige Anforderung darstellt, ist die Entwicklung einer Stromversorgung (z.B. Photovoltaik-Anlage mit Ladeeinheit) kein wesentlicher Punkt dieser Arbeit. Auch rechtliche Aspekte der Video-/Audioaufzeichnungen (z.B. Schutz der Privatsphäre) werden nicht im Rahmen dieser Arbeit behandelt.

1.3 Anwendungsszenario

Diese Arbeit ist im Kontext eines Forschungsprojektes entstanden, welches sich mit der visuellen und akustischen Überwachung (Monitoring) von Tieren beschäftigt mit dem Ziel deren Verhalten zu analysieren [Zeppelzauer 13]. Speziell in diesem Bereich sind autonome Überwachungssysteme notwendig, da im Freiland kaum Infrastruktur existiert. Autonome Systeme für das Monitoring im Freiland gibt es vorwiegend im Audio-Bereich (z.B. SM4 von "Wildlife acoustics, Inc." [Wil 16]). Für Videoaufnahmen sind nur sehr wenige, begrenzt konfigurierbare Systeme vorhanden, die weder Datenanalysen "on-the-fly" ermöglichen noch für den Einsatz im Feld aufgrund notwendiger externer Stromversorgung geeignet sind. Das Ziel dieser Arbeit ist es, ein solches System vor allem für den Zweck der Tierbeobachtung zu entwickeln. Sowohl für die Evaluierung der bestehenden Systeme, als auch für die Erstellung der Anforderungen und der "Proof-of-Concept"-Lösung wird eine möglichst stromsparende Lösung benötigt. Im Kontext des Forschungsprojektes, welches dieser Arbeit zugrunde liegt, wurden folgende Anwendungsfälle für die Tierbeobachtung definiert, welche mit dem prototypisch umgesetzten Überwachungssystem evaluiert werden sollen.

- Anwendungsfall 1: Eine nahezu durchgehende Videoaufzeichnung bei Tageslicht (Bildwiederholfrequenz und Auflösung sollen an Strom-/Platzverbrauch angepasst werden) und in der Nacht nur eine Audioaufnahme bei durch Sensoren erkannter Bewegung (z.B. Bewegungs- oder Ultraschallsensor). Dies würde sich eignen, um die Präsenz von Tieren

zu bestimmen.

- Anwendungsfall 2: Eine Video-Trigger gesteuerte Aufnahme mit beispielsweise einer Frequenz von einem Bild pro Sekunde und einem Objekt-Erkennungsalgorithmus, der eine Video- und Audioaufzeichnung auslöst, solange Objekte erkannt werden. Hiermit kann sowohl die Präsenz als auch das Verhalten einer bestimmten Tierart (z.B. Elefanten) analysiert werden.

- Anwendungsfall 3: Durchgängige Videoaufzeichnung im Zeitraffer-Modus (beispielsweise alle fünf Minuten ohne nachträgliche Datenanalyse). Dieser Fall eignet sich, um besonders lange und stromsparend Aufzeichnungen durchzuführen. Beispielsweise kann ein solches System über Tage autonom aufnehmen.

In allen Fällen ist eine Vielzahl von Entscheidungen notwendig. Anwendungsfall 1 wird im folgenden Flussdiagramm (Abbildung 1.1) vereinfacht dargestellt.

Weitere Anwendungsmöglichkeiten eines solchen autonomen Überwachungssystems, die über Tierbeobachtung hinausgehen, sind vielfältig. Generell wird hierbei an eine Überwachung von öffentlichen Plätzen gedacht, dazu zählen beispielsweise Verkehrsüberwachung und Personenbeobachtung.

Bei der Verkehrsüberwachung kann von einer kontinuierlichen Videoaufnahme ausgegangen werden. Gleichzeitig könnte bei Erkennen eines Fahrzeugs (durch Algorithmen der Objekterkennung) ein Trigger-Mechanismus die Speicherung weiterer Sensordaten (beispielsweise Sensoren, die eine Entfernungsmessung durchführen) auslösen.

Auch bei der Personenüberwachung könnten durch Gesichtserkennungsalgorithmen diverse darauf folgende Mechanismen ausgelöst werden, wie zum Beispiel eine Audioaufnahme.

1.3.1 Anforderungen an das System

Für die Entwicklung der “Proof-of-Concept”-Lösung wurden unter Betrachtung unterschiedlicher Monitoring-Szenarien folgende Mindestanforderungen festgelegt. Detaillierte Anforderungen werden nach der Evaluierung der Entwicklungsumgebungen und ähnlicher Systeme in Unterkapitel 3.1 ergänzt.

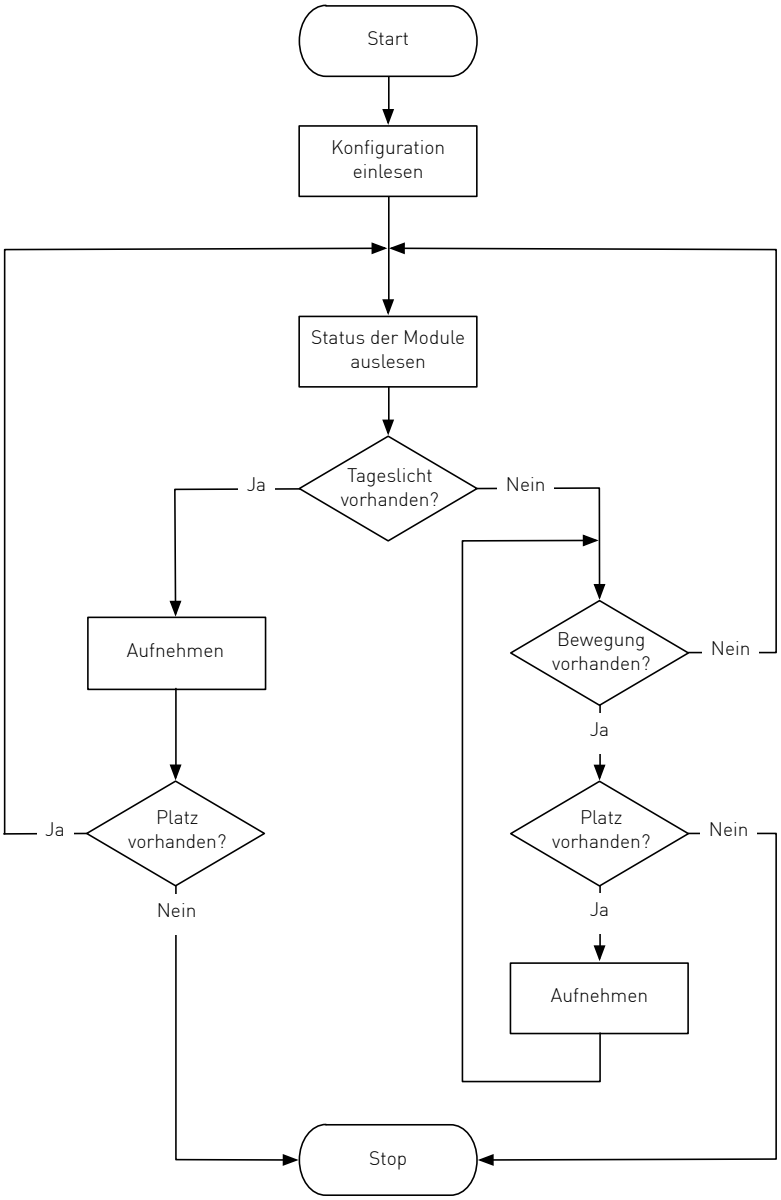

Abbildung 1.1: Flussdiagramm - Anwendungsfall 1

Generelle Anforderungen

- Die Aufnahme soll durch verschiedene Sensoren ausgelöst werden können. Unabhängig vom “Trigger-Sensor” sollen Audio-, Video-, Distanz-, Bewegungs- und Helligkeitsdaten möglichst energiesparend aufgenommen werden. Die Umsetzung der Sensoren bzw. Funktionen wird in Tabelle 1.1 dargestellt.
- Eine softwareseitige, durch Datenanalyse ausgelöste Triggerung soll möglich sein.

- Die Übertragung von Daten (bzw. Ergebnissen) soll über das TCP/IP-Protokoll erfolgen. Die Möglichkeit einer Mobilfunk-basierenden Datenübertragung soll gegeben sein.
- Die Speicherung der Daten soll auf einem auswechselbaren, externen Datenträger erfolgen.
- Die Datenanalyse (z.B. Objekterkennung) soll mit gängigen Bibliotheken zum maschinellen Sehen erfolgen (wenn nicht in Echtzeit möglich, dann asynchron durch einen Queueing-Mechanismus).
- Das Versetzen des Systems in Energiesparmodus soll auch von der Software gesteuert möglich sein.
- Das System soll kompakte Maße (max. 25 cm x 25 cm) besitzen und preisgünstig sein (im Bereich zwischen 200€ und 300€).
- Das zu erstellende Software-Framework soll leicht erweiterbar sein.

Sensor	**Aufnahme**	**Triggerung**	**Anmerkung zur Umsetzung**
Audio	+	-	Einkanal-Aufnahme. Mindestens eine 16 Bit Auflösung.
Video	+	+	Triggerung durch benutzerdefinierte Algorithmen zur Objekterkennung. Mindestauflösung von 640 x 480, 10 FPS. Einzelbilder in der HD-Auflösung von 1280 x 720 Pixel. Kamera soll eine Weitwinkel-Optik (Bildwinkel im Bereich zwischen 75° und 120°) sowie optional einen Auto-Fokus besitzen.
Distanz	+	+	Umsetzung z.B. durch einen Ultraschallsensor.
Bewegung	+	+	Umsetzung z.B. durch einen Infrarot-Bewegungsmelder.
Helligkeit	+	+	Umsetzung z.B. durch einen Fotowiderstand.
Zeit	+	+	Durch den Zeitpunkt definierte Aufnahme.

Tabelle 1.1: Benötigte Sensoren

Anforderungen an die Stromaufnahme

- Die maximale Leistungsaufnahme soll so gering wie möglich sein, um eine autonome Benutzung mittels Akku/Batterie (bzw. Photovoltaik-Anlage) zu ermöglichen. Genaue Kennzahlen werden nach der Analyse verschiedener Hardwareplattformen festgelegt (siehe

Kapitel 3.1).

Anforderungen an die Software-/Hardwareplattform

- Multithreading-Unterstützung.
- Audioaufnahme in Echtzeit.
- Videoaufnahme in Echtzeit.
- Videoverarbeitung mittels gängiger Bibliotheken zum maschinellen Sehen (wenn nicht in Echtzeit möglich, dann durch einen Queueing-Mechanismus).
- Einstellbare Aufnahmezeiten (Start/Stopp) sowie einstellbare Intervalle der Sensoren (bei einer Videoaufnahme zusätzlich die Auflösung).
- Autonome Aufnahme bei der Auslösung durch einen Trigger-Mechanismus (z.B. durch den Bewegungsmelder).
- Hardware soll zusätzlich gesteuert in Ruhe- bzw. Sparmodus versetzt werden (z.B. CPU-Frequenz abhängig vom erwarteten Aufwand).
- Optional: Entfernte Konfiguration und Datenübertragung (z.B. Nachrichtenversand beim Auftreten eines Ereignisses).

Die Erfüllung (sowie Bewertung) der hier festgelegten Anforderungen wird in Kapitel 4 evaluiert.

1.4 Beitrag

In dieser Arbeit wird, wie in Kapitel 1.2 erläutert, die Entwicklung eines Überwachungssystems präsentiert, das neben den Standardaufgaben, wie Video-/Audioaufnahme und Speicherung von Sensordaten, auch eine Auswertung und Vorfilterung der Daten ermöglicht. Die durchgängige Konfigurierbarkeit (bzw. Auswahlmöglichkeit) der Trigger sowie der Aufnahmesensoren ist ein Alleinstellungsmerkmal dieses Systems. Bei der Auswahl der Komponenten, sowohl im Bereich der Hardware als auch Software, sollen vornehmlich Open-Source-Entwicklungen ver-

wendet werden, um in weiterer Folge die Erkenntnisse dieser Arbeit im Rahmen eines Gesamt-Frameworks zur Verfügung zu stellen.

Bestehende Komplettlösungen sind entweder nicht leistungsfähig genug, bieten keine Möglichkeit der Datenauswertung direkt in der Kamera, sprengen den angesetzten Budgetrahmen (beispielsweise Elphel [Elphel, Inc. 15c], Kosten ca. 800€), vor allem Kamerasysteme mit hoher Auflösung werden nicht mehr hergestellt bzw. unterstützt (LeanXcam [LeanXcam 08]) und unterstützen die Aufnahme mehrerer Sensoren nicht. Bei bestehenden Komplettmodulen für maschinelles Sehen mit eingebautem Webinterface fehlen außerdem zusätzliche Erweiterungsmöglichkeiten.

1.5 Aufbau und Organisation

Die Arbeit ist in fünf Kapitel gegliedert. In Kapitel 1 werden Motivation, Problemstellung und Anwendungsszenario erläutert. Es folgt eine Analyse verfügbarer Lösungen aus Forschung und Wirtschaft, das sind Komplettsysteme, die in Konkurrenz zu diesem Thema gesehen werden können, deren Vor- und Nachteile, eine Evaluierung der zur Verfügung stehenden Hardware-Entwicklungsplattformen sowie die Evaluierung der Software-Entwicklungsumgebungen im Bereich Bildverarbeitung und maschinelles Sehen.

In Kapitel 3 werden die Anforderungen detailliert aufgeführt sowie die Evaluierung und Auswahl geeigneter Hard- und Software bzw. Softwarebibliotheken durchgeführt. Weiters wird die Architektur der Software beschrieben und eine Beschreibungssprache definiert, die der Konfiguration des Systems dienen soll.

Das letzte Kapitel fasst die wesentlichen Ergebnisse im Hinblick auf definierte Zielsetzungen zusammen, erläutert Stärken und Schwächen des Systems und wirft abschließend einen Blick auf zukünftige Entwicklungen und mögliche Erweiterungen. Im Anhang ist neben den Verzeichnissen auch eine Dokumentation der Implementierung sowie das Application Programming Interface (API) zu finden.

Kapitel 2

Hintergrund und verwandte Arbeiten

2.1 Verwandte Arbeiten

Hier werden ähnliche Projekte bzw. Eigenentwicklungen mit ähnlichen Problemstellungen aufgezeigt. Ein Videoüberwachungssystem, das auf dem Prinzip der reinen Aufnahme ohne Auswertung (siehe Kapitel 1.1), jedoch auf einem eingebetteten System basiert, wurde beispielsweise von Li und Hao [Li 08] beschrieben. Dieses nutzt eine auf ARM-Prozessoren basierende Hardware sowie eine USB-Kamera zur Videoaquisition. Die Aufgabe der Datenanalyse bzw. Auswertung wird hier nicht berücksichtigt, sondern nur die Möglichkeit einer Aufnahme mit weniger leistungsfähigen ARM-Systemen (verglichen zu handelsüblichen PCs).

In der Arbeit von Chianese et al. [Chianese 15] ist ein Beispiel für ein Sensoren-Netzwerk, das auch für Überwachungszwecke verwendet werden kann, beschrieben. Hier wird ein auf BeagleBone Black basierendes System als Teil eines Sensor-Netzwerks mit multiplen Knoten benutzt. Es ist weder eine Video- bzw. Audioaufnahme noch eine Auswertung der Daten im Fokus. Viel mehr steht das "Internet of Things" und die Interaktion zwischen verschiedenen, bis jetzt üblicherweise eigenständigen, nicht vernetzten Komponenten im Vordergrund.

Der Ansatz eines Systems, das sowohl audiovisuelle Daten aufnimmt als auch Eingaben von weiteren Sensoren ermöglicht, ist die SenseCam [Hodges 06] (Abbildung 2.1). Diese wurde für "LifeLogging"-Zwecke entwickelt und in zahlreichen wissenschaftlichen Publikationen erwähnt[1]. Während die Aufnahme verschiedener Sensordaten möglich ist (Video, Audio, Beschleunigung),

[1] Siehe: http://research.microsoft.com/en-us/um/cambridge/projects/sensecam/publications.htm

ist die Möglichkeit der Auswertung der Daten sowie eine Konfiguration des Systems zu diesen Zwecken nicht vorhanden. Auch die Verwendung der SenseCam im Bereich der Tierüberwachung wurde nicht evaluiert.

Abbildung 2.1: Microsoft Research SenseCam

Als Beispiel für eine autonome Kamera mit Stromerzeugung aus Solarzellen und einer Kommunikationsschnittstelle über Mobilfunk ist die Arbeit von Jens Gabrikowski [Gabrikowski 08] nennenswert, jedoch werden hier nur Standbilder und keine Video-/Audioaufnahmen erzeugt. Das Ziel der Arbeit war, die autonome Überwachung von Flugplätzen mit der Möglichkeit einer externen Konfiguration mittels eines Mobiltelefons zu realisieren. Als Abgrenzung zu der in dieser Arbeit angestrebten Lösung, wurde neben der nur langsamen (Time-Lapse) Videoaufnahme auch keine Möglichkeiten zur Auswertung der aufgenommenen Daten vorgesehen.

2.2 Kommerzielle Systeme

Generell werden Systeme, die in den technischen Kontext eingebunden sind, als eingebettete Systeme (“Embedded Systems”) bezeichnet. Diese können auf Arbeitsplatzcomputer-ähnlicher Hardware basieren, wobei einige Komponenten leicht abgewandelt sind (beispielsweise werden Prozessoren ohne eine MMU (Memory Management Unit) verwendet). Als Betriebssystem wird häufig ein angepasstes Linux-Betriebssystem genutzt. Basierend auf eingebetteten Systemen sind bereits einige Komplettsysteme am Markt erhältlich, die die Anforderungen dieser Arbeit teilweise erfüllen. Diese Komplettsysteme und auch Entwicklungsumgebungen, die für das Erreichen des Ziels verwendet werden können, werden in diesem Kapitel vorgestellt.

Unter den kommerziellen Systemen sind vor allem LeanXcam und Elphel hervorzuheben, da diese die meisten Features abdecken und sowohl Software als auch Hardware unter Open-Source-Lizenz verfügbar sind. Beide Kamerasysteme ermöglichen eine Weiterverarbeitung der Daten auf der Kamera-Hardware selbst, da sie die Möglichkeit besitzen in C (bzw. C++) geschriebene Programme auszuführen. Ein weiteres kommerziell verfügbares System von der Firma Viseum, die "Intelligent Moving Camera" (IMC), besteht aus mehreren Kameras und verfügt über Objekterkennungs- bzw. Objektverfolgungsfunktionalitäten.

2.2.1 Viseum IMC

Bei Viseum IMC (Abbildung 2.2) handelt es sich um ein wetterfestes Außenbereich-Kamera-Array zur multivariaten Szeneerfassung, das für die Überwachung des öffentlichen Raums in England eingesetzt wird. Da das System aus mehreren Kameras besteht, ist es laut dem Hersteller möglich, einen Bereich zwischen 2 und 135 Metern in jede beliebige Richtung abzudecken [IMC 16]. Die Auswertung der Daten, beispielsweise eine Gesichtsdetektion oder Objektverfolgung, ist nicht "on-board" sondern mit einem externen Auswertesystem möglich, jedoch werden kaum technische Einzelheiten bzw. der Preis des Systems bekannt gegeben. Aus diesem Grund wird diese Kamera in der Hardware-Vergleichstabelle nicht angeführt (Tabelle 2.1).

Abbildung 2.2: Viseum IMC Kamera-Array

2.2.2 LeanXcam

Das Open-Source-System wurde speziell für den Bereich des maschinellen Sehens entwickelt und wurde im Jahr 2008 vorgestellt (siehe Abbildung 2.3). Seitdem wird es in verschiedenen Hochschulen bzw. Unternehmen verwendet. Die Kamera ist vorwiegend im Bereich Prozessüberwachung im Einsatz. Als Betriebssystem wird eine für eingebettete Systeme angepasste Linux-Umgebung verwendet - "µClinux". Die gängigen Algorithmen für maschinelles Sehen werden von der an die Kamera angepassten Bibliothek LeanCV zur Verfügung gestellt. Diese in C geschriebene Open-Source-Bibliothek unterstützt die gleichen Datentypen, die auch von der weitverbreiteten Open-Source-Bibliothek OpenCV genutzt werden. Somit ist es möglich, bestehende OpenCV-Entwicklungen auf der LeanXCam zu verwenden. Für die Erweiterbarkeit sind "General Purpose"-Eingänge und Ausgänge (GPIO) vorhanden. Die Leistungsaufnahme ist mit unter 500mA bei 5V angegeben, was einem Maximum von 2,5 Watt entspricht. Die Kamera besitzt eine Auflösung von 752 x 480 Pixel und ist imstande Aufnahmen mit 60 Bildern pro Sekunde aufzunehmen. Für Audioaufnahmen steht jedoch kein Mikrofon zur Verfügung [Zahn 14]. LeanXcam ist für 351€ erhältlich (Stand 28.02.2016), jedoch wurde die Entwicklung als auch Wartung der Kamera im März 2015 eingestellt [LeanXcam 15].

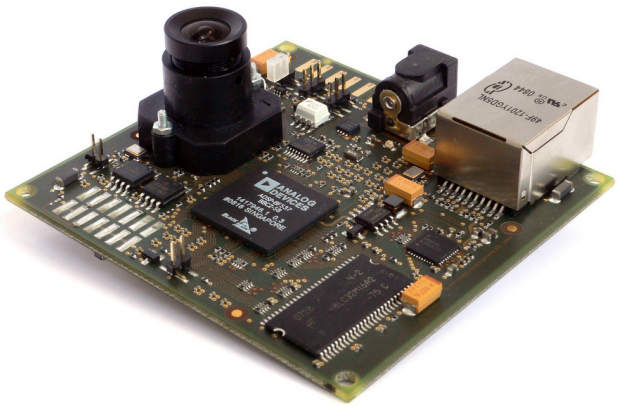

Abbildung 2.3: LeanXcam Open-Source Kamera

2.2.3 Elphel

Die autonomen Netzwerkkameras der Fa. Elphel (beispielsweise Modell 313 bzw. 353 [Elphel, Inc. 15b]) können ohne zusätzliche Verarbeitungseinheit (z.B. PC) für Aufnahmen sowohl mit sehr hoher Bildauflösung (2592 x 1936) als auch mit hoher Bildwiederholrate (240 FPS bei 320 x 240) eingesetzt werden. Für Überwachungsaufgaben mit Hochsicherheitsansprüchen, die eine hohe Auflösung erfordern, kann die Kamera dazu eingesetzt werden, um beispielsweise nur einen Teilbereich des aufgenommenen Bildes zu verwenden bzw. auch weit entfernte Objekte zu erkennen. Das Betriebssystem (GNU/Linux) bzw. die gesamte Software befindet sich auf dem internen Speicher und Aufnahmen können per Netzwerk an ein Subsystem geschickt werden. Als Verarbeitungseinheit wird hardwareseitig der "System-on-a-Chip" AXIS Etrax FS Dual-Core Prozessor verwendet, der sowohl eine CPU als auch einen Netzwerk-Kontroller beinhaltet [Axis Corporation 08]. Sowohl die Software als auch Aufnahme-Hardware sind unter Open-Source-Lizenz verfügbar. Um die Weiterverarbeitung der Daten auf der Kamera selbst durchzuführen, ist es möglich, in C geschriebene Applikationen direkt auf der Kamera laufen zu lassen. Die Stromversorgung wird aus dem LAN (gemäß IEEE 802.3af-Standard) bezogen. Je nach Spezifikation werden die Kameras zwischen 789€ und 4529€ angeboten (Stand 27.02.2016). Die Kamera verbraucht je nach Aufgabe (Ruhezustand, Aufnahme bzw. Aufnahme und Wiedergabe) zwischen 2,4 und 5,8 Watt [Elphel, Inc. 15a].

Abbildung 2.4: Elphel Model 353 Kamera

2.3 Mikrokontroller bzw. “System-on-a-Chip”-Komponenten

Es ist eine große Anzahl an Hardware-Kompontenten bzw. Hardwareplattformen vorhanden (über 120, siehe: [Wikipedia 16]) und zwar sowohl Komplett-Umgebungen aus dem Bereich des maschinellen Sehens als auch Hardware-Entwicklungsumgebungen, die für die Umsetzung der gewünschten Anforderungen (beispielsweise Verarbeitung der aufgenommenen Daten) verwendet werden können. Bei Komplett-Systemen bestehend aus einer Kamera (mit CMOS-Sensor bzw. Linse) und einer zusätzlichen Prozessoreinheit, um einfache Erkennungsalgorithmen auszuführen, ist die CMUcam zu nennen. Weitere Hardwareplattformen können entweder auf einem Mikrokontroller (z.B. Arduino) oder einem Komplett-System-Chip-basierenden PC-ähnlichen System (“System-on-a-Chip” oder auch als “Single-Board Computer” bezeichnet), beispielsweise BeagleBone oder Raspberry Pi, basieren. Während es für Mikrokontroller-basierende Systeme eine hohe Anzahl an Erweiterungsplatinen gibt, sind die Erweiterungsmöglichkeiten bei “System-on-a-Chip”- (SoC) Systemen spärlicher. Generell haben Mikrokontroller-basierende Systeme eine deutlich geringere Leistung sowie eine niedrigere Stromaufnahme. Die SoC-Systeme enthalten nahezu alle Hard- und Software-Komponenten eines Linux-Rechners, wie beispielsweise ein Betriebssystem, einen USB-Anschluss oder eine GPU (Graphics Processing Unit) mit HDMI-Anschluss (Raspberry Pi). Folgend werden vielversprechende Plattformen vorgestellt.

2.3.1 CMUcam

CMUcam (CMUcam Version 5 - Pixy) stellt eine Software- bzw. Hardware-Enwicklungsumgebung zur Verfügung, die speziell für den Bereich des maschinellen Sehens entwickelt wurde und seit August 2014 auf dem Markt ist. Sie ist aus der Partnerschaft zwischen dem Robotik-Institut der Carnegie Mellon Universität (US) und der Fa. Charmed Labs entstanden [Rowe 07]. Die CMUcam-Hardware besteht aus einem CMOS-Sensor und einem ARM-Cortex-M4-basierenden Dual-Core Mikrokontroller. Die Software ist unter Open-Source-Lizenz verfügbar und auch die Hardware-Stückliste sowie Schaltung stehen für einen Nachbau zur Verfügung. Das Besondere an der CMUcam ist, dass diese eine Detektion der Objekte selbst durchführen und diesen Output weitergeben kann. Eine Detektion der Objekte ist nur über deren Farbe bzw. über sogenannte Farbsignaturen (“color codes”) möglich. Das bedeutet, dass zwei zuvor festgelegte, nebeneinander auftre-

tende Farben als ein Objekt zusammengefasst werden können. Es können sieben verschiedene Farben (Objekte) mit einer Bildwiederholrate von 50 Bildern pro Sekunde erkannt werden. Bei der Verwendung von Farbsignaturen sind es über hundert verschiedene Objekte. Andere Algorithmen, zum Beispiel zur Gesichtserkennung, sind noch nicht verfügbar. Aufgrund der relativ leistungsarmen, dafür aber preisgünstigen Hardware ist die Entwicklung neuer Algorithmen nur schwer möglich und eine Verwendung der frei verfügbaren Bibliotheken für maschinelles Sehen, wie beispielsweise OpenCV, nicht möglich. Der Stromverbrauch der CMUcam wurde mit 140mA bei 5V (0,7 Watt) angegeben. Eine Kombination mit einer stromsparenderen Mikrokontroller-basierenden Hardwareplattform (beispielsweise Arduino) ist möglich, um neben der Aufnahme und Erkennung (um die sich die CMUcam bereits kümmert) die Erweiterbarkeit (z.B. Audioaufnahme) zu gewährleisten. Die aktuelle Version der CMUcam kostet 58€ (Stand 27.02.2016).

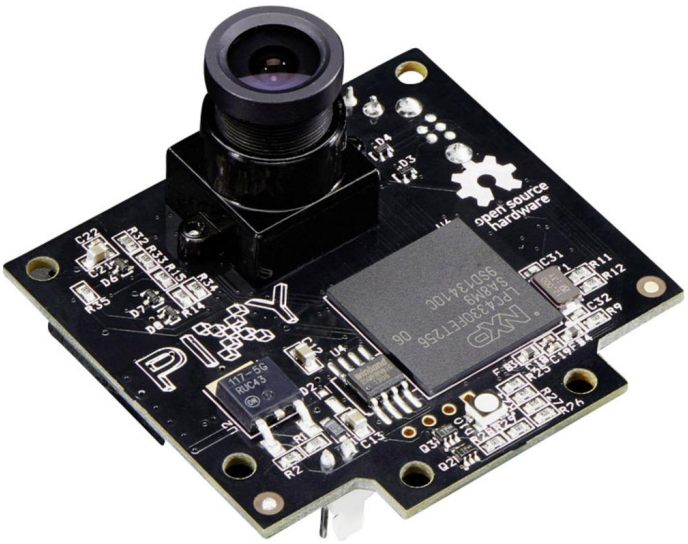

Abbildung 2.5: CMUcam 5 - Pixy

2.3.2 Arduino

Die bekannteste Mikrokontroller-basierende Entwicklungsumgebung ist Arduino. Diese basiert auf Atmel-Mikrokontrollern und wurde das erste Mal 2005 von den zwei Entwicklern Massimo Banzi und David Cuartielles der Open-Source-Community vorgestellt. Die Taktgeschwindigkeit beträgt je nach Modell typischerweise zwischen 8 und 16 MHz und alle Modelle (mittlerweile über zwanzig verschiedene) können mit einer Spannung von 5V betrieben werden. Die Hauptintention der Arduino-Hardware ist die Steuerung von I/O-Pins. Arduino ist vor allem für Mikrokontroller-Einsteiger und kleinere bis mittelgroße Projekte geeignet. Die Programmie-

rung erfolgt über die eigene Arduino-IDE und der damit erzeugte Binärcode kann über die IDE auf den Mikrokontroller übertragen werden. Es gibt eine sehr hohe Anzahl an Erweiterungsplatinen (sogenannte “Shields” - beispielsweise für die Motorsteuerung oder Datenspeicherung auf SD-Karten, siehe [Oxer 16]). Eine externe Bibliothek für maschinelles Sehen (z.B. OpenCV) ist zwecks geringer Leistungsfähigkeit nicht vorhanden.

Mittlerweile wird seit Ende 2013 auch ein Linux-basierendes System angeboten. Dieses gilt als das teuerste Arduino und wird um 77€ angeboten. Die Leistungsaufnahme der reinen Mikrokontroller-Boards ist jedoch um Faktor 10 geringer (ca. 200 mW). Die ursprünglichen Arduinos gibt es bereits ab 20€ bzw. Arduino Mini und Nano sind bereits um ca. 10€ erhältlich.

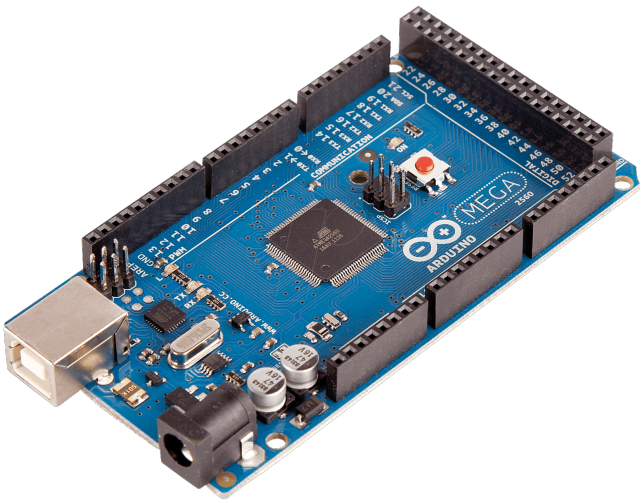

Abbildung 2.6: Arduino Mega 2560

2.3.3 Raspberry Pi

Das Raspberry Pi ist ein SoC-Rechner, der von der Raspberry Pi Foundation entwickelt wurde und seit Anfang 2012 am Markt erhältlich ist. Die Hardware besteht je nach Modell (bisher sechs) aus den Broadcom BCM2835 und BCM2836 SoC-Chips [Broadcom Europe Ltd 12], die die meisten Komponenten stellen und zwar unter anderem CPU (Computing Processing Unit), USB-Kontroller, GPU (Graphics Processing Unit), Netzwerk-Kontroller und Audio-Kontroller. Als Betriebssystem kommen zumeist angepasste Linux-Distributionen, beispielsweise “Embedded Linux” (eLinux), zum Einsatz. Beim Stromverbrauch werden je nach Last und Modell Werte zwischen 100mA und 800mA bei 5V angegeben (0,5-4 Watt). Die aktuelle Version 2 Modell B wird für 42€ angeboten. Seit 2013 ist ein eigenes Kamera-Modul für die Videoaufnahme erhältlich.

Für die Audioaufnahme kann ein Zusatzmodul bzw. eine USB-Soundkarte angeschafft werden. Da es sich um ein herkömmliches Linux-basierendes System handelt, kann die Softwareentwicklung in einer Vielzahl von Programmiersprachen vorgenommen werden, beispielsweise C++ oder Python.

Abbildung 2.7: Raspberry Pi Rev. B

2.3.4 BeagleBone / BeagleBone Black

Das BeagleBone ist auch ein SoC-Rechner, der auf dem OMAP3530-SoC [Tex 13] von Texas Instruments basiert und Mitte 2011 auf den Markt gebracht wurde. Wie Raspberry Pi, wird dieses mit einer für SoC-Systeme angepassten Linux-Version ausgeliefert. Jedoch hat das BeagleBone neben den üblichen Ein- bzw. Ausgängen (GPIO), die für Steuerungszwecke verwendet werden können, wesentlich mehr Hardware-Funktionalitäten, wie beispielsweise einen Hardware-Timer, I^2C-Bus, CAN-Bus sowie auch Analog-Eingänge (mit 12 Bit Analog/Digital-Wandlern) [Coley 13]. Für eine Audio-Verarbeitung (Wiedergabe/Aufnahme) kann außerdem eine eigene Erweiterungsplatine (Cape) erworben werden sowie ein Kamera-Cape für Videoaufnahmen. Über USB können sämtliche USB-Kameras angeschlossen werden. Das aktuelle BeagleBone Black wird für 62€ angeboten (Stand 27.02.2016). Auch bei BeagleBone bzw. BeagleBone Black ist bei der Softwareentwicklung eine Auswahl zwischen verschiedenen Programmiersprachen möglich.

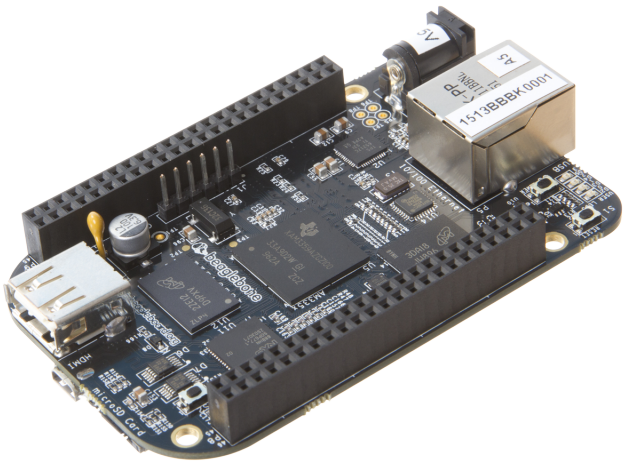

Abbildung 2.8: BeagleBone Black

2.3.5 Mehr-Kern SoC-Systeme

Um eine Vorselektion von Videodaten durchzuführen, sind Bibliotheken zur maschinellen Bilderkennung notwendig. Als “State-of-the-Art”-Methode für zahlreiche Anwendungen aus der maschinellen Bilderkennung (im Bereich der Klassifizierung und Erkennung) gelten Systeme mit Unterstützung der Grafikkarten-Programmierung, wie beispielsweise der Tegra K1 Chip mit CUDA-Unterstützung. Was die Leistungsaufnahme von solchen Systemen betrifft, wurden zum Beispiel bei dem von Professor Andrew Ng ins Leben gerufenen “Google Brain”-Projekt noch 1000 Server und 600 kWatt benötigt, um eine maschinelle Bilderkennung mittels neuralen Netzwerken (“Deep Neural Networks”) durchzuführen [Le 13] [Matsugu 03] [Krizhevsky 12]. Mittlerweile gibt es die “Low-Power Image Recognition”-Herausforderung[2], bei der diese Aufgabenstellungen mit einem Tegra K1, 12,5 Watt Prozessor bewältigt werden sollen. Systeme mit Unterstützung der Grafikkarten-Programmierung werden in Unterkapitel 2.3.5 vorgestellt.

Um die Rechenleistung zu erhöhen (z.B. um komplexe Erkennungsalgorithmen durchführen zu können), können neben Grafikprozessoren auch benutzerprogrammierbare, integrierte Schaltkreise (“Field Programmable Gate Array”, kurz FPGAs) verwendet werden. Grafikprozessoren (GPU) können dort effizient angewendet werden, wo (neben anderen Bedingungen) Berechnungen stark parallelisiert werden können. Für die Implementierung der Algorithmen kann CUDA (Technologie wodurch Programmteile durch den Grafikprozessor abgearbeitet werden können) von der Fa. Nvidia verwendet werden [Nvidia 08]. Als Alternative bietet sich OpenCL an, eine

[2] Low-Power Image Recognition Challenge (LPIRC), erstmals 2015 abgehalten - http://www.lpirc.net

Schnittstelle für uneinheitliche Parallelrechner, die neben Grafikprozessoren auch digitale Signalprozessoren ansprechen kann. Einige OpenCV-Algorithmen sind bereits in OpenCL implementiert. Auch bei CUDA werden angepasste OpenCV-Bibliotheken zur Verfügung gestellt. Bei den SoC-Entwicklungsumgebungen ist vor allem das "Nvidia Jetson TK1 DevKit" zu nennen, das aus einem Tegra K1-SoC mit 192 (CUDA-fähigen) Kernen sowie einer 4-Plus-1 Quad-Core ARM Cortex-A15 CPU besteht (Leistungsaufnahme unter 10 Watt) [NVIDIA Corporation 16]. Bei diversen Benchmark-Tests mit "State-of-the-Art"-Hardware konnten Berechnungen auf der GPU 10 bis 27 mal schneller durchgeführt werden [Rajan 14].

Bei den FPGAs werden Schaltungsstrukturen mittels Hardware-Beschreibungssprachen (Hardware Description Language) erstellt und nachfolgend als Konfiguration in den Baustein übertragen. Dadurch werden bestimmte Schalterstellungen aktiviert bzw. deaktiviert, was in Folge eine konkret implementierte digitale Schaltung ergibt. Bei den eingebetteten Systemen gibt es Werkzeuge, die eine Konstruktion auf Funktionsblockebene anbieten, z.B. Xilinx EDK (Embedded Development Kit). Funktionsblöcke, wie FIFOs, Prozessoren, serielle Schnittstellen, Ethernet-MAC-Layer, RAM-Controller, Parallel-IO etc., werden vom Hersteller zur Verfügung gestellt. Unter den FPGA-SoC-Systemen ist die "Parallella-16 Embedded Platform" mit 2 CPUs und 16 FPGA-Kernen (Leistungsaufnahme unter 5 Watt) von der Firma Adapteva nennenswert.

2.4 Vergleich unterschiedlicher Hardwareplattformen

Die in diesem Kapitel vorgestellten Plattformen werden in Tabelle 2.1 gegenübergestellt. Die Rechenleistung wird hierbei in Relation zur Leistungsaufnahme verglichen, um auch im Hinblick auf einen autonomen Betrieb mit alternativer Stromversorgung eine Abschätzung machen zu können. Obwohl die CPU in einem System nicht alleine für die Leistungsaufnahme verantwortlich ist, kann diese üblicherweise 50-60% der Leistung beanspruchen. Für den Vergleich diverser Systeme bzw. CPUs kann die Kennzahl "Leistung per Watt" (Performance per Watt - PPW) herangezogen werden [Huppler 13]. Jedoch sind solche Vergleiche alleine nicht aussagekräftig, da auch die Prozessorarchitektur berücksichtigt werden muss [Roberts-Hoffman 09]. Generell wird bei hohem Rechenbedarf bei möglichst wenig Stromverbrauch die Verwendung der CMP-Technologie (Mehr-Kern-Prozessoren) [Barroso 05] empfohlen, wie es beispielsweise beim Raspberry Pi 2 Model B (erhältlich seit Februar 2015) der Fall ist. Für den Vergleich

verschiedener Systeme kann auch die “Embedded Microprocessor Benchmark Consortium”-(EEMBC) Datenbank genutzt werden [EEMBC 16] [Poovey 09]. Hier sind allerdings nicht alle Systeme erfasst (Stand 27.02.2016).

System	Kamera[a]	Netzwerk	Audio[b]	Betriebs-system	CV Bibliothek	Prozessor	Anzahl Kerne	Leistungs-aufnahme	PPW[c]	CoreMark[d]	Kosten
LeanXcam	+	+	- / -	Linux	LeanCV	ADSP BF-537	1	2,5W	213	-	351€
Elphel 353	+	+	- / -	vmtl. Linux	-	AXIS ETRAX FS	1	2,4 - 5,8W	59	-	789€
CMUcam 5	+	-	- / -	n/a	vordefiniert	NXP LPC4330	2	0,7W	-	-	58€
Arduino Mega 2560	- / +	- / +	- / -	n/a	-	ATmega2560	1	0,2W	80	0.53	20€
Raspberry Pi	- / +	- / +	- / +	Linux	OpenCV	ARM1176JZF-S	1	1,5W	565	1.86	29€
Raspberry Pi 2 (B)	- / +	+	+ / +	Linux	OpenCV	ARM Cortex-A7	4	4W	417[e]	-	42€
BeagleBone	- / +	+	- / -	Linux	OpenCV	ARM Cortex-A8	1	1,5 - 2,5W	384	2.42	87€
BeagleBone Black	- / +	+	- / -	Linux	OpenCV	ARM Cortex-A8	1	1,05 - 2,3W	694	-	47€

Tabelle 2.1: Vergleich der Komplettsysteme bzw. Entwicklungsumgebungen

[a] Im System bzw. erweiterbar
[b] Audioaufnahme / Wiedergabe
[c] MIPS / Arithmetisches Mittel zwischen Min. und Max. des Leistungsverbrauchs [W]
[d] EEMBC CoreMark per MHz
[e] Angabe nur für einen Kern, bei mehreren Kernen ist das Amdahlsche Gesetz zu beachten

2.5 Sensorik

Alle Komponenten mit denen bestimmte physikalische Eigenschaften einer Umgebung (z.B. optisch durch den CCD/CMOS-Sensor oder durch die Messung des Schallwechseldrucks durch ein Mikrofon) qualitativ oder als Messgröße quantitativ erfasst werden können, können als Sensoren bezeichnet werden. Die meisten Hardware-Plattformen können üblicherweise mit verschiedenen Sensoren bzw. Komponenten (Shields/Capes) erweitert werden. Die hier vorgestellten Komponenten sind für die Arbeit von besonderem Interesse.

2.5.1 Video

Für Videoaufnahmen gibt es neben Modulen, die speziell für das entsprechende System entwickelt worden sind, auch die Möglichkeit eine USB-Kamera zu verwenden (SoC-Lösungen mit USB-Anschluss). Bei beiden Varianten werden üblicherweise CMOS-Sensoren, zumeist von der Firma OmniVision, bis ca. 5 Megapixel verwendet. Die Videoaufnahme wird meist bis zu einer Full-HD-Auflösung (1920 x 1080 Pixel) unterstützt.

Bei Arduino-Systemen gibt es neben der CMUcam, die bereits in diesem Kapitel vorgestellt wurde, auch noch Kamera-Module des Herstellers ArduCAM [ard 16]. Diese können sowohl aus einer als auch mehreren Autofokus-Kameras (OmniVision CMOS-Sensoren zw. 0,3 und 5 Megapixel) bestehen. ArduCAM-Module sichern bei Arduino-Systemen die Daten über den SPI-Bus direkt auf der SD-Karte. Um die Daten auszuwerten, ist die Rechenleistung eines Arduino Mikrokontroller-Boards nicht ausreichend.

Für Raspberry Pi gibt es ein offizielles "Raspberry Pi Camera Module[3]", das ebenfalls auf einem 5 Megapixel OmniVision CMOS-Sensor (OV5647) basiert [Hughes 13]. Die Kamera besitzt einen fixen Fokus mit einer Brennweite von 3,6 mm (entspricht 8 mm bei Kleinformat) und unterstützt Hardware-beschleunigte Aufnahmen im H.264-Codec. Für BeagleBone bzw. BeagleBone Black ist ebenfalls ein Kamera-Cape vorhanden. Dieses verwendet einen 3,1 Megapixel CMOS-Sensor der Firma Aptina [CircuitCo 16].

Die Verwendung einer USB-Kamera (beispielsweise C920 von der Fa. Logitech) ist mit allen Systemen, die einen USB-Anschluss besitzen (alle Raspberry Pi bzw. BeagleBone-Systeme) möglich.

[3] Siehe FAQs: https://www.raspberrypi.org/help/faqs/

Die Vorteile liegen darin, dass die Stromaufnahme hier laut USB 2.0 Protokoll 500mA nicht übersteigen darf [Compaq 00] und diese neben Full-HD Video- bzw. Fotoaufnahmefunktionen, wie z.B. Autofokus, auch Hardware-Codierung (H.264 oder MJPEG) und Audioaufnahmefunktionen bieten (sofern ein Mikrofon verbaut ist).

2.5.2 Audio

Wie bei Videoaufnahmen kann auch bei Audioaufnahmen auf eine Vielzahl von Lösungen zugegriffen werden. Bei Arduino-Systemen kann das Audio-Shield, das üblicherweise nur für die Wiedergabe entwickelt wurde, auch für Aufnahmen verwendet werden [Adafruit 15a] [WaveRP 16].

Weiters kann eine vereinfachte Aufnahme über die analogen Eingänge des Arduino erreicht werden [Patterson 15]. Bei den Linux-basierenden SoC-Systemen kann beispielsweise ein USB-Mikrofon bzw. eine USB-Soundkarte [g7smy.co.uk 13] oder eine speziell dafür entwickelte Erweiterung verwendet werden. Hier gibt es zum Beispiel bei BeagleBone das Audio-Cape [CircuitCo 14] bzw. bei Raspberry Pi die "Wolfson Audio Card" [Wol 14]. Zumindest eine Einkanalaufnahme mit einer Sample-Rate bis mindestens 19 KHz ist mit allen Lösungen erreichbar.

2.5.3 Distanz

Generell können Sensoren, die auf dem Prinzip der Laufzeitmessung oder Triangulation basieren, für Distanzmessungen verwendet werden. Bei der Laufzeitmessung wird davon ausgegangen, dass sich elektromagnetische und akustische Wellen mit bekannter Geschwindigkeit ausbreiten. Wenn ein Signal von einem Messobjekt reflektiert wird, kann aufgrund der Laufzeit die Entfernung zwischen zwei Punkten bestimmt werden. Beispielsweise gibt es Ultraschallsensoren, die auf diesem Prinzip basieren. Diese sind relativ günstig und können mit den meisten Entwicklungsumgebungen verwendet werden (beispielsweise HC-SR-04 [elecfreaks.com 15]). Jedoch liegt die maximal zuverlässig messbare Distanz bei ca. sechs Metern. Für Anwendungen mit größerer Distanz zum Messobjekt kann auf Laufzeitmessung mittels Laser oder Lasertriangulation zurückgegriffen werden. Hier liegt der Messbereich zwischen einem Meter und einigen Kilometern. Die Kosten solcher Sensoren (im Vergleich zu Ultraschallsensoren) sind jedoch um Faktor 20-25 höher. Beispielsweise kostet der "LIDAR-Lite v2"-Lasersensor ca. 105€ und soll für Entfernungen bis 40 Metern genutzt werden können [Pul 15]. Die Verwendung dieser Sensoren

ist ebenso mit allen Hardware-Plattformen, beispielsweise über den I^2C-Bus, möglich.

2.5.4 Bewegung

Bewegung kann aktiv mit elektromagnetischen Wellen, Ultraschall oder anhand von Infrarotstrahlung der Umgebung erkannt werden. Am häufigsten werden passive Sensoren verwendet, die Infrarotstrahlung registrieren (PIR-Sensoren). Hierbei wird die Wärmestrahlung vom Objekt (z.B. Mensch, Tier, Kraftfahrzeug) im mittleren Infrarotbereich gemessen. Der Erfassungsbereich von einem PIR-Sensor liegt je nach Typ bei ca. 40 Metern bei einem Einstrahlwinkel von 180°. PIR-Sensoren sind preiswert und können in allen Entwicklungsumgebungen ausgelesen werden. Zumeist geschieht dies durch das GPIO. Üblicherweise können sowohl Einschaltdauer als auch Umgebungshelligkeit (Hell-Dunkel-Grenze) durch zwei Widerstände (Potentiometer) festgelegt werden [Par 14].

2.5.5 Helligkeit

Fotodetektoren (Fotowiderstände) bzw. Lichtsensoren wandeln Licht unter Benutzung des fotoelektrischen Effekts in Spannung, Strom oder Frequenz um, welche dann weiterverarbeitet werden kann. Ein einfacher Fotowiderstand ändert je nach Lichteinstrahlung seinen Wert. Die durch den Effekt entstandene Spannung bzw. Stromveränderung kann beispielsweise durch analoge Eingänge des Arduinos bzw. BeagleBones registriert werden. Bei Entwicklungsumgebungen, die über keine analogen Eingänge verfügen, kann auch ein “Licht zu Frequenz-Umwandler” (z.B. TSL235 [Tex 94]) verwendet werden, da hier die Frequenzänderung durch die digitalen GPIO-Ports abgefragt werden kann.

2.5.6 Echtzeituhr

Da die meisten Entwicklungsumgebungen über keine Echtzeituhr, sondern nur über logische Uhren verfügen, ist ein Modul notwendig, das als physikalischer Zeitgeber fungiert. Solche Module ermöglichen es auch, die Uhrzeit beim Ausschalten des Systems zu erhalten und können beispielsweise über das I^2C-Protokoll angesprochen werden. Üblicherweise kann neben der Uhrzeit auch das Datum gesetzt sowie ausgelesen werden (siehe Datenblatt vom MI-DS1307-Modul

[Max 15]). Zur Synchronisation der Uhr im System kann bei Verfügbarkeit einer Netzwerk-Verbindung das “Network Time Protocol” (NTP) verwendet werden.

2.6 Entwicklungsumgebungen und Bibliotheken

Analog zur Hardware, gibt es auch bei der Wahl des Betriebssystems, der Programmiersprache, der Programmbibliotheken mit Algorithmen für Bildverarbeitung und maschinelles Sehen oder der möglichen Entwicklungsumgebungen eine Vielzahl an Auswahlmöglichkeiten, die in diesem Unterkapitel vorgestellt werden.

Neben den populären bzw. meistgenutzten Bibliotheken, wie z.B. OpenCV bzw. deren Abwandlungen und Wrapper (JavaCV, SimpleCV etc.), werden hier auch weniger bekannte Bibliotheken (FastCV, CCV) behandelt. Neben den wichtigsten Kriterien, wie Benutzbarkeit, Funktionalität und Zuverlässigkeit, wird vor allem die Lauffähigkeit auf “Embedded Systems” sowie die für die Softwareentwicklung wichtigen Aspekte, Dokumentation der API oder Debugging-Möglichkeiten, evaluiert. Wie aus der Umfrage der UBM Tech Electronics “Embedded Market Study” aus den Jahren 2010 bis 2014 unverändert herausgeht, ist gerade die Nutzung bzw. Einfachheit der Debugging-Möglichkeiten bei den eingebetteten Systemen der Punkt mit dem größten Verbesserungspotential. Weiters geht aus der gleichen Studie hervor, dass die Verfügbarkeit der Softwareentwicklungswerkzeuge noch vor der Leistungsfähigkeit des Prozessors maßgebend für dessen Auswahl ist [UBM Tech 14]. Für eine umfangreiche Softwarebewertung können beispielsweise Unterscheidbarkeitskriterien nach den ISO/IEC 9126-1 bzw. ISO/IEC 25010:2011 Standards (siehe Software-Engineering-basierende Checkliste [Jackson 11]) herangezogen werden. Die in diesem Kapitel vorgestellten Lösungen sowie deren Vor- und Nachteile wurden anhand einer Literaturrecherche ermittelt.

2.6.1 Bibliotheken für maschinelles Sehen

Bibliotheken für maschinelles Sehen können je nach Anwendungsgebiet grob in drei Kategorien eingeteilt werden:

- Bibliotheken, die hauptsächlich für Algorithmenentwicklung bzw. numerische Berechnungen verwendet werden (beispielsweise MATLAB) bzw. für robusten, nahezu system-

unabhängigen Produktiveinsatz geeignet sind (beispielsweise OpenCV).

- Bibliotheken, die auf Performance in einer bestimmten Umgebung optimiert sind (beispielsweise FastCV).
- Bibliotheken, die aus mehreren solchen bestehen und nicht ausschließlich für maschinelles Sehen verwendet werden (beispielsweise liegt der Fokus bei Processing und openFrameworks auf audiovisueller Kunst).

OpenCV

Die heutzutage bedeutendste Bibliothek für maschinelles Sehen ist OpenCV (CV - kurz für Computer Vision) [Bradski 00]. Sie wurde 1999 auf Initiative von "Intel Research" ins Leben gerufen. Als Hauptziel wurde die Entwicklung einer Bibliothek, die nicht nur die neuesten CV-Algorithmen, sondern auch einen optimierten Quellcode zur Verfügung stellt, definiert. Weitere Ziele waren die Wissensvermittlung bzw. Bereitstellung einer Infrastruktur, um eine Weiterverbreitung und leichtere Lesbarkeit von neuen Algorithmen bzw. darauf basierenden Applikationen zu ermöglichen. OpenCV ist plattformunabhängig auf Open-Source-Basis verfügbar. Weiters können Applikationen, die auf Basis von OpenCV verwirklicht wurden, auch kostenpflichtig vertrieben werden. Die größten Stärken von OpenCV liegen in der Geschwindigkeit und der großen Menge an Algorithmen, die relativ schnell aus neuesten Forschungsergebnissen in die Bibliothek aufgenommen werden. OpenCV ist in C++ geschrieben und auch die Basis-Programmierschnittstelle ist in C++ gehalten, allerdings sind in der Version 2.5 Wrapper, sowie Online-Dokumentationen für Python, Java sowie MATLAB verfügbar. Weitere Wrapper für Programmiersprachen wie C#, Perl, Ch und Ruby sind mittlerweile auch verfügbar, um die Plattform einer breiteren Zielgruppe zugänglich zu machen. Um eine bessere Performance zu erreichen, werden manche Algorithmen seit der Version 2.4.3 auch in OpenCL (siehe Kapitel 2.3.5) zur Verfügung gestellt.

JavaCV

JavaCV benutzt sowohl OpenCV als auch Wrapper von anderen Bildverarbeitungsbibliotheken, wie beispielsweise FFMPEG [Bellard 12] oder OpenKinect [Blake 11]. Diese auf Java basierende Bibliothek kann somit als Basis für Medienverarbeitung bzw. maschinelles Sehen in Java ver-

wendet werden. Von Seiten der Open-Source-Community wird jedoch bemängelt, dass die Dokumentation nicht vollständig ist bzw. nur wenige Implementierungsbeispiele vorhanden sind [Audet 11].

FastCV

Die FastCV-Bibliothek stellt für mobile Geräte eine optimierte Bibliothek für maschinelles Sehen zur Verfügung. Es werden auf ARM-Prozessoren basierende Systeme effizient unterstützt, jedoch ist die Bibliothek besonders für den Qualcomm Snapdragon Prozessor optimiert. Somit können komplexe Algorithmen für Gestenerkennung, Gesichtserkennung oder Texterkennung durch Verwendung der Hardwarebeschleunigung schneller ausgeführt werden [Qua 14].

SimpleCV

SimpleCV ist ebenfalls eine Open-Source-Bibliothek für maschinelles Sehen, die auf OpenCV aufbaut und für die Programmiersprache Python zur Verfügung steht [Demaagd 12]. Diese Bibliothek unterstützt die allgemeinen Aufgaben der Bildaquisition, Bildbearbeitung sowie Merkmalserkennung und kann somit auch für die Erstellung der "Proof-of-Concept"-Lösung verwendet werden. Als Anwendungsgebiet ist schnelles Prototyping beschrieben, das selbst für Einsteiger geeignet ist.

C-based Computer Vision Library

Als relativ neue Alternative zu OpenCV ist CCV zu nennen. Hierbei handelt es sich um eine "Applikationsgesteuerte" Bibliothek, die nur wenige, vom Anwendungsgebiet abhängige, moderne Algorithmen beinhaltet [Liu 10]. Beispielsweise ist hiermit die Klassifizierung mittels neuronalen Netzwerken oder auch Objekterkennung mit unterschiedlich trainierbaren, teilbasierten Modellen möglich [Felzenszwalb 10]. Die Bibliothek kann auf den meisten Unix-basierenden Systemen betrieben werden und ist beispielsweise für das Raspberry Pi vorkompiliert verfügbar.

MATLAB

Matlab ist eine Entwicklungsumgebung, die auf mathematischen Methoden basiert [Moler 80]. Diese wird primär für numerische Berechnungen mithilfe von Matrizen sowie Entwicklungen neuer Algorithmen verwendet. Für die Verwendung im Bereich des maschinellen Sehens steht das "Computer Vision System Toolbox" zur Verfügung [Inc. 16]. Die Hauptmerkmale in diesem Bereich sind unter anderem Merkmalerkennung, Extraktion und Abgleich, Objekterkennung (mit Viola-Jones [Viola 01] und trainierten Detektoren), Objektverfolgung und Bewegungsschätzung. Für die Verwendung im Bereich der eingebetteten Systeme bietet Matlab Unterstützung für das Raspberry Pi[4].

openFrameworks

openFrameworks ist nicht ausschließlich für maschinelles Sehen gedacht, sondern vor allem als eine Ansammlung verschiedener Bibliotheken (Frameworks) für einen breiteren Gebrauch (in C bzw. C++). Unter den verwendeten Bibliotheken sind beispielsweise OpenGL, PortAudio, Quicktime und OpenCV zu nennen. Die Portierbarkeit spielte bei der Erstellung des openFrameworks eine sehr wichtige Rolle. Somit ist eine Verwendung mit den meisten Betriebssystemen und Software-Entwicklungsumgebungen, beispielsweise Windows, OSX, Linux, iOS oder Android, möglich. Auch eine Verwendung mit Raspberry Pi, BeagleBone sowie ähnlichen auf ARM-Prozessoren basierenden Systemen ist möglich. Die von openFrameworks intendierte Zielgruppe ist im Bereich kreativer, künstlerischer Darstellung mit Verwendung eines Low-Level-Zugriffs auf die in Medien enthaltenen Daten tätig [Lieberman 05].

Processing

Auch Processing wird hauptsächlich im Bereich der audiovisuellen Kunst verwendet. Anders als bei openFrameworks wird hier jedoch keine Low-Level Programmiersprache (z.B. C/C++) sondern Java bzw. JavaScript verwendet. Auch Processing ist auf ARM-Prozessoren basierenden Systemen lauffähig. Seit der Version 3.0 ist der Hardware-Zugriff auf die GPIO-Schnittstelle des Raspberry Pi möglich [Adafruit 15b].

[4] Siehe: http://de.mathworks.com/hardware-support/raspberry-pi-simulink.html. Nur für von MATLAB in C-übersetzbare Funktionen möglich.

2.6.2 Zusammenfassung

Eine kurze Zusammenfassung der Bibliotheken sowie Möglichkeiten zu deren Verwendung ist in Tabelle 2.2 angeführt. Für weiterführende Literatur zu openFrameworks und Processing (auch in Verwendung mit eingebetteten Systemen) ist das Buch von J. Noble, “Programming Interactivity” [Noble 09], zu empfehlen.

Bibliothek	**Sprache**	**Bereich**	**Besonderheit**	**Verfügbarkeit**
OpenCV	C, C+, Java, Python	Appl.-Entw.	Breite Anwendbarkeit	Open Source
JavaCV	Java	Appl.-Entw.	Hardware-Beschleunigung	Open Source
FastCV	C, C++	Appl.-Entw.	Optimiert für Snapdragon	Proprietär
SimpleCV	Python	Prototyping	Einfache Bedienung	Open Source
CCV	C, C++	Appl.-Entw.	Wenige moderne Algorithmen	Open Source
MATLAB	MATLAB	Forschung	Breite Anwendbarkeit	Proprietär
openFrameworks	C, C++	AV-Kunst	Portierbar	Open Source
Processing	Java	AV-Kunst	Einfache Bedienung	Open Source

Tabelle 2.2: Vergleich der Bibliotheken sowie deren Anwendung

Kapitel 3

Ansatz und Implementierung

In diesem Kapitel werden die Anforderungen an das zu entwickelnde System detailliert aufgeführt und diskutiert sowie basierend auf einer Vorevaluierung die Auswahl der Hardware-Plattformen und der verwendeten Software (bzw. Softwarebibliotheken) und Hardware durchgeführt. Neben der Auswahl der Komponenten werden die Architektur der Hardware bzw. Software sowie die grundlegenden Prozessabläufe definiert. Wie bereits in Kapitel 2 ersichtlich, ist hier eine große Anzahl an Auswahlmöglichkeiten vorhanden. Weiters wird eine Beschreibungssprache definiert, die zur Konfiguration des Systems dienen soll.

3.1 Anforderungen

Die Mindestanforderungen wurden bereits in Kapitel 1 vorgestellt. Durch die Evaluierung der vorhandenen Systeme bzw. der zur Verfügung stehenden Hardware-Plattformen, werden hier endgültige Anforderungen an das System bzw. an die “Proof-of-Concept”-Lösung festgelegt. Neben den Anforderungen, die für das Gesamtsystem gelten (Tabelle 3.1), werden auch spezielle Anforderungen an bestimmte Sensoren bzw. an die Software (Tabelle 3.2) und den Stromverbrauch definiert.

Es ist erwünscht, den Betrieb des Systems autonom, ohne externe Stromversorgung durchführen zu können. Die Energie soll durch eine kleine Photovoltaik-Anlage zur Verfügung gestellt werden können. Beispielsweise produziert eine 25 x 25 cm große Anlage typischerweise ca. 9

Watt (Spitzenwert)[1]. Es soll sichergestellt werden, dass die Benutzung (Aufnahme) sowie das Aufladen der Batterie gleichzeitig durchgeführt werden kann. Somit soll die maximale Leistungsaufnahme des Systems selbst (exkl. Batterieladen) bei ca. 5 Watt liegen (siehe Tabelle 3.4).

Anforderungen - Gesamtprojekt	**Verbindlichkeit**	**Umsetzung**
Aufnahme von verschiedenen, definierten Sensordaten	Pflicht	Audio-, Video-, Distanz-, Bewegungs- und Helligkeitsdaten, siehe Tabelle 3.3
Datenübertragung über TCP/IP - Ethernet oder WLAN-Schnittstelle	Pflicht	USB-WLAN-Stick, On-Board-Netzwerkkarte, Netzwerkkarten-Zusatzplatine
Speicherung der Daten auf einem auswechselbaren, externen Datenträger	Pflicht	USB-Festplatte, eingebaute SD-Karte
Mustererkennung (Objekterkennung) mit gängigen Bibliotheken zum maschinellen Sehen	Pflicht	z.B. OpenCV
Versetzen in Sparmodus mittels Software	Pflicht	CPU-Drosselung
System soll kompakte Maße besitzen und preisgünstig sein	Pflicht	-
Möglichkeit einer Mobilfunk-basierenden Datenübertragung	Optional	GPRS/3G Modul
Das zu erstellende Software-Framework soll leicht erweiterbar sein, vor allem im Hinblick auf die Verwendung neuer Sensoren und Erstellung neuer Erkennungsalgorithmen.	Pflicht	Wenn möglich, Verwendung von Objektorientierter Software Architektur mit zur Verfügung gestellten Interfaces

Tabelle 3.1: Anforderungen - Gesamtsystem

[1] Beispielsweise "Voltaic 9 Watt Solar Panel" - http://www.voltaicsystems.com/9-watt-panel

Anforderungen - Software	Verbindlichkeit	Umsetzung
Multithreading-Unterstützung	Pflicht	Linux: POSIX-Threads, C++11 Multithreading
Audio- und Videoaufnahme in Echtzeit (Synchronizität soll gegeben sein)	Pflicht	-
Videoverarbeitung mittels gängiger Bibliotheken zum maschinellen Sehen	Pflicht	OpenCV, FastCV
Erstellung eines Queueing-Mechanismus	Pflicht	-
Einstellbare Aufnahmezeiten (Start, Stopp) sowie einstellbare Intervalle der Sensoren (bei Videoaufnahme zusätzlich die Auflösung)	Pflicht	-
Einfache Konfiguration des Systems	Pflicht	XML-Konfigurationsdatei
Autonome Aufnahme bei der Auslösung durch einen Trigger-Mechanismus (z.B. durch den Bewegungsmelder)	Pflicht	-
Aufnahme mehrerer Sensoren gleichzeitig (z.B Audio, Video, Bewegung, Helligkeit)	Pflicht	Multithreading
Unterschiedliche Arten von Trigger-Verfahren: Zeitliche Trigger, Sensor-basierte Trigger (z.B. Helligkeit > Threshold) und Algorithmus als Trigger (z.B. Aufnahme, wenn ein Objekt erkannt wird)	Pflicht	-
Hardware soll in Ruhe- bzw. Sparmodus versetzt werden können	Pflicht	Linux, CPU-Drosselung
Entfernte Konfiguration	Optional	Webservice
Nachrichtenversand bei Auftreten eines Ereignisses	Optional	Webservice (Push-API)
Leicht erweiterbares Software-Framework	Pflicht	-

Tabelle 3.2: Anforderungen - Software

Anforderungen - Sensoren	Verbindlichkeit	Umsetzung
Einkanal-Audioaufnahme	Pflicht	USB-Mikrofon + Audio-Eingang, USB-Webkamera mit Mikrofon
Videoaufnahme in Echtzeit	Pflicht	USB-Kamera, Kamera-Module - Evaluierung der vorgestellten Systeme notwendig
Distanzmessung	Pflicht	Ultraschall- oder LIDAR-Sensor
Bewegungserkennung	Pflicht	Infrarot-Bewegungsmelder (Digital)
Helligkeitsmessung	Pflicht	Helligkeitssensor 5V bzw. Fotowiderstand (Analog)
Video: Mindestauflösung des Videosensors 1280 x 720 Pixel	Pflicht	USB-Kamera bzw. Kamera-Module
Mindestbildwiederholrate bei Aufnahme 10 FPS	Pflicht	-
Weitwinkel-Optik (Bildwinkel im Bereich zwischen 75° und 120°)	Pflicht	Typischerweise werden Kameras mit einer solchen Optik versehen
Autofokus	Optional	-

Tabelle 3.3: Anforderungen - Sensoren

Die durch Sensoren (siehe Tabelle 3.3) generierten Daten können jeweils aufgenommen (Datenträger) bzw. für das Starten einer Aufnahme verwendet werden. Ob ein Sensor nur für die Aufnahme oder als Trigger verwendet werden kann, wurde bereits in Kapitel 1 und Tabelle 1.1 festgelegt.

Anforderungen - Stromaufnahme	Verbindlichkeit	Umsetzung
Maximale Leistungsaufnahme (inkl. aller Komponenten, sowie Batterieladen) soll 9 Watt nicht überschreiten	Pflicht	-
Leistungsaufnahme von 0,12 kWh pro Tag soll nicht überschritten werden (5W * 24h)	Pflicht	-

Tabelle 3.4: Anforderungen - Stromaufnahme

3.2 Vorevaluierung

Für die Entwicklung des "Proof-of-Concept"-Systems wurde im Vorfeld eine Selektion der Hardware-Plattformen durchgeführt, um eine engere Auswahl für die Vorevaluierung zu erhalten. Hierbei wurden ein Mikrokontroller-basierendes System (Arduino 2560), ein Ein-Kern-SoC-System (BeagleBone bzw. BeagleBone Black) sowie ein Mehr-Kern-SoC-System (Raspberry Pi rev. A und 2 B) ausgewählt. Andere leistungsfähigere Systeme aus dem Mehr-Kern-SoC-Bereich, wie beispielsweise "Nvidia Jetson TK1", wurden aufgrund des hohen Preises bzw. zu hohen Stromverbrauchs nicht weiter berücksichtigt.

Die Auswahlkriterien für die Selektion und das weitere Vorgehen im Zuge der Vorevaluierung sind wie folgt festgelegt (Priorität absteigend):

- Stromaufnahme des Gesamtsystems,
- Erweiterungsfähigkeit des Systems,
- Leistungsfähigkeit des Systems im Hinblick auf Video- und Audioaufnahme sowie auf Ausführung von komplexen Algorithmen aus dem Bereich des maschinellen Sehens,
- Verfügbarkeit der Hardware-Plattformen und Bibliotheken zur Softwareentwicklung.

In der Phase der Vorevaluierung soll die detaillierte Stromaufnahme des Gesamtsystems (inkl. Sensoren) anhand der Datenblätter sowie in weiterer Folge anhand von Strom- und Spannungsmessungen ermittelt werden.

3.2.1 Stromverbrauch

Der Stromverbrauch wurde anhand der Datenblätter bzw. im Falle des Fehlens der Datenblätter durch Literaturrecherche erhoben (siehe Tabelle 3.5). Es wurden nur die Kennzahlen für die Platinen selbst sowie für jeweils kompatible Komponenten angegeben. Bei allen anderen Komponenten (z.B. für Distanz- oder Bewegungssensor) kann davon ausgegangen werden, dass der Stromverbrauch unabhängig vom verwendeten Board ist[2].

[2] Ausgenommen bei Raspberry Pi, da keine analogen Eingänge vorhanden sind (beispielsweise für die Verwendung eines Foto-Widerstands zwecks Helligkeitsmessung). Hier kann zum Beispeil ein AD-Wandler oder "Licht zu Frequenz-Umwandler - TSL235" verwendet werden. Die Stromaufnahme ist jedoch unabhängig von der ausgewählten Lösung im µA-Bereich und somit vernachlässigbar.

Komponente	Arduino 2560 [W]	BeagleBone Black [W]	Raspberry Pi Rev. A [W]	Raspberry Pi 2 B [W]
System	0,13 - 0,70	1,05 - 2,30	1,50	4,00
Netzwerk (WLAN/RJ45)	0,05 - 0,60[a]	-	1,25 - 2,50[b]	-
Videoaufnahme	0,70 - 0,70[c]	1,25 - 2,50[d]	1,25[e]	1,25[e]
Gesamt	0,88 - 2,00	2,30 - 4,80	4,00 - 5,25	5,25

Tabelle 3.5: Stromverbrauch

[a] W5100 Chipset - Arduino-Shield [WIZ 08]
[b] USB-Netzwerkkarte, max. 500mA
[c] CMUcam5 [Lang 16]
[d] USB-Kamera, max. 500mA - inkl. Mikrofon und Videokomprimierung
[e] Pi Camera Module [pic 16]

Alleine wegen des Stromverbrauchs ist das Arduino 2560 zu bevorzugen, da hier von einer maximalen Leistungsaufnahme von 2 Watt auszugehen ist. Dieses hat, wie bereits in Kapitel 2.3.2 angesprochen, eine hohe Anzahl an Erweiterungsplatinen (Shields). Um die Anforderungen zu erfüllen, ist das Netzwerk-Shield für die Datenübertragung sowie die CMUcam5 [Lang 16] und/oder das ArduCAM-Shield [ard 16] für die Videoaufnahme bzw. Objekterkennung erforderlich. Bei BeagleBone Black, das trotz höherer CPU-Taktfrequenz weniger Strom als das BeagleBone (Version 1) benötigt, ist zur Erfüllung der Anforderungen primär eine Erweiterungsplatine (Kamera-Cape) oder eine USB-Kamera notwendig. Ähnlich können Raspberry Pi-Boards sowohl mit einer USB-Kamera als auch mit dem extra für dieses Board entwickelten Kamera-Modul (Pi Camera Module) verwendet werden.

3.2.2 Vergleich der Hardware-Plattformen

Im Folgenden werden die unterschiedlichen Hardware-Plattformen in Hinsicht auf die gegebenen Anforderungen verglichen. Die Anforderungen lassen sich vereinzelt mit nahezu jedem dieser Systeme umsetzen (siehe Tabelle 3.6). Mit dem Mikrokontroller-basierenden Arduino 2560 ist es jedoch nicht möglich, Videodaten aufzunehmen und nachträglich auszuwerten. Für die

Aufnahme ist die Zusatzplatine ArduCAM notwendig, die die Daten direkt auf der SD-Karte speichert. Durch die sehr langsame Verarbeitungsgeschwindigkeit des Mikrokontrollers mangelt es auch an Möglichkeiten zur maschinellen Bildverarbeitung. Eine weitere Limitierung stellt die niedrige Geschwindigkeit (max. 400 kbit/sec.) des I^2C-Busses dar, der für die Kommunikation zwischen dem Arduino und der ArduCAM dient [ard 09]. Was jedoch möglich ist, ist die Verwendung des CMUcam-Zusatzmoduls, das bereits eine Objekterkennung durchführt (siehe Kapitel 2.3.1) und nur die Ergebnisse an das Arduino-Board liefert. Im Rahmen einer Testimplementierung wurde jedoch festgestellt, dass der gesamte Arbeitsspeicher des Arduino 2560 (16 kBytes) nach Verwendung der Bibliotheken für die Speicherung der Daten auf einer SD-Karte sowie eines Web-Servers für die Kommunikation über das TCP/IP-Netzwerk bereits ausgereizt worden ist.

Komponente	**Arduino 2560**	**BeagleBone Black**	**Raspberry Pi A**	**Raspberry Pi 2 B**
Audio	+	+	+	+
Video	-[a]	+	+	+
Bewegung	+	+	+	+
Distanz	+	+	+	+
Helligkeit	+	+	+	+[b]

Tabelle 3.6: Erweiterungsmöglichkeiten

[a] Entweder Aufnahme (ArduCAM - direkt auf der SD-Karte) oder Objekterkennung (CMUcam) möglich.
[b] Durch Verwendung von "Licht-zu-Frequenz-Umwandler - TSL235".

Die Erweiterungsmöglichkeiten betreffend ist BeagleBone Black zu bevorzugen. Dieses hat neben den GPIO-Schnittstellen und dem SPI-BUS (analog zu Raspberry Pi) auch analoge Eingänge sowie eine programmierbare Echtzeit-Einheit (Programmable Real-Time Unit - PRU), die es ermöglicht zeitkritische Anwendungen unabhängig vom Linux-Kernel laufen zu lassen. Die PRU-Einheit ist dazu gedacht, industrielle Kommunikationsprotokolle zu implementieren und kann beispielsweise für die Kommunikation zwischen dem Ultraschallsensor und BeagleBone genutzt werden. Wenn es um die Verarbeitungsgeschwindigkeit geht, sind zwischen BeagleBone Black und Raspberry Pi 2 B nur bei Algorithmen, die auf Mehr-Kern-Prozessoren ausgelegt sind, Un-

terschiede zu erwarten. Hierbei ist noch anzumerken, dass beispielsweise bei OpenCV Mehrkern unterstützende Algorithmen nur vereinzelt vorhanden sind, da hier der Fokus auf Grafikkarten-Programmierung (CUDA) liegt [Pulli 12]. Eine Testimplementierung auf diesen Systemen wurde nicht durchgeführt.

3.3 Auswahl der Hardware und Software

Aus dem Ergebnis der Vorevaluierung ist ersichtlich, dass die Anforderungen mit Arduino 2560 nicht zu erfüllen sind. Im Vergleich zwischen BeagleBone Black und Raspberry Pi-Lösungen wurde BeagleBone Black aufgrund des niedrigeren Stromverbrauchs sowie der weitaus besseren hardwarebedingten Erweiterbarkeit für die Implementierung der "Proof-of-Concept"-Lösung ausgewählt. Um BeagleBone bzw. Raspberry Pi Hardware stromsparender zu betreiben, ist es jedoch bei beiden möglich die CPU-Taktfrequenz runterzusetzen [Brodowski 13].

Bei einem Projekt mit dem Hauptfokus auf Datenauswertung, d.h. Ausführung von komplexen Algorithmen (vorausgesetzt diese sind für Mehr-Kern-Systeme ausgelegt), und weniger auf Stromverbrauch wäre eine Raspberry Pi 2 B Lösung zu bevorzugen.

Für die Regulierung des Eingangsstroms bzw. Ladevorgangs kann das BeagleBone Power-Cape verwendet werden. Weiters wird durch das Power-Cape eine komplette Abschaltung des Systems sowie eine Einschaltung nach verstrichener Zeit ermöglicht. Die Schaltpläne sowie Firmware dafür sind frei verfügbar. Die Auswahl weiterer Komponenten wird in den folgenden Unterkapiteln begründet.

3.3.1 Betriebssystem und Programmiersprache

Generell sind alle Linux-Distributionen, die auf ARM-Prozessoren kompilierbar sind, auf den BeagleBone-Systemen lauffähig. Ausgeliefert werden die Systeme mit Ångström Linux, das speziell für eingebettete Systeme entwickelt wurde.

Hier wurde Debian Linux vor allem wegen einer breiteren Community-Unterstützung anstelle von Ångström Linux bevorzugt. Als Linux-Kernel wurde Linux ARM 4.2.1. mit Real-Time Support ausgewählt. Real-Time Kernel haben eine niedrige Latenzzeit, was beispielsweise für Audioaufnahmen notwendig ist. Um die volle Funktionalität der BeagleBone Black-Hardware auszu-

schöpfen, ist weiterhin die Verwendung von Linux mit Unterstützung der “Device Tree Overlays” notwendig. Diese ermöglichen es, die zur Verfügung stehenden physischen Anschlüsse mit verschiedenen Funktionen zu überlagern. Beispielsweise ist für die Verwendung des AD-Wandlers oder der PRU-Einheit das Einspielen der entsprechenden “Device Tree”-Überlagerung notwendig. Dies ist bei Debian Linux möglich.

Als Programmiersprache wurde C++ gewählt. Die Gründe dafür liegen unter anderem bei dem Geschwindigkeitsvorteil sowie dem geringeren Speicherverbrauch (da nur 512 MB zur Verfügung stehen) gegenüber interpretierten Programmiersprachen, wie beispielsweise Python [Prechelt 00]. Ein weiterer Vorteil von C bzw. C++ ist die Möglichkeit der hardwarenahen Programmierung. Obwohl im Bereich der eingebetteten Systeme die Programmiersprache C gegenüber C++ bevorzugt wird, basiert dies teilweise auf Fehlannahmen, dass C++ langsamer sei und unverhältnismäßig mehr Speicherplatz benötigen würde. Dies wird beispielsweise von Dominic Herity [Herity 98] widerlegt.

3.3.2 Video und Audio

Für die Videoaufnahme wird das Kamera-Cape [CircuitCo 16] bzw. eine USB-Webkamera vom Typ Logitech C920 und für die Audioaufnahme das BeagleBone Audio-Cape sowie ebenfalls die USB-Kamera genutzt. Im Zuge der Evaluierung (Kapitel 4) werden die Ergebnisse der Aufnahme in Bezug auf Stromverbrauch, Qualität und Geschwindigkeit für die verschiedenen Kameras und Aufnahmemethoden ermittelt.

3.3.3 Echtzeituhr

Als Zeitgeber wird das Modul DS1307 (bereits in Kapitel 2.5.6 vorgestellt) verwendet. Das Modul kommuniziert über das I^2C-Protokoll mit dem BeagleBone-System. Nach erfolgreicher Registrierung des Moduls kann die Uhrzeit direkt in der Linux-Benutzerschnittstelle mit dem Befehl “`hwclock -r|w -f /dev/rtc1`” gesetzt bzw. ausgelesen werden [Max 15]. Hier ist zu beachten, dass die Betriebsspannung (Vcc) des Moduls 5 Volt beträgt und der Spannungspegel für ein logisches High maximal 5,3 Volt (Vcc + 0,3) betragen kann. Dies kann bei BeagleBone Black zu einer Überhitzung bzw. Zerstörung von Bauteilen führen, da ein logisches High hier max. 3,3 Volt betragen darf [Coley 13]. Um dies zu vermeiden, wird ein I^2C-Bus-Pegelwandler verwendet.

3.3.4 Bewegung

Hier wurde ein einfacher und preisgünstiger PIR-Sensor, der über die GPIO-Eingänge des BeagleBone ausgelesen werden kann, ausgewählt. Ebenfalls soll auch hier ein Pegelwandler verwendet werden, da das logische High des PIR-Sensors höher als bei BeagleBone Black ist.

3.3.5 Helligkeit

Um Helligkeit zu erfassen, wird der eingebaute Analog-Digitalwandler des BeagleBone Black und ein Fotowiderstand verwendet werden. Es ist zu beachten, dass Fotowiderstände keine kalibrierten Belichtungsmesser sind, sondern nur relative Lichtänderungen messen können. Um die Lichtintensität in sinnvollen Maßeinheiten zu erhalten, ist eine Lookup-Tabelle erforderlich.

3.3.6 Distanz

Für die Distanzmessung wurde vor allem aus Kostengründen der HC-SR-04 Ultraschallsensor (statt einem Lasersensor) ausgewählt [elecfreaks.com 15]. Dieser funktioniert nach dem Prinzip der Laufzeitmessung und benötigt somit ein exaktes Trigger-Signal sowie eine Echtzeit-Messung der Laufzeitverzögerung. Diese Aufgabenstellung kann über die programmierbare Echtzeit-Einheit (Programmable Real-Time Unit - PRU) der BeagleBone-Entwicklungsumgebung zuverlässig gelöst werden.

3.3.7 Auswahl der Programmierbibliotheken

Wie bereits in Unterkapitel 3.3.1 angegeben, wurden als Basis für das System C++ sowie ein Linux-Betriebssystem ausgewählt. Somit bietet sich an, die meistverbreitete Bibliothek für maschinelles Sehen, nämlich OpenCV, zu verwenden. Für die Videoaufnahme kann direkt die OpenCV-Klasse "VideoCapture" verwendet werden. Weiters ist der Zugriff auf Video- und Audio-Datenquellen (Aufnahmegeräte) auf Linux-Systemen über die Video4Linux2-Bibliothek möglich. Für die Audioaufnahme kann sowohl die ALSA-Audio-Bibliothek (Low-Level) als auch eine daran aufbauende (High-Level) Bibliothek, wie beispielsweise PortAudio, verwendet werden. Neben den unten vorgestellten Programmierbibliotheken soll für gleichzeitig stattfindende Aufgaben, beispielsweise gleichzeitige Audio- und Videoaufnahme, die C++11 Thread-Erweiterung

verwendet werden. Um die XML-Konfigurationsdatei auszulesen bzw. auszuwerten, wird die "libxml++"-Bibliothek verwendet, die eine XPath-Unterstützung bietet und somit die Möglichkeit besitzt, nur bekannte Teile des XML-Dokuments zu adressieren und auszuwerten.

OpenCV

OpenCV wird vor allem wegen der breiten Unterstützung, Vielseitigkeit und der breiten Erweiterungsmöglichkeiten ausgewählt. Bei anderen, in Kapitel 2.6.1 angeführten (C bzw. C++ basierenden) Bibliotheken für maschinelles Sehen, werden die fehlenden Dokumentationen bzw. fehlenden Beispiel-Umsetzungen erwähnt (z.B. JavaCV). Für das Ausscheiden der FastCV-Bibliothek (Optimierung für ARM-Prozessoren) ist vor allem die beschränkte Anzahl an Algorithmen verantwortlich. Die Verwendung von openFrameworks (mit OpenCV) wurde aus dem Grund ausgeschlossen, dass nur wenige Bibliotheken aus dem Gesamt-Framework tatsächlich Verwendung finden könnten und die nicht verwendeten den beschränkten Speicherplatz verbrauchen würden (z.B. OpenGL, GLEW, QuickTime, FreeTime). Bei der Verwendung der OpenCV-Bibliothek soll es durch die "Proof-of-Concept"-Lösung möglich sein, aufgenommene Videos bzw. Einzelbilder direkt im OpenCV-Format (`cv::Mat`) zu speichern und in weiterer Folge zu manipulieren. Unter anderem könnte die Optimierung der Aufnahmegeschwindigkeit des OpenCV-Framegrabbers bzw. Video-Grabbers notwendig sein, um die Anforderungen zu erfüllen.

PortAudio

Bei der Verwendung der Audio-Bibliothek wurde PortAudio ausgewählt. Dies ist vor allem der Verständlichkeit und Einfachheit der Programmierschnittstelle zu verdanken, da diese eine schnelle und einfache Umsetzung der Aufnahme und Speicherung der Daten ermöglicht. Weiters ist mit der Auswahl der PortAudio-Bibliothek eine einfache Portierung (beispielsweise auf ein Windows-Betriebssystem) möglich. Wie aus der Arbeit von Topliss et al. [Topliss 14] hervorgeht, ist die Latenz-Zeit bei der Audioaufnahme sowohl mit dem USB-Mikrofon als auch dem BeagleBone Audio-Cape sehr niedrig und ermöglicht somit die Verwendung des BeagleBone als digitales Musikinstrument.

3.4 Hardwarearchitektur

Die Hardwarearchitektur wurde auf Basis von BeagleBone Black erstellt. Die in den Unterkapiteln 3.3.2 bis 3.3.6 ausgewählten Sensoren können über verschiedene BeagleBone-Einheiten angesprochen werden.

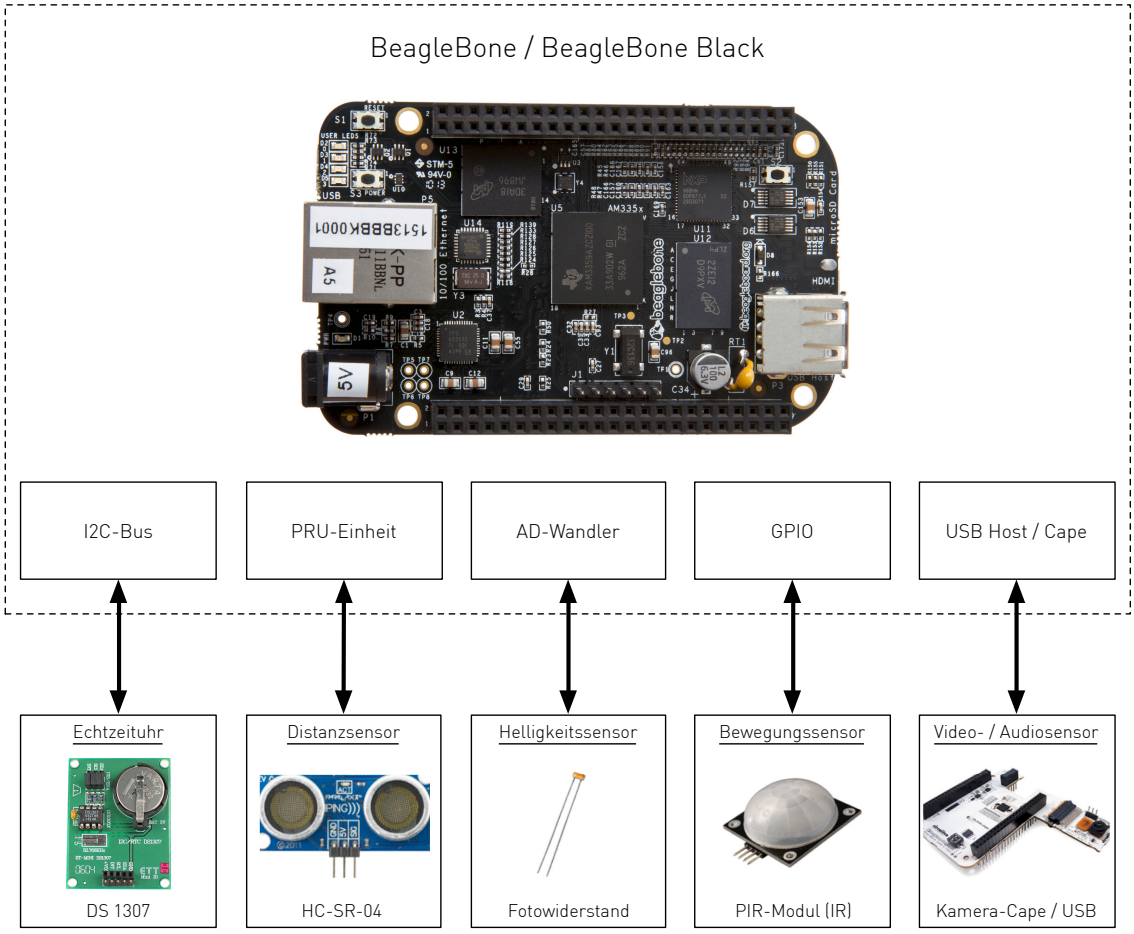

Abbildung 3.1: Allgemeines Blockschaltbild der Hardware-Architektur

Bei der Vorevaluierung und Entwicklung der “Proof-of-Concept”-Lösung wurden bewusst Sensoren gewählt, die von verschiedenen BeagleBone-Einheiten angesprochen werden sollen, um alle Einsatzgebiete bzw. die Vielfältigkeit der BeagleBone-Hardware zu demonstrieren und Musterlösungen zu den verschiedenen Einheiten zur Verfügung zu stellen. Beispielsweise kann bei BeagleBone der “Licht zu Frequenz-Umwandler - TSL235” [Tex 94] verwendet werden und direkt an die GPIO-Einheit angeschlossen werden (analog zum Bewegungsmelder). Stattdessen

wurde ein einfacher Fotowiderstand direkt an einen AD-Wandler angeschlossen.

Es wurden somit folgende BeagleBone-Einheiten für die Verwendung mit den jeweiligen Sensoren ausgewählt (siehe Abbildung 3.1):

- PRU-Einheit in Verwendung mit dem Ultraschallsensor: Hierbei handelt es sich, wie bereits in Kapitel 3.2.2 angesprochen, um eine programmierbare Echtzeit-Einheit, die es ermöglicht, zeitkritische Anwendungen unabhängig vom Linux-Kernel laufen zu lassen. Diese Eigenschaft wird vom Ultraschallsensor benötigt, da sichergestellt werden muss, dass ein nahezu perfektes Rechtecksignal (Trigger) an den Sensor geliefert wird bzw. dieses zurückempfangen wird. Anhand der verstrichenen Zeit wird die Distanz ermittelt. Um die PRU-Einheit zu verwenden, muss das richtige "Device-Tree-Overlay" geladen werden.

- GPIO-Einheit in Verwendung mit dem Bewegungsmelder: Die GPIO-Einheit ist eine vom Benutzer ansprechbare, digitale TTL-Schnittstelle mit einem "High"-Pegelwert von ca. 1,8 Volt. Zuerst wird festgelegt, ob der Port ein Aus- oder Eingang ist, danach kann dieser gesetzt bzw. ausgelesen werden (z.B. durch das Schreiben bzw. Auslesen der Datei: "/gpio07/value").

- AD-Wandler in Verwendung mit dem Fotowiderstand: Der Fotowiderstand ändert je nach Lichteinstrahlung seinen Wert. Mittels dem im BeagleBone integrierten AD-Wandler kann die dadurch erfolgte Spannungsänderung (Ohm'sches Gesetz) ausgelesen werden. Die Verwendung des AD-Wandlers benötigt eine Änderung des "Device-Tree-Overlays" (analog zur PRU-Einheit). Anders als bei GPIO kann der AD-Wandler nur ausgelesen werden (z.B. durch das Auslesen der Datei: "/in_voltage0_raw").

- I^2C-Bus in Verwendung mit der Echtzeituhr: Der 1982 von der Fa. Philips entwickelte (relativ langsame) serielle Datenbus wird hauptsächlich für die Kommunikation zwischen verschiedenen Schaltungsteilen benutzt und soll direkt über das Betriebssystem ausgelesen werden können.

- USB-Bus/Cape-Host in Verwendung mit dem Video- bzw. Audiosensor: Auch hier werden die jeweiligen Daten durch das Auslesen der jeweiligen vom Treiber zur Verfügung gestellten Datei (z.B. "/dev/video0") erhalten. Die Hardware muss im Normalfall nicht explizit beim System vom Benutzer registriert werden (Plug & Play).

3.5 Softwarearchitektur

Es soll ein Objektorientiertes Framework für den Zugriff auf die Sensoren und die Konfiguration des Systems entwickelt werden. Um die Erweiterbarkeit zu ermöglichen, sollen definierte Klassen (beispielsweise die Klasse "Sensor") als Schnittstellen implementiert und zur Verfügung gestellt werden. Diese sollen in C++ als abstrakte Klassen mit virtuellen Funktionen realisiert werden.

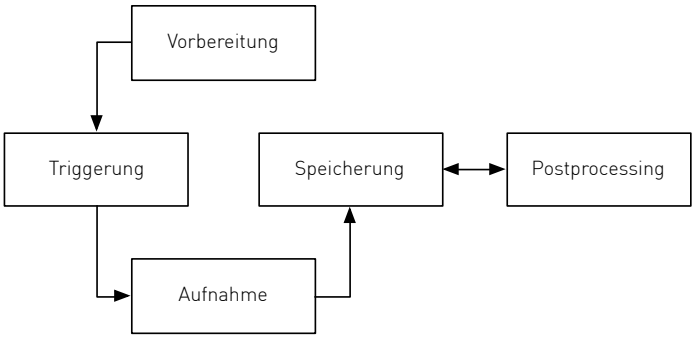

Abbildung 3.2: Allgemeines Funktionsdiagramm der Architektur

Die einzelnen logischen Komponenten sind in Abbildung 3.2 ersichtlich. Im ersten Schritt (Vorbereitung) wird die Konfiguration ausgelesen und der weitere Workflow festgelegt. Damit wird ein Ereignis definiert, das die Aufnahme startet (Triggerung). Außerdem wird festgelegt, welche Daten in welchem zeitlichen Abstand bzw. wie lange aufgenommen und im weiteren Schritt weiterverarbeitet werden sollen. Beim Schritt der Speicherung werden auch bestimmte Metadaten zur erfolgten Aufnahme festgehalten. Im letzten Schritt (Postprocessing) kann eine Auswertung, beispielsweise eine Gesichtserkennung, oder eine Kodierung/Multiplexen der Video-/Audiodaten erfolgen.

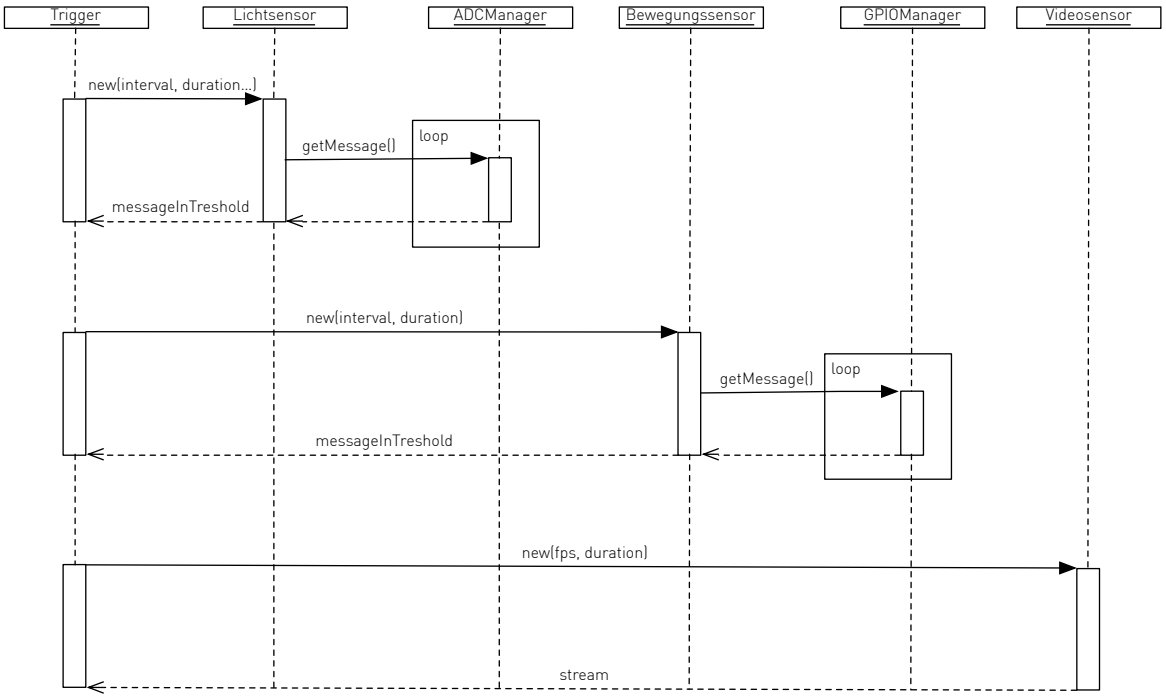

Abbildung 3.3: Sequenzdiagramm des synchronen Datenzugriffs

Für den Austausch der Daten zwischen den einzelnen Komponenten sind zwei Verfahren zu implementieren. Einerseits soll für sofortige Auswertungen der vom Sensor generierten Daten ein direkter Zugriff ermöglicht werden, beispielsweise "getMessage()" eines Bewegungsmelders (synchron). Andererseits soll ein sequentieller Zugriff auf die erstellten Daten möglich sein (asynchron). Je nach Aufgabenstellung können aufgenommene Daten beispielsweise nacheinander einem dafür vorgesehenen Thread ("Postprocessor") übergeben werden, während der direkte Zugriff auf Daten zwischen Sensor und Trigger ermöglicht werden soll. Hier soll ein Queueing-Mechanismus bzw. eine auf Nachrichten basierende Datenübertragung vorgesehen werden. Eine Queue (FIFO) ist nicht auf einen bestimmten Daten- bzw. Objekttyp festgelegt, sondern kann durch den Benutzer angepasst werden. Diese zwei Prozesse (direkter Zugriff auf Daten bzw. Zugriff über eine Queue) werden in den Abbildungen 3.3 und 3.4 dargestellt.

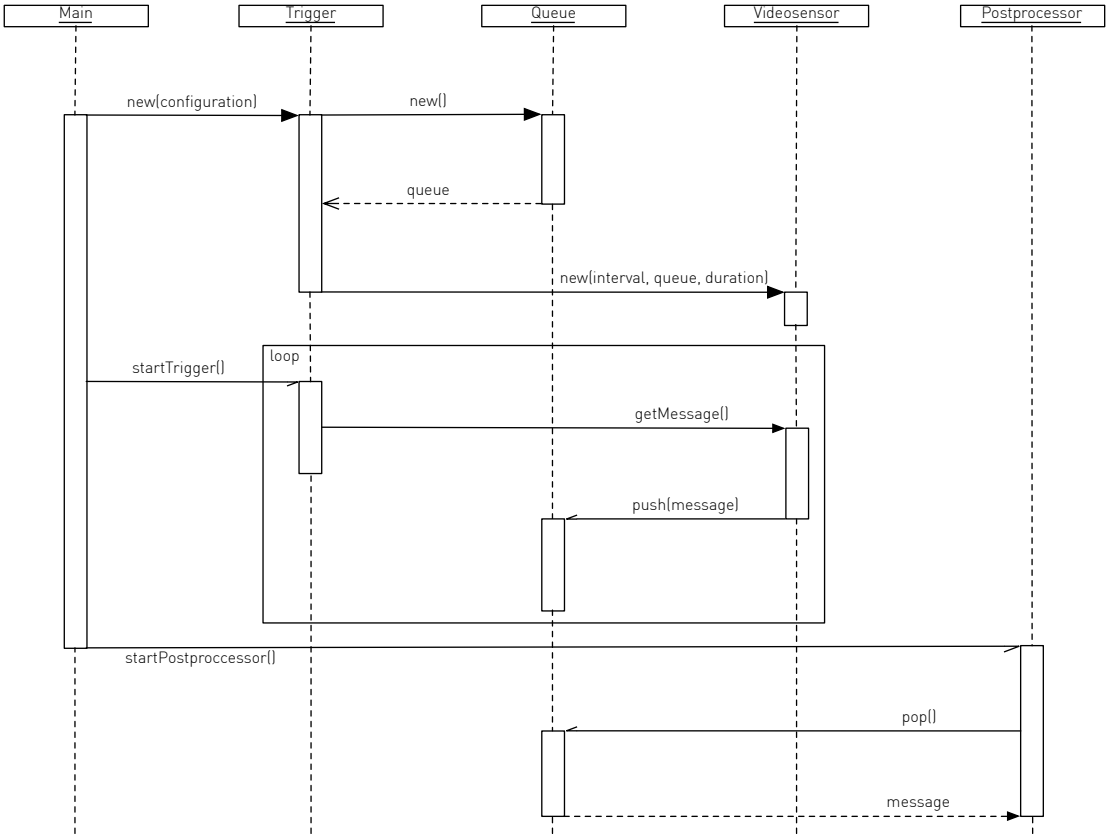

Abbildung 3.4: Sequenzdiagramm des asynchronen Datenzugriffs

3.5.1 Modellierung

Zur Erstellung des Klassenmodells wurde die Anforderungsbeschreibung hinsichtlich Objektklassen, ihrer Attribute und ihrer Beziehungen untersucht. Ein Klassendiagramm ist in Abbildung 3.5 dargestellt[3]. Alle in den Anforderungen definierten Sensoren für die Datenaufnahme sollen basierend auf dem Sensor-Interface aufbauen. Bei den Benutzer-Algorithmen soll ein Beispielalgorithmus (z.B. Gesichtsdetektion) realisiert werden. Ergebnisse der Aufnahme werden in einem abstrakten "Stream"-Datentyp abgelegt. Streams (z.B. Audio oder Video) enthalten neben Messwerten auch Metadaten und können zu einem "Medium" verknüpft werden. Sensordaten können direkt abgerufen ("getMessage()") oder einer Queue ("setQueue()") zugewiesen werden.

[3] Aus Platzgründen sind nicht alle "Setter und Getter"-Methoden angegeben. Es werden stattdessen nur Attribute angegeben (wie auch bei den Interfaces "Sensor" und "userAlgorithm").

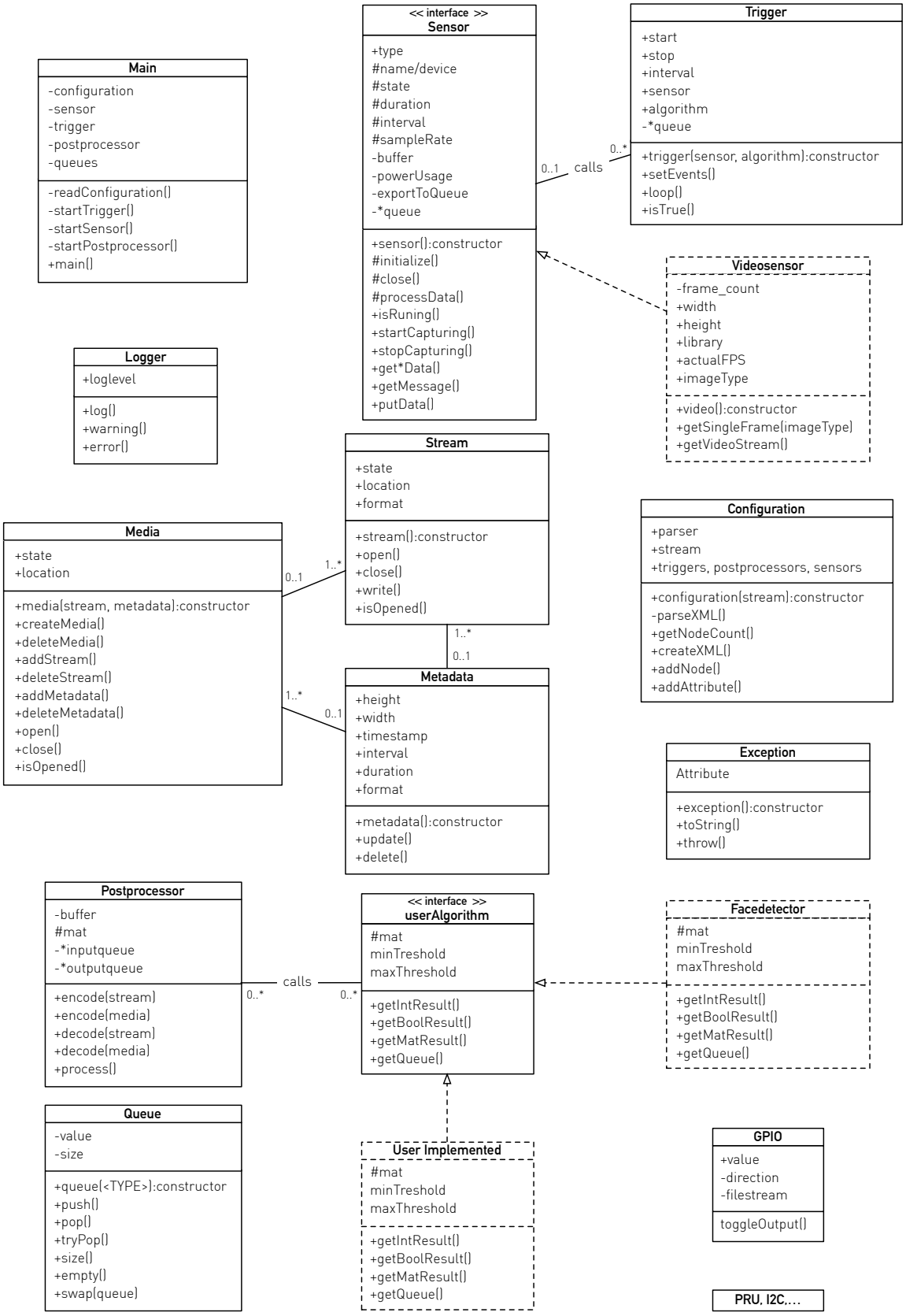

Abbildung 3.5: Klassendiagramm

3.5.2 Herausforderungen

Durch die Limitierungen der Hard- und Software im Embedded-Bereich (siehe Kapitel 2) ist zu erwarten, dass es sowohl bei der Datenakquisition als auch bei der Auswertung zu Performance-Problemen kommen kann. Es wird versucht, diese Probleme durch Optimierung der verwendeten Bibliotheken und Komponenten zu lösen. Bei der Synchronisierung der Daten sollen eventuell auftretende Ausfälle (z.B. einzelne Framedrops bzw. fehlende Messwerte) beispielsweise durch die Anpassung an die tatsächlich aufgenommenen Anzahl der Bilder/Messwerte berücksichtigt werden.

Verarbeitungsgeschwindigkeit

Da bei der Aufnahme von hochauflösenden Video- und Audiodaten sehr große Datenmengen entstehen können, soll eine möglichst effiziente Komprimierung verwendet werden. Es soll zudem möglich sein, eine Dekomprimierung für jedes Einzelbild isoliert (keine P, B-Frames) durchzuführen und dieses direkt in das OpenCV-Format (`cv::Mat`) zu bringen. Die Möglichkeit einer "On-the-Fly"-Komprimierung direkt auf der Kamera (Logitech C920) soll erörtert werden, um den Prozessor zu entlasten.

Nebenläufigkeit

Um die Nebenläufigkeit zu ermöglichen, soll das System Multithreading unterstützen. Dies ist vor allem für die Optimierung der Auslastung und für eine je nach Zielsetzung ausgeglichene Ressourcenverteilung notwendig. Zu diesem Zweck soll die C++11 Multithreading-Bibliothek verwendet werden. Da unter Umständen die Verwendung von ein und derselben Ressource (z.B. Bewegungssensor) nur von einer Programminstanz gleichzeitig erfolgen darf, ist dies bei der Implementierung zu berücksichtigen.

Synchronisation

Um eine Synchronisierung der Daten zu ermöglichen, soll eine genaue Protokollierung durchgeführt werden (z.B. Timestamps der einzelnen Datenpunkte). Bei der gleichzeitigen Aufnahme von Video- und Audiodaten muss die Gesamtaufnahme beim Multiplexverfahren an die jeweils

kürzere Einzelaufnahme angepasst werden. Auch hierfür ist eine Protokollierung der tatsächlich aufgenommenen Bilder nötig. In Kapitel 4 soll eine Evaluierung hinsichtlich der auftretenden Synchronisationsprobleme erfolgen.

3.6 Konfiguration

Die Konfiguration des Systems wird über ein XML durchgeführt. Dieses wird beim Starten des Programms automatisch geladen. Zusätzlich ist ein interaktives Menü für Debugging und Entwicklungszwecke geplant.

Für die Konfiguration wurden im XML folgende Knoten definiert:

- Recording: Festlegung des Aufnahmemodus (z.B. "On-Detection": Bei einfacher Sensor-/Zeit-Triggerung, "Use-Case-1": Benutzerdefinierter Modus - beispielsweise für komplexe Triggerungsmechanismen, z.B. Triggerung durch zwei verschiedene Sensoren (siehe Abbildung 1.1).
- Sensor: Einstellungen des verwendeten Sensors (z.B. Auflösung, Format, Aufnahmedauer).
- Trigger: Sensor, der das Aufnahme-Ereignis anstoßt sowie Start- und Stopp-Zeitpunkt bzw. Intervall definiert (z.B. Bewegungssensor mit Sekundenintervall).
- Postprocessor: Wie sollen die Daten weiterverarbeitet werden (z.B. durch Gesichtserkennungsalgorithmen oder Multiplexing von Video- und Audio-Streams)?

Die Konfiguration des ersten Anwendungsfalls, der in der Einleitung beschrieben ist (siehe Abschnitt 1.3), würde somit wie folgt aussehen:

```
<recording mode="use-case-1" name="light-motion-trigger">
  <sensor type="video" format="mjpeg" width="320" height="240" fps="10" duration=
    "30s" extend="none" store="true"/>
  <sensor type="audio" format="raw" bit="8" samplerate="16000" duration="30s"
    extend="10m"/>
  <trigger type="light" treshold-min="0.5" treshold-max="1.0" start="
    20151118172100" stop="20151217172100" interval="10000"/>
```

```
  <trigger type="motion" start="20151118172100" stop="20151217172100" interval="
    10000"/>
  <trigger-operator operator="AND"/>
</recording>
```

Listing 3.1: Anwendungsfall 1: *Eine nahezu durchgehende Videoaufzeichnung bei Tageslicht und in der Nacht nur eine Audioaufnahme bei durch Sensoren erkannter Bewegung.*

Der zweite Anwendungsfall aus Abschnitt 1.3 kann mit dem generischen “On-Detection”-Modus wie folgt realisiert werden:

```
<recording mode="on-detection" name="motion-trigger">
  <sensor type="video" format="mjpeg" width="640" height="480" fps="10" duration=
    "30s" extend="none" store="true"/>
  <sensor type="audio" format="raw" bit="8" samplerate="16000" duration="30s"
    extend="10m"/>
  <sensor type="distance" samplerate="10" duration="1m"/>
  <sensor type="light" samplerate="1" duration="1s"/>
  <sensor type="motion" samplerate="10" duration="10s"/>
  <trigger type="video" algorithm="face-detection" width="320" height="240"
    treshold-min="2" treshold-max="2" start="20151108172100" stop="20151217172100
    " interval="5000"/>
  <postprocessor type="avmux" priority="normal" video-format="mjpeg" audio-format
    ="mp2" container="mp4"/>
</recording>
```

Listing 3.2: Anwendungsfall 2: *Eine Video-Trigger gesteuerte Aufnahme mit beispielsweise einer Frequenz von einem Bild pro Sekunde und einem Objekt-Erkennungsalgorithmus, der eine Video- und Audioaufzeichnung auslöst, solange Objekte erkannt werden.*

Bei Anwendungsfall 2 ist der Vollständigkeit halber ein “Postprocessor”-Knoten angefügt, der aus den aufgenommenen Video- und Audiodaten eine kombinierte “mp4”-Datei erstellt.

Der dritte Anwendungsfall aus Abschnitt 1.3 kann auch mit dem generischen “On-Detection”-Modus wie folgt realisiert werden:

```
<recording mode="on-detection" name="motion-trigger">
  <sensor type="video" format="mjpeg" width="1280" height="720" fps="1" duration="1" extend="none" store="true"/>
  <trigger type="time" interval="3000000"/>
</recording>
```

Listing 3.3: Anwendungsfall 3: *Durchgängige Videoaufzeichnung im Zeitraffer-Modus (ein Einzelbild je 5 Minuten).*

Die genaue Spezifikation der Konfigurationssprache kann in der XML-Schema (XSD) Notation angegeben werden:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="recording">
    <xs:annotation>
      <xs:documentation>sensor must be used either as trigger or in content</
    xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="sensor" maxOccurs="unbounded" minOccurs="1">
          <xs:annotation>
            <xs:documentation>type: audio, distance, light, motion, video, time
                duration/extend format: s, m, h (seconds, minutes, hours), once,
    none
      </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="type" use="required"/>
                <xs:attribute type="xs:string" name="format" use="optional"/>
                <xs:attribute type="xs:short" name="width" use="optional"/>
                <xs:attribute type="xs:short" name="height" use="optional"/>
                <xs:attribute type="xs:float" name="fps" use="optional"/>
                <xs:attribute type="xs:string" name="duration" use="optional"/>
```

```
                <xs:attribute type="xs:string" name="extend" use="optional"/>
                <xs:attribute type="xs:string" name="store" use="optional"/>
                <xs:attribute type="xs:byte" name="bit" use="optional"/>
                <xs:attribute type="xs:int" name="samplerate" use="optional"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="trigger" maxOccurs="unbounded" minOccurs="1">
            <xs:annotation>
              <xs:documentation>type: audio, distance, light, motion, video, time
                  algorithm: face-detection
                  duration/extend format: s, m, h (seconds, minutes, hours), once
    , none
        </xs:documentation>
            </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="type" use="required"/>
                <xs:attribute type="xs:short" name="width"/>
                <xs:attribute type="xs:short" name="height"/>
                <xs:attribute type="xs:string" name="algorithm"/>
                <xs:attribute type="xs:byte" name="treshold-min"/>
                <xs:attribute type="xs:byte" name="treshold-max"/>
                <xs:attribute type="xs:long" name="start"/>
                <xs:attribute type="xs:long" name="stop"/>
                <xs:attribute type="xs:short" name="interval"/>
                <xs:attribute type="xs:string" name="save-result"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="trigger-operator" maxOccurs="1" minOccurs="0">
            <xs:annotation>
              <xs:documentation>operator format: AND, OR</xs:documentation>
            </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
```

```
              <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="operator" use="required"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="postprocessor" maxOccurs="unbounded" minOccurs="0">
          <xs:annotation>
            <xs:documentation> type: avmux, video
                 algorithm: face-detection
      </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="type" use="required"/>
                <xs:attribute type="xs:string" name="algorithm" />
                <xs:attribute type="xs:byte" name="treshold-min"/>
                <xs:attribute type="xs:string" name="treshold-max"/>
                <xs:attribute type="xs:string" name="save-result"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="mode" use="required"/>
      <xs:attribute type="xs:string" name="name" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 3.4: Beschreibung der XML-Konfiguration in der XSD-Notation

Kapitel 4

Evaluierung und Ergebnisse

In diesem Kapitel werden die Ergebnisse der Systemevaluierung vorgestellt sowie das Erreichen der Ziele diskutiert. Es werden folgende Evaluierungen durchgeführt:

- Evaluierung von einzelnen Sensoren und Aufgaben zuerst im Hinblick auf Verarbeitungsgeschwindigkeit. Hier werden Sensor-spezifische Kennzahlen, wie beispielsweise die Auflösung des Sensors oder maximale Sample-Rate des aufnehmenden Sensors, ermittelt.
- Evaluierung des Stromverbrauchs bei der Aufnahme des jeweiligen Sensors,
- Evaluierung des typischen Stromverbrauchs je nach Anwendungsfall,
- Evaluierung des Software-Frameworks im Hinblick auf einfache Konfiguration und Verwendung. Hierbei werden kurze Quelltext- bzw. Konfigurations-Beispiele angegeben, um dies darzustellen.

Eine Bewertung der "Proof-of-Concept"-Lösung in Hinsicht auf Erfüllung der Anforderungen sowie das Aufzeigen der Grenzen des Systems und deren Erweiterungsmöglichkeiten werden anschließend in Kapitel 5 durchgeführt.

4.1 Videoaufnahme und Auswertung

Die Echtzeit-Videoaufnahme ist eine der wichtigsten Anforderungen an die "Proof-of-Concept"-Lösung. Um diese zu ermöglichen, wurden bereits in der frühen Phase der Implementierung

Tests durchgeführt, um etwaige Engpässe bzw. Mängel zu entdecken. Dabei wurde festgestellt, dass bei der unkomprimierten (RAW) Videoaufnahme (unabhängig von der Aufnahme-Software), sowohl mit dem BeagleBone Kamera-Cape als auch mit der USB-Kamera das Verarbeiten (Einlesen bzw. Schreiben) der Daten zu viel Zeit in Anspruch nimmt. Sequentielle I/O Performance-Messungen auf der SD-Karte (32 GB Lexar HC-MicroSD) haben beim Schreibzugriff eine Datenrate von $\varnothing 11 \pm 2$ MB ($n=3$) pro Sekunde ergeben. Dies bedeutet, dass bei einer Full-HD-Auflösung nur maximal ein Bild pro Sekunde verarbeitet werden konnte. Auch was die Datenmenge betrifft, hat das aufgenommene 10 Sekunden-Video (10 FPS) eine Dateigröße von 1,2 Gigabyte, was nicht praktikabel ist.

Um jedoch auch die Verarbeitungsgeschwindigkeit zu steigern und die CPU, die bei dem RAW-Format sehr beansprucht wird, zu entlasten, wurde die Videokomprimierung auf der USB-Kamera (Logitech C920), wie in Kapitel 3.5.2 vorgeschlagen, umgesetzt. Als Codierungsformat wurde Motion-JPEG (MJPEG) gewählt, da hier keine Inter-Frame Abhängigkeit besteht und eine Decodierung der einzelnen Bilder in OpenCV selbst möglich ist. Um einen weiteren Geschwindigkeitszuwachs zu erzielen, wurde OpenCV in der neuesten Version (3.0) für die SIMD-Prozessoren-Unterstützung neu kompiliert. Weiters wurde die üblich verwendete Bibliothek für MJPEG-Codierung/Decodierung (libjpeg) durch die ebenfalls SIMD-optimierte Bibliothek (libjpegturbo) ersetzt. Hierbei konnte die Frame-Rate bei der Aufnahmen mit einer Auflösung von 640 x 480 Pixel bei über 50% signifikant erhöht werden. Die Einzelbilder wurden mittels der Video4Linux2-Bibliothek ausgelesen, da der von OpenCV zur Verfügung gestellte Video-Grabber (Klasse "`VideoCapture`") nicht auf bereits von der USB-Kamera codierte Daten zugreifen konnte. Somit konnte die Videoaufnahme in einer Full-HD-Auflösung mit ca. 15 Bildern pro Sekunde erreicht werden. Wenn eine Weiterverwendung der Bilddaten im `cv::Mat`-Format und somit eine anfallende Decodierung und Erstellung von der `cv::Mat`-Matrix erwünscht war, konnte eine Verarbeitungsgeschwindigkeit (inkl. Datenspeicherung) von ca. sieben Bildern pro Sekunde erreicht werden (siehe Tabelle 4.1). Bei den Messwerten handelt es sich um den arithmetischen Mittelwert von drei Messungen ($n=3$).

Mit der verwendeten MJPEG-Komprimierung entsteht bei einer Auflösung von 640 x 480 Pixel und 10 FPS eine Datenmenge von 1,7 GB pro Stunde. Die maximal mögliche verwendete SDHC-Kartengröße beträgt 32 GB und würde komplett für die Aufnahme zur Verfügung stehen, da sich das Betriebssystem im internen BeagleBone-Speicher befindet. Somit können mit diesen

Einstellungen maximal etwa 18 Stunden Videodaten aufgenommen werden.

Im weiteren Verlauf wurden die Einzelbilder durch einen Gesichtserkennungsalgorithmus weiterverarbeitet (siehe Kapitel 3.5). Der Gesichtserkennungsalgorithmus, der in OpenCV auf Basis der Arbeit von Viola et al. [Viola 04] implementiert wurde, benötigt für die Erkennung je nach Auflösung zwischen einer Sekunde und drei Sekunden. Somit muss diese Zeit zu den Werten in Tabelle 4.1 dazu gerechnet werden, um die Frame-Rate mit durchgeführter Gesichtserkennung zu erhalten.

Verfahren (n=3)	**Bilder pro Sekunde [FPS] bei 640 x 480**	**Bilder pro Sekunde [FPS] bei 1280 x 720**	**Bilder pro Sekunde [FPS] bei 1920 x 1080**
Unkomprimierte Aufnahme im RAW-(YUYV) Format	5,7	1,6	0,6
Aufnahme mittels H.264 “On-Camera”-Codierung	27,1	28,6	27,8
Aufnahme mittels MJPEG “On-Camera”-Codierung, Decodierung mittels libjpeg-Bibliothek	13,3	12,5	14,6
Aufnahme mittels MJPEG “On-Camera”-Codierung, Decodierung mittels libjpegturbo-Bibliothek	29,1	23,4	15,6
Aufnahme mittels MJPEG “On-Camera”-Codierung, Decodierung mittels libjpeg-Bibliothek und Erstellung eines cv::Mat-Objekts (OpenCV 2.0)	14,6	5,8	2,1
Aufnahme mittels MJPEG “On-Camera”-Codierung, Decodierung mittels libjpegturbo-Bibliothek und Erstellung eines cv::Mat-Objekts (OpenCV 3.0)	15,6	13,2	7,0

Tabelle 4.1: Evaluierung der Videoaufnahme und Verarbeitungsgeschwindigkeit

4.1.1 Rolle der OpenCV-Bibliothek

Die OpenCV-Bibliothek konnte wider Erwartung nicht für die Videoaufnahme verwendet werden, da, wie bereits in Kapitel 4.1 erwähnt, die Hardwarebeschleunigung der USB-Kamera nicht unterstützt wird. Somit wurde die beschleunigte (komprimierte) Aufnahme zusätzlich mit der Video4Linux2-Bibliothek realisiert. Das Decodieren der Videodaten aus dem MJPEG-

Format, die Erstellung eines cv::Mat-Objekts sowie die Auswertung der Videodaten (Gesichtserkennung) erfolgt mittels den von OpenCV zur Verfügung gestellten Methoden. Die Aufnahme ist auch in RAW (YUYV) sowie H.264-Formaten möglich. Die Datenübertragung zwischen der "Video"-Klasse, die für die Aufnahme zuständig ist, und der "Algorithm"-Klasse, die für die Auswertung der Daten zuständig ist, wurde mit einem Queueing-Mechanismus realisiert. Zuerst wird eine Queue mit dem gewünschten Datentyp (z.B. cv::Mat oder ocMessage[1]) erstellt und diese dem "Video"-Objekt zugewiesen. Jedes aufgenommene Frame kann im weiteren Schritt geholt und analysiert werden. Somit ist die Implementierung von Algorithmen, die sowohl auf nur einem Frame als auch auf mehreren aufeinanderfolgenden Frames basieren, möglich.

4.2 Audioaufnahme

Wie in Kapitel 3.3.7 erwähnt, wurden zwei Audio-Bibliotheken zur Implementierung vorgeschlagen (PortAudio bzw. ALSA). Aufgrund einer besseren Multithreading-Unterstützung wurde die Umsetzung mit der PortAudio-Bibliothek realisiert. Beide Aufnahmegeräte, sowohl das BeagleBone Audio-Cape als auch das Mikrofon der USB-Kamera, werden hierbei unterstützt. Derzeit wird bei der Audioaufnahme ein eigenständiger Thread aufgerufen, der die Audiodaten direkt in einen PortAudio-Puffer (Ringpuffer) speichert und diese erst beim Beenden des Aufnahmevorgangs auf das Dateisystem schreibt. Um eine "gleichzeitige" Speicherung durchzuführen, wäre ein weiterer Thread notwendig, der periodisch auf den Puffer zugreift und die Daten in die Datei schreibt[2]. Die Änderung wird als trivial erachtet, wurde aber aus Performance-Gründen nicht realisiert. Im Zuge der Evaluierung wurden Mono, 16 Bit-Audioaufnahmen bei einer Sample-Rate von 16kHz bis vier Stunden durchgeführt. Sowohl die PortAudio-Bibliothek als auch das Mikrofon der USB-Kamera unterstützen Sample-Raten bis 44 kHz mit einer Auflösung von 16-Bit bei einer Stereoaufnahme. Allerdings kann bei dieser Einstellung durch die hohe Datenmenge nur eine isolierte Audioaufnahme problemlos auf dem System erfolgen. Eine Komprimierung der Audiodaten (z.B. im MP3-Format) ist nur nachträglich mit einer entsprechenden Bibliothek möglich.

[1] Eigener Nachrichtendatentyp
[2] Quellcode siehe: http://portaudio.com/docs/v19-doxydocs/paex__record__file_8c_source.html

4.3 Bewegungserkennung

Die Bewegungserkennung wurde, wie erwartet, mit einem PIR-Sensor, angeschlossen an die GPIO-Schnittstelle des BeagleBone, realisiert. Für die Datenübertragung zwischen dem PIR-Sensor und der GPIO-Schnittstelle wurde die Klasse "GPIO-Manager" implementiert, die auf dem Singleton-Entwurfsmuster basiert und Methoden zum Setzen und Auslesen des Zustands durch die "Motion"- (Sensor) Klasse ermöglicht. Die Bewegung wird zuverlässig unabhängig von Lichtverhältnissen erkannt. Der Sensor kann bis zur einer Sample-Rate von über 200 Messungen pro Sekunde verwendet werden. Um eine zugesicherte, höhere Sample-Rate zu erhalten, ist entweder die Verwendung von Real-Time Methoden, womit wiederum Multithreading erschwert wird, oder die Verwendung der PRU-Einheit, da diese unabhängige Ressourcen zur Verfügung stellt, notwendig. Der maximale Einstrahlwinkel beträgt laut Recherche zwischen 80-90°. Die Bewegung wird noch in einer Entfernung von 7-10 Meter erkannt [Par 14].

4.4 Distanzmessung

Für die Distanzmessung wird die PRU-Einheit verwendet. Um diese zu nutzen, ist hierbei, wie in Kapitel 3.3.1 beschrieben, die Verwendung der "Device-Tree Overlays" notwendig. Da diese auf dem bisher verwendeten Linux-Debian ARM Kernel 4.2.1 nicht mehr funktioniert haben, wurde ein Downgrade auf Version "4.1.5-ti-rt-r10 " durchgeführt[3]. Für das Laden und Ausführen des separaten für die PRU-Einheit in C geschriebenen Quellcodes[4] wurde die "PRU-Manager"-Klasse implementiert. Das "Device-Tree Overlay" muss vor der Ausführung des PRU-Programms geladen werden. Auch diese Aufgabe wird, wenn nötig, vom "PRU-Manager" durchgeführt. Dem Benutzer stehen, wie bei anderen Sensoren auch, die Methoden (z.B. "getDouble()") zum Auslesen der Distanz in Zentimetern zu Verfügung. Wie im Datenblatt empfohlen, sollen für eine zuverlässige Messung ca. 60ms eingeplant werden, um das zurückgesendete Signal nicht mit dem neuen Trigger-Signal zu vermischen [elecfreaks.com 15]. Somit kann der Sensor bis zu einer Sample-Rate von 16 Messungen pro Sekunde verwendet werden. Allerdings verringert sich die Sample-Rate durch häufiges Auslesen der PRU-Einheit sowie das Formatieren und Schreiben (C++ Befehl fprintf) der Ergebnisse und des Timestamps auf 8-12 Samples pro Sekunde. Der Einstrahlwinkel

[3] Siehe: http://www.embeddedhobbyist.com/2015/09/beaglebone-black-updating-device-tree-files/
[4] PRU-Quellcode zur Verwendung des HC-SR04-Sensors: https://github.com/luigif/hcsr04

für eine zuverlässige Messung beträgt in etwa 15 bis 40°[elecfreaks.com 15][Cyt 13].

4.5 Helligkeitsmessung

Um bei BeagleBone eine analoge Eingabe zu ermöglichen, müssen hier ebenfalls “Device-Tree Overlays” verwendet werden. Der auf dem Board vorhandene AD-Wandler (“ti_am335x_adc”) muss in den Linux-Kernel geladen werden, bevor die analogen Signale ausgelesen werden können. Diese Aufgabe wird, wie bei den anderen Sensoren, von einer Singleton “ADCManager”-Klasse übernommen. Dem Benutzer werden für die Abfrage des Werts die Methoden der Klasse “Light” (z.B. “getInt()”) zur Verfügung gestellt. Geliefert werden relative Änderungen in dem vom AD-Wandler definierten Bereich von 0 bis 4095 (“getInt()”) sowie auf einem Einheitsintervall abgebildeten Intervall (“getDouble()”), das die Werte in einem Bereich von 0 bis 1 zurückliefert. Als Schwellwert bei den Messungen zum Stromverbrauch wurden Werte von unter 0,5 als “Nacht” und darüber als “Tag” festgelegt. Das Sensor kann zuverlässig bis zur einer Sample-Rate von über 200 Messungen pro Sekunde verwendet werden, allerdings ist bei der relativ geringen Änderung der Helligkeit über den Tag verteilt auch eine Messung pro Minute ausreichend.

4.6 Nutzung der Sensoren im Software-Framework

Konfiguration des Systems

Um die Einfachheit des Software-Frameworks im Hinblick auf einfache Konfiguration und Nutzung darzustellen, wird hier ein vollständiges Konfigurations-Beispiel sowie ein kurzer Quelltext, der die Verwendung des Software-Frameworks erläutert, gezeigt. Die Konfiguration des Systems ist im Listing 4.1 ersichtlich.

```
<?xml version="1.0"?>
<recording mode="on-detection" name="motion-trigger">
  <sensor type="video" format="mjpeg" width="640" height="480" fps="25" duration=
    "5h" extend="none" store="true"/>
  <sensor type="audio" format="raw" bit="8" samplerate="16000" duration="1m"
    extend="10m"/>
```

```
  <sensor type="distance" samplerate="1" duration="1m"/>
  <sensor type="light" samplerate="1" duration="1m"/>
  <sensor type="motion" samplerate="1" duration="1m"/>
  <trigger type="time" start="20160221150500" stop="20160221200000" interval="
    60000"/>
  <trigger type="video" algorithm="face-detection"  width="640" height="480"
    treshold-min="1" treshold-max="1" start="20151102125959" stop="20151209172100
    " interval="10000"/>
  <trigger-operator operator="AND"/>
  <postprocessor type="video" algorithm="user-algorithm" treshold-min="5"
    treshold-max="10" save-result="xml"/>
  <postprocessor type="avmux" video-format="mjpeg" audio-format="mp2" container="
    mp4"/>
</recording>
```

Listing 4.1: Vollständige Konfiguration des Systems.

Beispiel zur Nutzung im Software-Framework

Die Nutzung der wesentlichen Software-Komponenten im Sinne der Erweiterbarkeit, beispielsweise Sensoren oder Benutzeralgorithmen, erfolgt auf Basis der Interfaces. Im Listing 4.2 ist die Nutzung des Software-Frameworks dargestellt.

```
// Erstellung des Video-Sensor-Objekts mit den Parametern aus dem XML

Video video(VIDEO_DEFAULT_DEVICE, width, height, VIDEO_USE_V4L2); //
    Aufnahmegereat - Standard: Logitech USB-Kamera, Aufloesung und die verwendete
     Bibliothek werden definiert. Neben Video4Linux2 ist auch eine Aufnahme mit
    OpenCV möglich.
Trigger *videotrigger = new Trigger();
FaceDetector *faceDetector = new FaceDetector();
Queue <ocMessage> queue;

videotrigger -> setQueue(&queue);
thread t_trigger = videotrigger -> startTrigger(&video, faceDetector, interval,
    start, stop, tresholdMin, tresholdMax);
t_trigger.detach(); // Trigger-Thread laeuft weiter. Frames werden in die Queue
    gespeichert und vom FaceDetector ausgelesen.
```

```
11
   // Erstellung des Audio-Sensor-Objekts mit den Parametern aus dem XML
13
   Audio audio(AUDIO_PORTAUDIO_DEFAULT_DEVICE, AUDIO_USE_PORTAUDIO,
       AUDIO_MODE_INPUT_ONLY, audio_callback); // PortAudio für die Eingabe nutzen.
       Funktion audio_callback wird aufgerufen sobald eine Aufnahme begonnen hat (
       derzeit nicht in Verwendung)
15 Stream stream(&audio, false, true, format); // Sensor, ist lesbar, ist
       beschreibbar, Format z.B. RAW (PCM)

17 // Speicherung eines Audio-Streams mit den Parametern aus dem XML
   stream.open(STREAM_DIRECTION_WRITE);
19 audio.writeAudioStream(stream, samplerate, duration, AUDIO_INPUT_MONO,
       AUDIO_DEFAULT_INPUT_FRAMES); // Samplerate aus dem XML, vordefinierte Mono
       aufnahme, Frames pro Buffer die vom Audio-Callback geliefert werden (
       PortAudio)
   stream.close();
21
   // Erstellung des Motion-Sensor-Objekts - jeder Sensor kann als Trigger (siehe
       oben) aufgerufen werden, als auch deren Wert direkt im Programmcode
       ausgelesen werden. z.B.:
23
   Motion motion;
25 if (motion.getBool()) {
     // Bewegung erkannt.
27 }

29 Distance distance;
   if (distance.getDouble()) {
31   // Liefert die Entfernung des Objekts in cm zurueck.
   }
33
   Light light;
35 if (light.getDouble()) {
     // Liefert die Helligkeit zueruck.
37 }
```

Listing 4.2: Verwendung der Sensoren zur Triggerung oder Aufnahme

Im Listing 4.2 sind die Erstellungen der Sensor-Instanzen und der darauf basierende Trigger-Instanzen dargestellt. Es müssen Instanzen der "Video", "Trigger" und "Algorithm" (FaceDetector)-Objekte erstellt werden. Dem Trigger wird eine Queue zugewiesen und es wird ein eigenständiger Trigger-Thread gestartet. Der "Video"-Sensor wird somit unter Berücksichtigung der Start- und Stopp-Zeit sowie dem im XML festgelegten Intervall als Trigger definiert. Die Software-Architektur samt dem Queuing-Mechanismus wurde bereits in Kapitel 3.5 besprochen und konnte wie gewünscht umgesetzt werden.

Ab der Zeile 12 wird die Instanzierung und die Speicherung ("writeAudioStream") eines vom "Audio"-Sensor erzeugten "Streams" vorgezeigt. Das Schreiben in einen "Stream" kann auch für die ab der Zeile 24 gezeigten Sensoren (für Bewegungs- oder Helligkeits-Messung) durchgeführt werden. Hier sind jedoch Beispiele gezeigt, wie man Augenblickswerte abfragen kann.

Ausgabe des Systems

Bei den Sensordaten, die nur eine lesbare ASCII-Textdatei erzeugen, wird neben dem Start- und Stopp-Zeitpunkt sowie der gewünschten Sample-Rate zusätzlich zu jedem Datenpunkt ein Index sowie ein Timestamp gespeichert. Im Logging-Mechanismus werden zudem Warnungen ausgegeben, sobald ein Datenverlust eingetreten ist. Ein Auszug der Ausgabe einer Distanzmessung kann dem Listing 4.3 entnommen werden.

```
start: Wed Feb 24 21:54:29 2016 / samplerate: 10 (samples per second)
[0 / 02 24 21:54:29 2016 ]->213.40 cm
[1 / 02 24 21:54:29 2016 ]->166.44 cm
[2 / 02 24 21:54:30 2016 ]->121.13 cm
[3 / 02 24 21:54:30 2016 ]->165.18 cm
[4 / 02 24 21:54:30 2016 ]->194.93 cm
[5 / 02 24 21:54:30 2016 ]->129.16 cm
[6 / 02 24 21:54:30 2016 ]->129.59 cm
[7 / 02 24 21:54:30 2016 ]->130.10 cm
[8 / 02 24 21:54:30 2016 ]->233.88 cm
[9 / 02 24 21:54:30 2016 ]->93.62 cm
end: Wed Feb 24 21:54:30 2016 / samples: 10
```

Listing 4.3: Beispielhafte Ausgabe einer Distanzmessung

4.7 Echtzeituhr

Wie in Kapitel 3.3.3 vorgeschlagen, wurde die Echtzeituhr auf Basis des DS1307-Moduls realisiert. Die Uhrzeit wird zusätzlich beim Starten des Linux-Programms über das “Network Time Protocol” mit einer entfernten Uhr synchronisiert.

4.8 Synchronisation

Bei der Aufnahme der Sensordaten, bei denen die gesicherte, gleichmässige periodische Aufnahme nicht möglich ist (z.B. durch Verwendung von Multithreading) sowie auch keine Zeitstempel für jeden einzelnen Messwert erfasst werden können, muss davon ausgegangen werden, dass zwischen den verschiedenen Sensoren eine Asynchronität entstehen kann. Dies betrifft sowohl Audio- als auch Videodaten.

Eine Asynchronität der Video- und Audiodaten ist vor allem nach der Erstellung einer gemeinsamen Datei, die sowohl eine Video- als auch Audiospur enthält, sofort ersichtlich. Während die Audioaufnahme bei der “Proof-of-Concept”-Lösung unabhängig von der Qualität (z.B. Datentyp, Bits, Anzahl der Kanäle) konstant bleibt, kommt es vor, dass bei der Videoaufnahme die erwünschte Anzahl an Bildern pro Sekunde wegen fehlender Leistungsfähigkeit der Hardware (CPU-Geschwindigkeit, I/O-Geschwindigkeit) nicht erreicht werden kann. Hier muss eine nachträgliche Korrektur vorgenommen werden. Zum einen kann die Synchronisation der Video- und Audiodaten anhand der tatsächlichen Anzahl der Bilder pro Sekunde erfolgen und zum anderen kann die Länge der Gesamtaufnahme an die kürzere Spur von beiden (Audio oder Video) angepasst werden.

Bei der “Proof-of-Concept”-Lösung wurde eine solche Asynchronität anhand der audiovisuellen Evaluierung einer Audio- und Video-Aufnahme eines mit Untertiteln versehenen abgespielten Videos festgestellt. Die Evaluierung hat ergeben, dass bei längeren Aufnahmen (ab etwa 25 Minuten) eine Asynchronität entsteht und diese bei Aufnahmen von mehreren Stunden sogar einige Sekunden betragen kann.

Bei allen anderen Sensoren kann die Asynchronität durch die Verwendung von Zeitstempeln bei jedem Messwert bei der Auswertung berücksichtigt werden.

Das Erstellen einer gemeinsamen Video- und Audio-Datei (Multiplexen) wird von der “Proof-of-Concept”-Software über einen externer Prozess “avconv” in einem separaten Thread (über die “posix_spawn”-Funktionalität) aufgerufen. Dabei werden die zwei Spuren sowie die gewünschten Parameter zur Transcodierung übergeben.

4.9 Stromverbrauch

Die Evaluierung des Stromverbrauchs wurde in zwei Bereiche unterteilt. Zum einen wurde der Stromverbrauch je nach Trigger bzw. Aufnahmesensor ermittelt und zum anderen je nach Anwendungsfall (siehe Szenarien aus Kapitel 1.3). Hier ist besonders der Mittelwert des Stromverbrauchs über eine längere Zeitspanne interessant.

4.9.1 Messverfahren

Um den Mittelwert des Stromverbrauchs in verschiedenen Verarbeitungsstufen des Systems, beispielsweise bei der Triggerung, der Aufnahme oder Wartezeit, zu erhalten, ist hier eine Langzeitmessung am zielführendsten. Hier kann beispielsweise mit einem Volt- bzw. Amperemeter in sehr kurzen Intervallen ein Messwert abgelesen werden oder ein Leistungsmessgerät verwendet werden. Da der Stromverbrauch über mehrere Stunden ermittelt werden soll, wurde hierbei als Messinstrument das handelsübliche 230 Volt Leistungsmessgerät “COST CONTROL 3000” der Firma Basetech verwendet [Basetech 09]. Die Messergebnisse wurden durch Stichproben-Messungen mit einem Volt- bzw. Amperemeter evaluiert. Das für die Stromversorgung verwendete Netzteil besitzt laut Datenblatt einen Wirkungsgrad von 70% und muss beim Messergebnis berücksichtigt werden [viv 16].

Alle Testläufe wurden n-mal wiederholt. Wenn es sich um keine Einzelmessungen handelt, sind bei den Messwerten jeweils die Mittelwerte sowie Standardabweichungen über die n-Messungen angegeben. Um den Verbrauch in elektrischer Leistung (P) zu erhalten, muss die vom Messgerät in Kilowattstunden angegebene Energie (E) wie folgt umgerechnet werden:

$$P = 1000 * E / t \tag{4.1}$$

4.9.2 Ergebnisse

In Tabelle 4.2 sind die Ergebnisse der Messung aufgelistet. Zuerst werden die Messungen pro Anwendungsfall (drei Messungen über 24 Stunden) und danach pro Sensor angegeben. Um den Stromverbrauch je nach Belastung des Systems zu ermitteln wurden alle Anwendungsfälle in zwei verschiedenen Intensitäten ausgeführt.

Bei “moderaten” Anwendungen wurde die Triggerung alle fünf Sekunden ausgelöst und die Dauer der Video- und Audioaufnahme beträgt 30 Sekunden. Bei “intensiven” Anwendungen wurde die Triggerung alle drei Sekunden ausgeführt und die Aufnahme dauert mindestens eine Stunde. Die CPU-Geschwindigkeit wird bei einer Bildwiederholungsrate unter acht Bildern pro Sekunde und einer Auflösung bis 640 x 480 per Software auf 300 MHz beschränkt, um den Stromverbrauch niedrig zu halten. Bei höheren Auflösungen bzw. Bildwiederholungsraten wird diese auf das Maximum von 1000 MHz gesetzt. Bei der Benutzung aller anderen Sensoren wurde die Prozessorgeschwindigkeit auf 300 MHz gesetzt.

Wie aus den Ergebnissen der Evaluierung ersichtlich ist, benötigt die Aufnahme mit einer sehr langen Video- und Audiodauer (“intensiv”) sowie häufiger Triggerung deutlich mehr Ressourcen. Dies ist vor allem mit der hohen Prozessorauslastung zu begründen und wird bei Anwendungsfall 2, bei dem eine häufige Gesichtserkennung durchgeführt wird und bei dem der Stromverbrauch am Höchsten ist, deutlich. Jedoch liegt der Stromverbrauch unterhalb von den in den Anforderungen gesetzten Werten. Sowohl die geforderten 0,12 Kilowattstunden pro Tag als auch die maximale Leistungsaufnahme von 8 Watt werden deutlich unterschritten. Besonders hervorzuheben ist, dass bei der Aufnahme von mehreren Sensoren der Stromverbrauch nur sublinear steigt. Auch dies ist besonders aus den Messungen des Stromverbrauchs aus den Anwendungsszenarien in Kapitel 1.3 ersichtlich. Somit kann vor allem eine reine Überwachungstätigkeit ohne Datenanalyse am Stromsparendsten durchgeführt werden können.

Anwendungsfall / Sensor	**Messung 1**		**Messung 2**		**Messung 3**		**Mittelwert**	
	Energie [kWh][a]	Leistung [W][b]	Energie [kWh][a]	Leistung [W][b]	Energie [kWh][a]	Leistung [W][b]	Energie [kWh][a]	Leistung [W][b]
Anwendungsfall 1 “moderat”, n=3; Auslösungen: $\varnothing 12 \pm 2$; [c]	0,014	0,58	0,014	0,58	0,021	0,875	0,016	0,68
Anwendungsfall 1 “intensiv”, n=3; Auslösungen: $\varnothing 14,3 \pm 1,52$; [d]	0,049	2,04	0,035	1,45	0,042	1,75	0,042	1,75
Anwendungsfall 2 “moderat”, n=3; Auslösungen: $\varnothing 28 \pm 9,89$;[e]	0,014	0,58	0,021	0,875	0,028	1,16	0,021	0,87
Anwendungsfall 2 “intensiv”, n=3; Auslösungen: $\varnothing 12,3 \pm 2,51$; [f]	0,049	2,04	0,035	1,48	0,042	1,75	0,042	1,75
Anwendungsfall 3 “moderat”, n=3; Auslösungen: $\varnothing 2880 \pm 0$; [g]	0,014	0,58	0,021	0,875	0,021	0,875	0,018	0,77
Anwendungsfall 3 “intensiv”, n=3; Auslösungen: $\varnothing 8640 \pm 0$; [h]	0,028	1,16	0,035	1,45	0,035	1,45	0,032	1,36
Video, 640 x 480 25 FPS - MJPEG[i]	0,042	1,75	0,035	1,45	0,042	1,75	0,039	1,65
Audio, Mono 16 Bit[i]	0,021	0,87	0,021	0,87	0,028	1,16	0,023	0,97
Distanz, 1 Sample pro Sekunde[i]	0,018	0,75	0,022	0,93	0,019	0,81	0,02	0,83
Bewegung, 1 Sample pro Sekunde[i]	0,017	0,72	0,021	0,87	0,021	0,87	0,019	0,82
Helligkeit, 1 Sample pro Sekunde[i]	0,021	0,82	0,017	0,72	0,021	0,87	0,019	0,82

Tabelle 4.2: Evaluierung des Stromverbrauchs

[a] Energieverbrauch pro Tag (24h)
[b] Nach Formel 4.1
[c] Trigger: Distanz & Helligkeit: 60 Sekunden; Video: 1h / 640 x 480 3 FPS - MJPEG
[d] Trigger: Distanz & Helligkeit: 3 Sekunden; Video: 1h / 640x480 10 FPS - MJPEG
[e] Trigger: Gesichtserkennung alle 5 Sekunden; Video: 30 Sek. / 640x480 10 FPS - MJPEG; Audio: 30 Sek. / Mono 16 Bit; Distanz & Helligkeit: 1 Min. / 1 Sample pro Sekunde; Bewegung: 10 Sek. / 1 Sample pro Sekunde
[f] Trigger: Gesichtserkennung alle 3 Sekunden; Video: 1h / 640x480 10 FPS - MJPEG; Audio: 1h / Mono 16 Bit; Distanz & Helligkeit & Bewegung: 1h / 1 Sample pro Sekunde
[g] Trigger: alle 30 Sekunden; Video: 1s / 1280x720 1 FPS - MJPEG
[h] Trigger: alle 10 Sekunden; Video: 1s / 1280x720 1 FPS - MJPEG
[i] n=3; Trigger: einmalig; Aufnahme: 24h des jeweils angegebenen Sensors (ausgenommen Audio - hier wurde die Aufnahme jede 4h gestartet)

Kapitel 5

Zusammenfassung

Das Ziel dieser Diplomarbeit war, das Potential und die Machbarkeit eines möglichst autonomen und vom Benutzer konfigurierbaren Überwachungssystems mit handelsüblichen Komponenten zu evaluieren sowie eine preisgünstige “Proof-of-Concept”-Lösung zu entwickeln. Das System soll im Stande sein, getriggert durch verschiedene Sensoren, Aufnahmen zu starten und ist ursprünglich für den Zweck der Tierbeobachtung im Feld gedacht. Daher lag der Fokus in der Entwicklung eines Systems, das möglichst stromsparend funktionieren soll.

Im Rahmen der Diplomarbeit wurden zuerst verschiedene Software- und Hardware-Plattformen evaluiert mit denen die Anforderungen umgesetzt werden konnten. Nachfolgend wurde eine “Proof-of-Concept”-Lösung entwickelt und ausführlich getestet. Nach der ersten Vorevaluierung der Hardware-Plattformen (es sind über 120 verschiedene eingebettete Systeme am Markt) wurde letztlich aus drei verschiedenen Systemen das BeagleBone Black ausgewählt. Nach der Auswahl der Komponenten (Sensoren) wurde die Hardware- und Software-Architektur entworfen und mit der Implementierung begonnen. Die Implementierung des Software-Frameworks, das aus ungefähr 9.500 “Lines of Code” (LOC) besteht, wurde in C++ unter Verwendung der Bibliotheken Video4Linux2, PortAudio, libxml++ sowie der OpenCV-Bibliothek für maschinelles Sehen durchgeführt. Die Konfiguration des Systems kann mittels einer XML-Konfigurationsdatei durchgeführt werden. Anschließend wurde eine Evaluierung vor allem im Hinblick auf die Verarbeitungsgeschwindigkeit und Stromaufnahme durchgeführt sowie deren Ergebnisse diskutiert. Im Folgenden werden die Erkenntnisse aus der Arbeit zusammengefasst, limitierende Faktoren identifiziert und zukünftige Erweiterungen erörtert.

5.1 Ergebnisse

Sowohl die Audio- als auch Videoaufnahme konnte im gewünschten Umfang realisiert werden. Eine durchgängige Aufnahme mit einer Auflösung von 640 x 480 Pixel und 10 FPS sowie zusätzlichen Audiodaten kann bei einer Benutzung von SD-Speicherkarten (Maximum von 32 GB laut SDHC-Standard) etwa 16 Stunden betragen. Eine Erweiterung des Speicherplatzes kann beispielsweise mit einem externen Datenträger durchgeführt werden (z.B. USB-Festplatte, Netzwerkspeicher). Die Synchronizität zwischen der Video- und Audiospur ist bei Aufnahmen nur bis zu einer halben Stunde gegeben. Eine Unterbrechung und nochmalige Durchführung der Aufnahme (in eine neue Datei) würden das Problem beheben. Parallel zur Video- und Audioaufnahme kann auch eine gleichzeitige Auswertung der Daten durchgeführt werden, jedoch ist ab einer gewissen Datenmenge (bzw. Verarbeitungszeit) eine Serialisierung dieser Aufgabe empfehlenswert. Auch bei der Verwendung von anderen, im Rahmen dieser Arbeit evaluierten Sensoren konnten alle Anforderungen erfüllt werden. Hier wurde das System je nach Anwendungsfall und für jeden Sensor separat evaluiert. Neben der Ermittlung der Verarbeitungsgeschwindigkeit pro Sensor wird auch eine Gesamtevaluierung durchgeführt wurde. Diese Vorgangsweise wurde auch für den Stromverbrauch durchgeführt. Durch den sublinearen-Stromverbrauch (siehe 4.9.2) ist bei dem entwickelten System im Schnitt ein Verbrauch zwischen $\varnothing 0,68 \pm 0,16$ Watt und $\varnothing 1,75 \pm 0,29$ Watt ($n = 3$), je nach Anwendungsfall und den verwendeten Sensoren, zu erwarten. Bei Volllast ist eine momentane Leistungsaufnahme von 3,5 Watt gegeben, die allerdings in den Anwendungsfällen nie erreicht wurde. Somit kann das umgesetzte System als Konkurrenz zu anderen weitaus teureren Systemen bzw. als Ersatz zu der eingestellten LeanXcam dienen.

Zusammenfassend lässt sich sagen, dass mit der “Proof-of-Concept”-Lösung alle Anforderungen erfolgreich umgesetzt werden konnten. Es konnte eine preisgünstige Hardware zusammengebaut werden sowie ein frei verfügbares Open-Source-Framework geschaffen werden, das alle Anforderungen aus Kapitel 3.1 erfüllt. Der Stromverbrauch des Systems und der Sensoren ist selbst bei der Videoaufnahme dank der Videokomprimierung, die in der USB-Kamera durchgeführt wird, sogar niedriger als in der Vorevaluierung angenommen (siehe Kapitel 3.2.1). Einzig bei längeren, gleichzeitigen Video- und Audioaufnahmen kommt es nach einer gewissen Zeit zu einer Asynchronität zwischen den beiden Spuren.

5.2 Grenzen des Systems und Erweiterungsmöglichkeiten

Durch die Auswahl der Plattform und der Sensoren sind bewusst Entscheidungen getroffen worden, die Limitierungen mit sich bringen. Beispielsweise ist für die Distanzmessung aufgrund niedriger Kosten ein Ultraschallsensor gewählt worden, obwohl die maximal zuverlässig messbare Distanz (0 bis ca. 6 Meter) verglichen mit einem Lasersensor (bis 50 Meter) relativ gering ist. Auch die Verarbeitungsgeschwindigkeit ist bei einer Full-HD-Aufnahme mit sieben Bildern pro Sekunde relativ gering. Hier sind definitiv Verbesserungsmöglichkeiten vorhanden. Eine weitere Limitierung ist die bei der Video- und Audioaufnahme entstandene Asynchronität und softwareseitig eine fehlende Priorisierung der laufenden Threads, da die Lösung dieser Aufgabe mit der bisher verwendeten C++11-Threading Bibliothek nicht möglich ist.

Möglicherweise können die Probleme der Verarbeitungsgeschwindigkeit und Asynchronität alleine mit neueren, leistungsfähigeren Modellen der BeagleBone-Hardware behoben werden. Auch die Verwendung von separaten Threads zur "zeitnahen" Speicherung von Audiodaten, wie bereits in Kapitel 4.2 beschrieben, könnte ein Ansatz sein, um die Asynchronität zu beheben. Als softwareseitige Erweiterung kann die Speicherung der Metadaten sowie die Katalogisierung der Ergebnisse beispielsweise über eine relationale Datenbank (z.B. SQLite[1]) erfolgen, um die Auswertungen der Daten leichter zu ermöglichen. Beispielsweise könnte man damit nur Daten abrufen, in denen eine bestimmte Tierart erkannt worden ist.

Neben den Erweiterungsmöglichkeiten des Software-Frameworks, sind die Erweiterungsmöglichkeiten auch hardwareseitig vielfältig. Neben den bereits im vorherigen Abschnitt angesprochenen Sensoren zur Distanzmessung, kann auch die Einbindung eines Monitor-Capes und die Unterstützung einer grafischen Benutzerschnittstelle (inkl. Touch-Bedienung) je nach Anwendungsgebiet vorteilhaft sein. Somit wäre bei der Benutzung einer grafischen Benutzerschnittstelle auch im Feld eine Konfiguration des Systems möglich. Weiters kann das System um die bereits in Kapitel 3.3 vorgeschlagenen Power-Capes zur Unterstützung des Ladevorgangs erweitert werden. Aber auch die Entwicklung eines wasserdichten Gehäuses für eine Verwendung im Aussenbereich wäre möglich. Hier kann auch der Einsatz von IR-Dioden zur Beleuchtung bei Nachtaufnahmen von Vorteil sein. Dabei bietet es sich an neben den frei verfügbaren BeagleBone-Gehäusen ein mit 3D-Druck-Verfahren angepasstes Gehäuse zu erstellen, dass auf

[1] Die SQLite-Bibliothek lässt sich direkt in die Anwendungen integrieren, sodass keine weitere Server-Software benötigt wird.

die jeweiligen Komponenten (Kamera, Sensoren, IR-Dioden, Touch-Display) abgestimmt ist.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Adafruit 15a] Adafruit. Add music to your arduino. Online-Quelle. https://learn.adafruit.com/adafruit-wave-shield-audio-shield-for-arduino, 2016.02.28.

[Adafruit 15b] Adafruit. Processing 3.0 - hardware-io. Onine-Quelle. https://learn.adafruit.com/processing-on-the-raspberry-pi-and-pitft/hardware-io, 2016.02.28.

[ard 09] Arduino i^2c bus speed. Online-Quelle. http://forum.arduino.cc/index.php?topic=16793.0, 2016.02.29.

[ard 16] Arducam homepage. Online-Quelle. http://www.arducam.com, 2016.02.29.

[Audet 11] S Audet. Javacv. https://github.com/bytedeco/javacv.

[Axis Corporation 08] Axis Corporation. Axis etrax fs - a flexible multi-processor system-on-chip for network devices. Brochure. http://www.axis.com/files/datasheet/ds_etrax_fs_32179_en_0805_lo.pdf, 2016.02.26.

[Barroso 05] Luiz André Barroso. The price of performance. Queue, 3(7):48–53, 2005.

[Basetech 09] Basetech. Leistungsmesser, cost control 3000. Online-Quelle. http://produktinfo.conrad.de/datenblaetter/125000-149999/125333-an-01-ml-Cost_Control_3000_de_en_nl.pdf, 2016.02.29.

[Bautista Saiz 14] V. Bautista Saiz, F. Gallego. Gpu: Application for cctv systems. Tagungsband: Security Technology (ICCST), 2014 International Carnahan Conference on, Seiten 1–4, Oct 2014.

[Bellard 12] Fabrice Bellard, M Niedermayer et al. Ffmpeg. Availabel from: http://ffmpeg. org, 2012.

[Blake 11] Joshua Blake et al. Openkinect.

[Bradski 00] G. Bradski. Opencv. Dr. Dobb's Journal of Software Tools, 2000.

[Broadcom Europe Ltd 12] Broadcom Europe Ltd. Bcm2835 arm. Datenblatt. https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf, 2016.02.27.

[Brodowski 13] Dominik Brodowski, Nico Golde. Linux cpufreq governors. Linux Kernel. https://www. kernel. org/doc/Documentation/cpu-freq/governors. txt, 2013.

[Chianese 15] Angelo Chianese, Francesco Piccialli, Giuseppe Riccio. Designing a smart multisensor framework based on beaglebone black board. In Computer Science and its Applications, Seiten 391–397. Springer, 2015.

[CircuitCo 14] CircuitCo. Circuitco: Audio cape revb. Online-Quelle. http://elinux.org/CircuitCo:Audio_Cape_RevB, 2016.02.28.

[CircuitCo 16] CircuitCo. Circuitco: Beaglebone 3.1mp camera. Online-Quelle. http://elinux.org/CircuitCo:BeagleBone_3.1MP_Camera, 2016.02.27.

[Coley 13] Gerald Coley. BeagleBone Black - System Reference Manual, rev a5.2 Auflage, 2013. https://www.adafruit.com/datasheets/BBB_SRM.pdf.

[Compaq 00] Intel Compaq, Lucent Intel et al. Universal serial bus specification revision 2.0. Tagungsband: USB Implementers\ Forum, 2000.

[Cucchiara 00] Rita Cucchiara, Massimo Piccardi, Paola Mello. Image analysis and rule-based reasoning for a traffic monitoring system. Intelligent Transportation Systems, IEEE Transactions on, 1(2):119–130, 2000.

[Cutler 00] Ross Cutler, Larry S. Davis. Robust real-time periodic motion detection, analysis, and applications. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22(8):781–796, 2000.

[Cyt 13] Cytron Technologies Sdn. Bhd. Product User's Manual - HC-SR04 Ultrasonic Sensor, v 1.0 Auflage, 05 2013.

[Demaagd 12] Kurt Demaagd, Anthony Oliver, Nathan Oostendorp, Katherine Scott. Practical

Computer Vision with SimpleCV: The Simple Way to Make Technology See. O'Reilly Media, Inc., 2012.

[EEMBC 16] EEMBC. Eembc - industry-standard benchmarks for embedded systems. Online-Quelle. http://www.eembc.org, 2016.02.27.

[elecfreaks.com 15] elecfreaks.com. Ultrasonic ranging module hc - sr04. Online-Quelle. http://www.micropik.com/PDF/HCSR04.pdf, 2016.02.28.

[Elphel, Inc. 15a] Elphel, Inc. Elphel 313 series cameras. Online-Quelle. http://www.elphel.com/model313/, 2015.04.18.

[Elphel, Inc. 15b] Elphel, Inc. Elphel nc353l highly reconfigureable camera. Brochure. http://www3.elphel.com/sites/default/files/Elphel_353_Brochure_2010_V05.pdf, 2015.04.18.

[Elphel, Inc. 15c] Elphel, Inc. Elphel nc353l series cameras. Online-Quelle. http://www3.elphel.com/model_353_cameras, 2015.04.18.

[Felzenszwalb 10] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, Deva Ramanan. Object detection with discriminatively trained part-based models. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 32(9):1627–1645, 2010.

[g7smy.co.uk 13] g7smy.co.uk. Recording sound on the raspberry pi. Online-Quelle. http://www.g7smy.co.uk/?p=283, 2016.02.28.

[Gabrikowski 08] Jens Gabrikowski. User generated content am beispiel einer autonom arbeitenden webcam. Master Thesis, 2008.

[Herity 98] Dominic Herity. C++ in embedded systems: Myth and reality. Embedded Systems Programming, 11(2):48–71, 1998.

[Hodges 06] Steve Hodges, Lyndsay Williams, Emma Berry, Shahram Izadi, James Srinivasan, Alex Butler, Gavin Smyth, Narinder Kapur, Ken Wood. UbiComp 2006: Ubiquitous Computing: 8th International Conference, UbiComp 2006 Orange County, CA, USA, September 17-21, 2006 Proceedings, Kapitel SenseCam: A Retrospective Memory Aid, Seiten 177–193. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Hughes 13] James Hughes. The raspberry pi camera - part 1. The MagPi - A Magazine for

Raspberry Pi Users. https://www.raspberrypi.org/magpi-issues/MagPi14.pdf, 2016.02.27.

[Huppler 13] Karl R. Huppler. Selected Topics in Performance Evaluation and Benchmarking: 4th TPC Technology Conference, TPCTC 2012, Istanbul, Turkey, August 27, 2012, Revised Selected Papers, Kapitel Performance Per Watt - Benchmarking Ways to Get More for Less, Seiten 60–74. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[IMC 16] IMC Viseum. IMC Intelligent Moving Camera CCTV, 2016. http://www.viseum.co.uk/downloads/panoramic-security-camera.pdf.

[Inc. 16] MathWorks Inc. Matlab - computer vision system toolbox. Online-Quelle. http://de.mathworks.com/products/computer-vision/, 2016.02.28.

[Jackson 11] Mike Jackson, Steve Crouch, Rob Baxter. Software evaluation: criteria-based assessment. 2011.

[Krizhevsky 12] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Tagungsband: Advances in neural information processing systems, Seiten 1097–1105, 2012.

[Kruse 98] Eckhard Kruse, Friedrich M Wahl. Camera-based monitoring system for mobile robot guidance. Tagungsband: Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on, Band 2, Seiten 1248–1253. IEEE, 1998.

[Lang 16] Jean-Philippe Lang. Cmucam5: Pixie homepage. Online-Quelle. http://www.cmucam.org/projects/cmucam5, 2016.02.29.

[Le 13] Quoc V Le. Building high-level features using large scale unsupervised learning. Tagungsband: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, Seiten 8595–8598. IEEE, 2013.

[LeanXcam 08] Supercomputing Systems AG LeanXcam. Leanxcam. Online-Quelle. https://github.com/scs/leanXcam/wiki, 2015.04.18.

[LeanXcam 15] Supercomputing Systems AG LeanXcam. Leanxcam end-of-life status. Online-Quelle. https://www.scs.ch/fileadmin/images/leanXcam/EOL_Letter_leanXcam.pdf, 2015.03.19.

[Li 08] J. Li, W. Hao. Research and design of embedded network video monitoring system based on linux. Tagungsband: Computer Science and Software Engineering, 2008 International Conference on, Band 5, Seiten 1310–1313, Dec 2008.

[Lieberman 05] Zachary Lieberman. openframeworks. Online-Quelle. http://openframeworks.cc/about/, 2016.02.28.

[Liu 10] L. Liu. C-based/cached/core computer vision library, a modern computer vision library. Online-Quelle. http://libccv.org, 2016.02.28.

[Mas 15] Massachusetts Institute of Technology. The MIT License (MIT), 2015. http://opensource.org/licenses/MIT.

[Matsugu 03] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. Neural Networks, 16(5):555–559, 2003.

[Max 15] Maxim Integrated Products Inc., 120 San Gabriel Drive, Sunnyvale, CA 94086. DS1307 64 x 8, Serial, I^2C Real-Time Clock, 2015. http://datasheets.maximintegrated.com/en/ds/DS1307.pdf.

[Moler 80] Cleve Moler et al. Matlab - user's guide. Alberquerque, USA, 1980.

[Noble 09] Joshua Noble. Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks. O'Reilly Media, Inc., 2009.

[NVIDIA Corporation 16] NVIDIA Corporation. Jetson embedded platform. Online-Quelle. https://developer.nvidia.com/embedded-computing, 2016.02.27.

[Nvidia 08] CUDA Nvidia. Programming guide.

[Oxer 16] Jonathan Oxer. Arduino shield list. Online-Quelle. http://shieldlist.org, 2016.02.27.

[Par 14] Parallax Inc. PIR Sensor (555-28027), v 2.3 Auflage, 03 2014.

[Patterson 15] David Patterson. Arduino (mega) audio recording. Online-Quelle. http://www.instructables.com/id/Arduino-Mega-Audio-File-logging/, 2016.02.28.

[pic 16] Raspberry pi, how much power does the camera module use? Online-Quelle. https://www.raspberrypi.org/help/faqs/\#cameraPower, 2016.02.29.

[Poovey 09] Jason A Poovey, Markus Levy, Shay Gal-On, Thomas M Conte. A benchmark characterization of the eembc benchmark suite. IEEE Computer Society, 2009.

[Prechelt 00] Lutz Prechelt. An empirical comparison of seven programming languages. Computer, 33(10):23–29, 2000.

[Pul 15] PulsedLight Inc. LIDAR-Lite v2 Overview, 2015. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/lidarlite2DS.pdf.

[Pulli 12] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, Victor Eruhimov. Realtime computer vision with opencv. Queue, 10(4):40:40–40:56, April 2012.

[Qua 14] Qualcomm Technologies, Inc. FastCV SDK Release Notes, 2014. https://developer.qualcomm.com/qfile/27288/releasenotes.pdf.

[Rajan 14] Ajitha Rajan, Subodh Sharma, Peter Schrammel, Daniel Kroening. Accelerated test execution using gpus. Tagungsband: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, Seiten 97–102. ACM, 2014.

[Roberts-Hoffman 09] Katie Roberts-Hoffman, Pawankumar Hegde. Arm cortex-a8 vs. intel atom: Architectural and benchmark comparisons. Dallas: University of Texas at Dallas, 2009.

[Rowe 07] Anthony G Rowe, Adam Goode, Dhiraj Goel, Illah Nourbakhsh. Cmucam3: an open programmable embedded vision sensor. 2007.

[Sasson 91] Steven J Sasson, Robert G Hills. Electronic still camera utilizing image compression and digital storage. US Patent 5,016,107.

[Tex 13] Texas Instruments. OMAP3530 and OMAP3525 Applications Processors, 2013. http://www.ti.com/lit/ds/symlink/omap3530.pdf.

[Tex 94] Texas Instruments. TSL235 - Light to frequency converter, September 1994. http://www.ti.com/lit/ds/symlink/tsl235.pdf.

[Topliss 14] James William Topliss, Victor Zappi, Andrew McPherson et al. Latency performance for real-time audio on beaglebone black. 2014.

[UBM Tech 14] UBM Tech. 2014 embedded market study. Online-Quelle. http://bd.eduweb.hhs.nl/es/2014-embedded-market-study-then-now-whats-next.pdf, 2016.02.28.

[Viola 01] Paul Viola, Michael Jones. Rapid object detection using a boosted cascade of simple features. Tagungsband: Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, Band 1, Seiten I–511. IEEE, 2001.

[Viola 04] Paul Viola, Michael J Jones. Robust real-time face detection. International journal of computer vision, 57(2):137–154, 2004.

[viv 16] Vivanco universal-netzteil. Online-Quelle. http://www.vivanco.de/Startseite/Produkte/Energie/Netzteile-und-Ladeadapter/Universal/vivanco-27822-universal-netzteil.html?id=nsession, 2016.02.29.

[WaveRP 16] WaveRP. Waverp - arduino library for recording and playing wave files. https://code.google.com/archive/p/waverp/, 2016.02.28.

[Wikipedia 16] Wikipedia. Comparison of single-board computers. https://en.wikipedia.org/wiki/Comparison_of_single-board_computers, 2016.02.27.

[Wil 16] Wildlife Acoustics, Inc. Song Meter SM4 - Bioacoustics recorder, 2016. https://www.wildlifeacoustics.com/images/documentation/SM4-USER-GUIDE.pdf.

[WIZ 08] WIZnet Co., Inc. W5100 Datasheet, version 1.1.6 Auflage, 2008. https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf.

[Wol 14] Wolfson Microelectronics. Wolfson Audio Card User Documentation, 1 Auflage, 2014. https://www.element14.com/community/servlet/JiveServlet/downloadBody/65691-102-4-292052/Wolfson%20Raspberry%20Pi%20Soundcard%20Manual%20_%20IMG%20install_%20V1.2.pdf.

[Zahn 14] Klaus Zahn, Mariana Reyes Peres. User documentation for leanXcam. Hochschule Luzern, v 1.00 Auflage, 09 2014.

[Zeppelzauer 13] Matthias Zeppelzauer, Angela S Stöger, Christian Breiteneder. Acoustic detection of elephant presence in noisy environments. Tagungsband: Proceedings of the 2nd ACM international workshop on Multimedia analysis for ecological data, Seiten 3–8. ACM, 2013.