DEVELOPER MANUAL SOFTWARE-INTERNSHIP

Detection of the Hessdalsphenomena in video streams with OpenCV

Adviser
Prof. Hakan Kayal
Ana Vodopivec

Lukas Beierlieb, Lukas Wolz, Jan-Philipp Heilmann

Table of contents

Introduction

This is the developer manual for the RTSP-Observation application which was developed in the winter semester 2016/2017 at the University Würzburg. The application analyses video streams from cameras in the Hessdalen valley in Norway to detect occurrences of the Hessdalsphenomena, for more information to these phenomena check out:

http://www.hessdalen.org

The hole project is developed in Java, so a recent Java Runtime Environment is required for running this application.

For video processing the OpenCV library is needed (http://opencv.org/downloads.html). See readme more details. For transmitting mails this software is using the JavaMail API (https://java.net/projects/javamail/pages/Home). See readme more details. The source archive file includes a git repository. A detailed code documentation is provided by the archived Javadoc.

General

Main

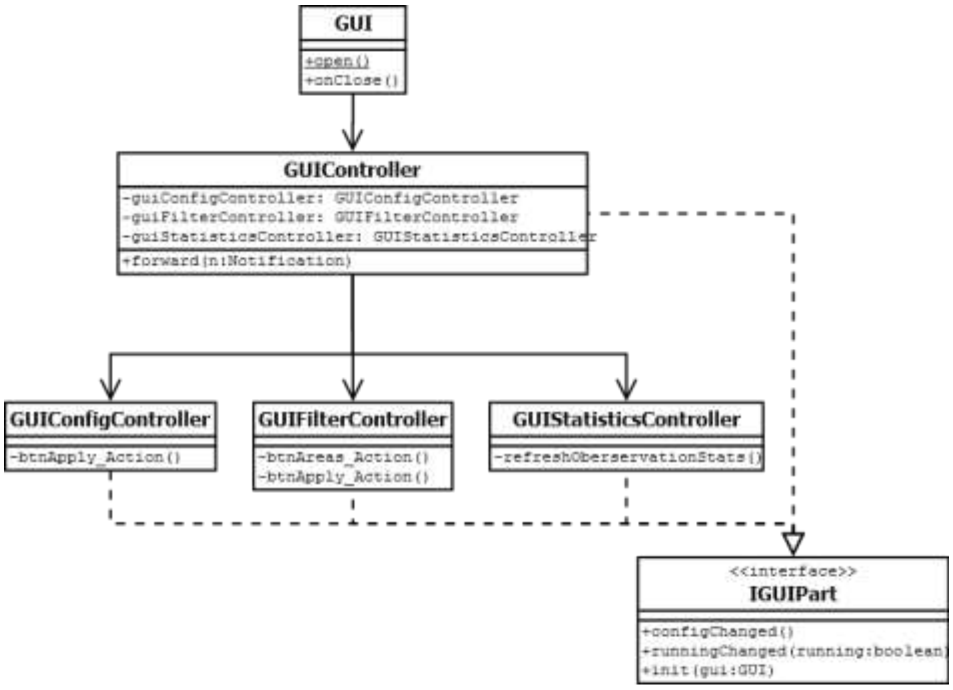
This class contains the central start point of the application and does the parsing of the command-line arguments in processArgs(List<String> args).

Singleton

Class for central storage of data, which should accessible for every point in the application. To set and access data use Singleton.getInstance(). This creates a singleton instance, if not already exists, and returns it.

GUI

RTSP-Observation has a graphical user interface (GUI) for setting up the configuration, performing the observation and viewing information and statistics. It basically consists of three central tabs Config, Filter and Statistics and is completely developed with JavaFX. To improve the maintainability and make comprehensible it is divided into multiple parts. The code is separated from the GUI elements. These are stored in four FXML files, one for each tab content and the central 'GUI.fxml', which consists of elements that are shown independent form the tab selection. The code is divided in the same manner. Each FXML file has its associated controller class such as GUIController. The simplified class diagram below shows the structure of the GUI.



Each controller implements the Interface IGUIPart, which unifies calls and communication. It consists of:

void configChanged(): Will be called, if the configuration file has changed.

void init(GUI gui): Will be called, after Part is added to the central GUI.

void runningChanged(boolean running): Will be called, if the observation status has changed.

The GUI will be directly opened by the main method with GUI.open(). This creates a new instance of GUI and calls start() such as loadGUI(). Where the central FXML file 'GUI.fxml' will be loaded and creates an instance of the associated controller GUIController. The FXML file also includes the other FXML files, which therefore create an instance of matching controllers. Subsequently GUIController will be initialized with init(GUI gui), which also initializes the other controllers. Afterwards the main windows should appear on the screen.

VideoIO

Frame

The Frame class is a wrapper for the OpenCV Mat object. It was needed because the Mat java objects themselves do not need a lot of RAM in the JVM, but allocated quite some memory for the actual image data natively, which means the JVM doesn't know about that. The natively allocated memory is released when mat.release() is called or the Mat object is deleted by the garbage collector. We usually can't rely on the garbage collection because of the small size the Mat objects occupy internally and the big amount of RAM it allocated natively. Therefore we must determine when a Mat objected is not needed anymore and call its release() function. When a frame is created, it will be assumed it is in use by the creator, so the usage counter starts with value 1. If anybody else wants to use the frame (and make sure it won’t get released) he has to call the use() method, and then call the finished() method when he doesn't need the frame anymore. When everybody finished his task on the frame (usage gets down to 0) the mat.release() method is called, cleaning up the now unneeded native memory. At the moment the Frame is class redundant because garbage collection is trigger manually, releasing unused Mat objects and native memory periodically.

FrameLoader

The abstract FrameLoader class is a template for classes that load OpenCV Mat objects and want to share them with other threads that consume them (refer to FrameConsumer). A class implementing this just has to call the setCurrentFrame(Mat) method with its newly loaded Mat, the FrameLoader will take care of wrapping it into a Frame and notifying everybody that is waiting for Frames of this Loader implementation.

RTSPLoader

The RTSPLoader is an implementation of the FrameLoader. It loads its Mat objects through OpenCV's VideoCapture class which can load them from a RTSP stream, but loading from video files is possible as well.

FrameConsumer

The abstract FrameConsumer class is a template for classes that want to do execute some task on frames created by a FrameLoader implementation. It creates itself an own thread for execution which waits for the specified FrameLoader to announce a new Frame which it will hand through to the consume(Frame) method, in which each implementation can do something with it. If multiple FrameConsumer are listening to the same FrameLoader, a new Frame will be processed by every FrameConsumer in parallel, though they may not start at the exact same time. It is important not to modify the content of the Frame since those changes will also apply to everybody else. Instead it is recommended to create a copy of the Frame’s Mat object and modify that.

CanvasUpdater

The CanvasUpdater is an implementation of FrameConsumer, which draws the frames it consumes onto the associated JavaFX ImageView. Its name is a leftover of the time the GUI used a canvas to display images.

IAACanvasUpdater

The IAACanvasUpdater extends the normal CanvasUpdater. It contains an additional IgnoreAreaApplier which draws pink rectangles to the displayed image that are then also shown on screen.

PreEventBuffer

The PreEventBuffer is an implementation of FrameConsumer which buffers the frames that will, if the analyze detects something, be written to disk to see how exactly the objected "spawned". It does that by having a array of Frames and a pointer which tells where the next Frame will be saved. If that index gets so big that is would be pointing outside of the array, it is set at the begin of the array (so it’s basically a ringbuffer). To allow somebody to iterate over the current state of the buffer, its iterator() method copies the current array and provides access to those frames, from the oldest to the newest.

TimeOutChecker

The TimeOutChecker is an implementation of FrameConsumer, which notifies a given VideoIOController when the moment that the last frame occurred is longer ago than specified from its maxDelay attribute.

VideoSaver

The VideoSaver is an implementation of FrameConsumer which is supposed to record parts of the stream and save them on disk. The life of a VideoSaver object goes through following states:

Write PreEventBuffer: get current state of PreEventBuffer, iterate through it and write every frame to file. All frames that occur in that time through consume are saved in the delayBuffer.

Write delayBuffer: go through the delayBuffer and write every frame to file. All frames that occur in that time also append to the delayBuffer, but by sometime the buffer should get empty.

Write live: from then on write every new frame directly to file.

Write postEvent: after calling writePostEventFramesAndStop() the amount of frames written in the PreEventBuffer state should be written down (live, of course) to record how the object disappears. After that the VideoSaver should stop. While it didn’t reach the end quite yet, a call of restart() will put it back into the ‘write live’ state.

A VideoSaver instance should only be used once.

Recorder

The Recorder is basically a wrapper around the VideoSaver. Because the VideoSaver starts as soon as its constructor gets called and can't be reused after stopping, the Recorder provides easy methods to start and stop and manages corresponding VideoSaver instances.

AnalyzeController

The AnalyzeController is an implementation of FrameConsumer. It copies and modifies new frames, so that some areas get ignored (refer to IgnoreAreaApplier). It feeds those to an implementation of the IAnalyzer interface. Depending on the result the analyzation it starts and stops recording to capture those events. It also keeps track of some statistics regarding the analyze.

LibraryLoader

The LibraryLoader is responsible for loading the native libraries for OpenCV. It should only be needed when the program started.

SaveLocationGenerator

The SaveLocationGenerator class mainly provides a method for creating file locations for videos and images that should be saved. It isn't intended to create objects of this class.

IAAChangeController

The IAAChangeController is a JavaFX GUI class which allows to graphically specify areas for the IgnoreAreaApplier. To start it, call the run method with the stream URL and the current ignore area string (can also be empty). When the window is closed, the run method returns the new ignore area string created by the user.

VideoIOController

VideIOController is the class that manages all the other components from videoio.implementation to run a video analysis. It starts as soon as an object is instantiated and stops with a call of kill().

IAnalyzer

IAnalyzer is an interface that allows for multiple analyzing strategies to be implemented. An implementation must define following methods:

boolean analyze(Mat mat): Return if the Mat shows a Hessdalen-event.

void drawFrameWithHighlights(Frame original): If the last analyze(Mat) call returned true, highlight the spot where the object was detected in the given Frame.

Detect

Detect implements the interface IAnalyzer and is mainly for detecting changes in video. It consists of two public methods:

boolean analyze(Mat mat): This method detects only changes in videos by comparing to consecutive frames with each other. Frames are handled as matrices. For each frame the difference to this predecessor will be calculated and afterwards converted into a grayscale. This allows to apply a threshold for removing unwanted changes such as noise. The resulting is a binary frame, where unchanged areas cloud be imagined as black and detected areas as white. Then the white areas will be eroded from their borders to remove for example singe detected pixels. The remaining areas will be dilated that may grows some areas together. This supports detecting a change as one area instead of multiple small changes. This process is compensated, that a big enough area has about the same size after. Subsequently the areas from the binary frame are stored in a list with their approximated size. Now the list can be sorted and filtered to get rid of changes that are too small or too big such as flickering of the whole image. Then it returns true, if a change area is left after the procedure.

void drawFrameWithHighlights(Frame original): Highlight the spots on the given frame, which were found to contain an object (in the previous call of analyze).

Notification

The class Notification only contains the data needed for a notification, were as the NotificationHandler is responsible for forwarding. They could be send as output via console, adding a log entry, messages for GUI or send a mail. There are four different types of notification ERROR, WARNING, INFO and RECOGNITION defined in the enum NotificationType. A notification consists of:

type: Defines the type of notification. If null, INFO would be used.

message: Message of the notification.

attachment: Only used for sending mails. If null or withPreview in configuration is false, no attachment would be sent.

date: Occurrence of the notification. If null, current date would be used.

These values can be set via the constructor of Notification and via getters and setters. Afterwards the instance of Notification still needs to be send with forward(). There are exist other methods for example forwardWithoutMail(). Primary to prevent loops, if SMTPClient forward an error to not get notified again.

Log

The LogHandler creates with its constructor a PrintWriter object which is used by the logNotification method to write notifications to the log file. Every log message goes throw the NotificationHandler, it is not intended to use the LogHandler directly.

SMTPClient

Transmitting mails to SMTP servers or relays. This Class is using the JavaMail API (https://java.net/projects/javamail/pages/Home).

It consists of two public methods:

static boolean checkAddress(String mail): Checks, if a mail address seems to be valid.

static void sendMail(String recipient, String subject, String text, String attachment, Date date): Sends a mail to a SMTP server or relay, if enabled. Transmitter parameters are read from the configuration file. If the SMTP server / rely is not capable of authentication or the password in the configuration file is empty, no authentication will be used. Until now it doesn't support SSL/TLS or STARTTLS encryption, but should be possible with the API.

Configuration

The ConfigIO class provides the loadConfig and saveConfig method, which are need to read and write the configuration file from and to the hard drive. Its methods can only be accessed by the configHandler.

The ConfigHandler provides the general functionality for the configuration, it ensures that the load and save methods are only called with proper values and for example contains a method which can be used to save array in the configuration file.

To ensure the only proper key/value pairs are added to the configuration every configuration parameter gets a getter and a setter method in the Config class which extends the ConfigHandler. To add new parameters simply add the methods to this calls.

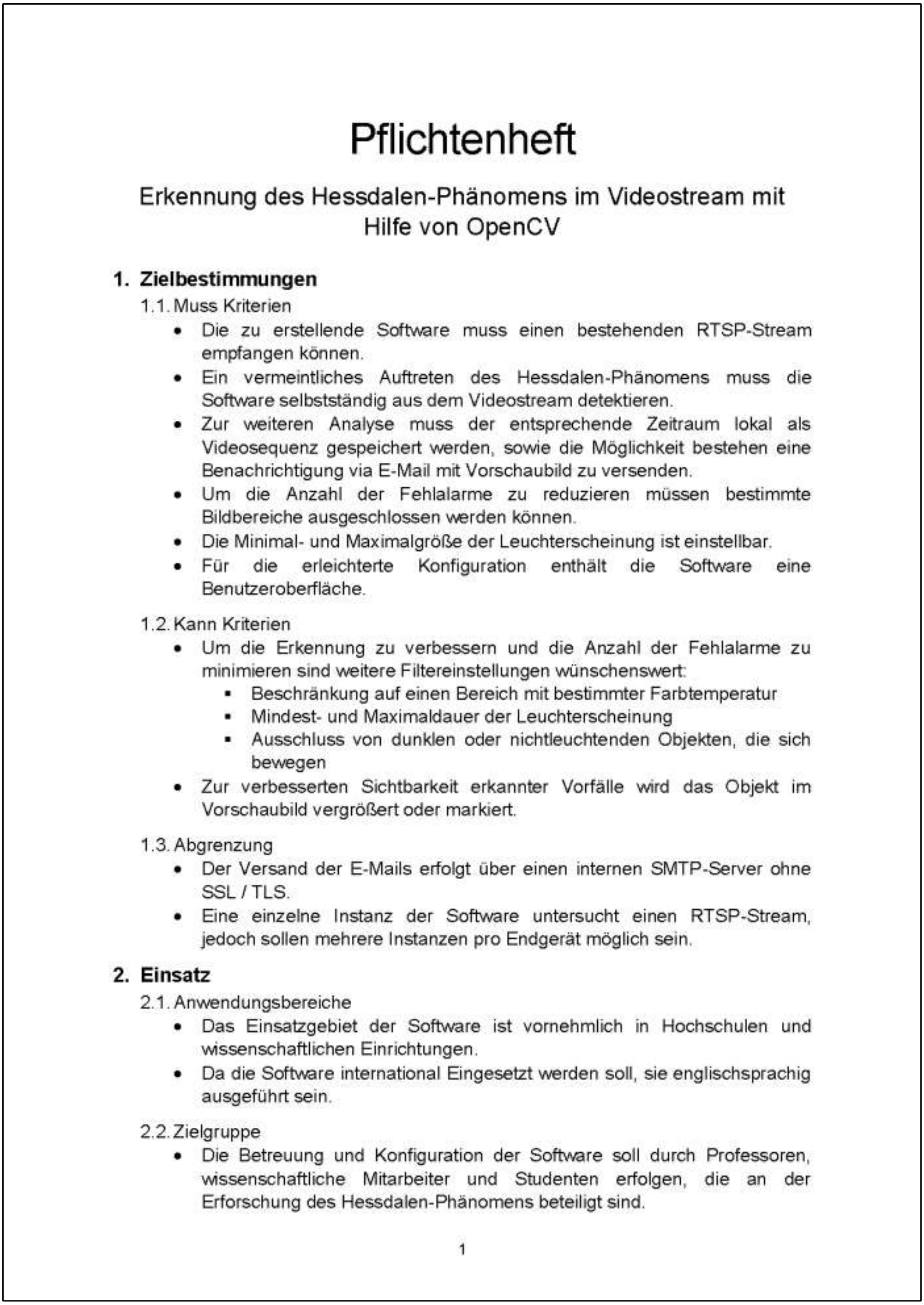
Notes for succeeding projects

To reduce false detections, the recording could be started after in two successive frames will be something detected. Helpful in this manner cloud be an additional filter, which takes color temperature into account, to only detect changes of light.

The functionality to ignore areas could be directly integrated into the main window.

Specification sheet

On the next pages you will find an specification sheet. It is written in German because the customer specified just before completing the document, that the main language of the project is English, especially any user related parts.



Pflichtenheft

Erkennung des Hessdalen-Phänomens im Videostream mit Hilfe von OpenCV

1. Zielbestimmungen

1.1. Muss Kriterien

- Die zu erstellende Software muss einen bestehenden RTSP-Stream empfangen können.
- Ein vermeintliches Auftreten des Hessdalen-Phänomens muss die Software selbstständig aus dem Videostream detektieren.
- Zur weiteren Analyse muss der entsprechende Zeitraum lokal als Videosequenz gespeichert werden, sowie die Möglichkeit bestehen eine Benachrichtigung via E-Mail mit Vorschaubild zu versenden.
- Um die Anzahl der Fehlalarme zu reduzieren müssen bestimmte Bildbereiche ausgeschlossen werden können.
- Die Minimal- und Maximalgröße der Leuchterscheinung ist einstellbar.
- Für die erleichterte Konfiguration enthält die Software eine Benutzeroberfläche.

1.2. Kann Kriterien

- Um die Erkennung zu verbessern und die Anzahl der Fehlalarme zu minimieren sind weitere Filtereinstellungen wünschenswert:
  - Beschränkung auf einen Bereich mit bestimmter Farbtemperatur
  - Mindest- und Maximaldauer der Leuchterscheinung
  - Ausschluss von dunklen oder nichtleuchtenden Objekten, die sich bewegen
- Zur verbesserten Sichtbarkeit erkannter Vorfälle wird das Objekt im Vorschaubild vergrößert oder markiert.

1.3. Abgrenzung

- Der Versand der E-Mails erfolgt über einen internen SMTP-Server ohne SSL / TLS.
- Eine einzelne Instanz der Software untersucht einen RTSP-Stream, jedoch sollen mehrere Instanzen pro Endgerät möglich sein.

2. Einsatz

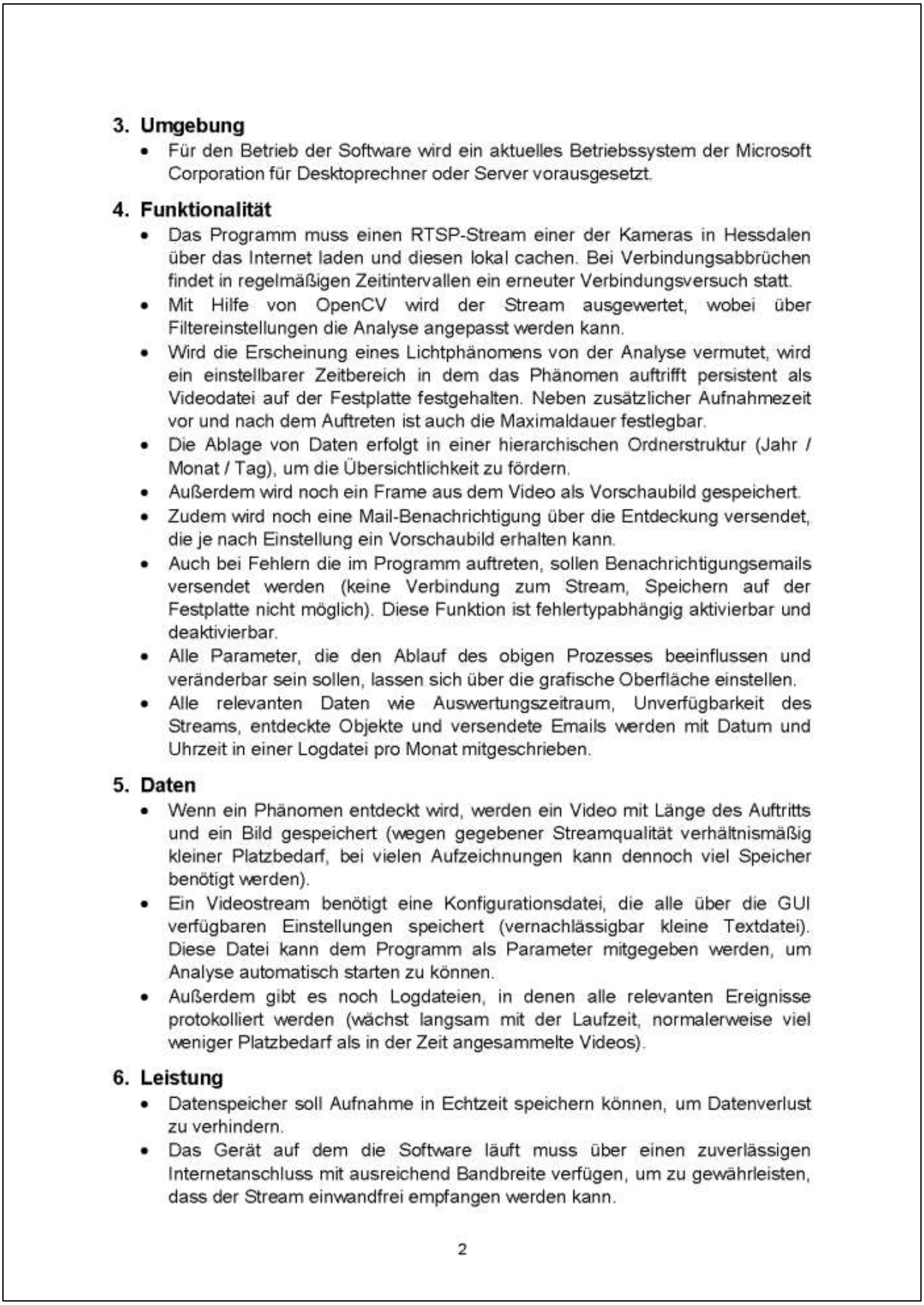
2.1. Anwendungsbereiche

- Das Einsatzgebiet der Software ist vornehmlich in Hochschulen und wissenschaftlichen Einrichtungen.
- Da die Software international Eingesetzt werden soll, sie englischsprachig ausgeführt sein.

2.2. Zielgruppe

- Die Betreuung und Konfiguration der Software soll durch Professoren, wissenschaftliche Mitarbeiter und Studenten erfolgen, die an der Erforschung des Hessdalen-Phänomens beteiligt sind.

1



3. Umgebung

- Für den Betrieb der Software wird ein aktuelles Betriebssystem der Microsoft Corporation für Desktoprechner oder Server vorausgesetzt.

4. Funktionalität

- Das Programm muss einen RTSP-Stream einer der Kameras in Hessdalen über das Internet laden und diesen lokal cachen. Bei Verbindungsabbrüchen findet in regelmäßigen Zeitintervallen ein erneuter Verbindungsversuch statt.
- Mit Hilfe von OpenCV wird der Stream ausgewertet, wobei über Filtereinstellungen die Analyse angepasst werden kann.
- Wird die Erscheinung eines Lichtphänomens von der Analyse vermutet, wird ein einstellbarer Zeitbereich in dem das Phänomen auftrifft persistent als Videodatei auf der Festplatte festgehalten. Neben zusätzlicher Aufnahmezeit vor und nach dem Auftreten ist auch die Maximaldauer festlegbar.
- Die Ablage von Daten erfolgt in einer hierarchischen Ordnerstruktur (Jahr / Monat / Tag), um die Übersichtlichkeit zu fördern.
- Außerdem wird noch ein Frame aus dem Video als Vorschaubild gespeichert.
- Zudem wird noch eine Mail-Benachrichtigung über die Entdeckung versendet, die je nach Einstellung ein Vorschaubild erhalten kann.
- Auch bei Fehlern die im Programm auftreten, sollen Benachrichtigungsemails versendet werden (keine Verbindung zum Stream, Speichern auf der Festplatte nicht möglich). Diese Funktion ist fehlertypabhängig aktivierbar und deaktivierbar.
- Alle Parameter, die den Ablauf des obigen Prozesses beeinflussen und veränderbar sein sollen, lassen sich über die grafische Oberfläche einstellen.
- Alle relevanten Daten wie Auswertungszeitraum, Unverfügbarkeit des Streams, entdeckte Objekte und versendete Emails werden mit Datum und Uhrzeit in einer Logdatei pro Monat mitgeschrieben.

5. Daten

- Wenn ein Phänomen entdeckt wird, werden ein Video mit Länge des Auftritts und ein Bild gespeichert (wegen gegebener Streamqualität verhältnismäßig kleiner Platzbedarf, bei vielen Aufzeichnungen kann dennoch viel Speicher benötigt werden).
- Ein Videostream benötigt eine Konfigurationsdatei, die alle über die GUI verfügbaren Einstellungen speichert (vernachlässigbar kleine Textdatei). Diese Datei kann dem Programm als Parameter mitgegeben werden, um Analyse automatisch starten zu können.
- Außerdem gibt es noch Logdateien, in denen alle relevanten Ereignisse protokolliert werden (wächst langsam mit der Laufzeit, normalerweise viel weniger Platzbedarf als in der Zeit angesammelte Videos).

6. Leistung

- Datenspeicher soll Aufnahme in Echtzeit speichern können, um Datenverlust zu verhindern.
- Das Gerät auf dem die Software läuft muss über einen zuverlässigen Internetanschluss mit ausreichend Bandbreite verfügen, um zu gewährleisten, dass der Stream einwandfrei empfangen werden kann.

2

7. Benutzeroberfläche

- Die Oberfläche verfügt über ein Formular in welchem der zu beobachtende Eingangsstream, sowie die Mail-Adresse welche über potentielle Phänomen-Erkennungen, Warnungen und Fehler informiert werden soll und einen Pfad, wo die aufgezeichneten Videos lokal gespeichert werden sollen, eingegeben werden müssen.
- Benutzerelemente sollen über einen Tooltip mit kurzer Beschreibung verfügen.
- Anschließend hat der Anwender die Möglichkeit verschiedene Filtereinstellungen zu setzen. Hierfür wird auch ein Standbild des aktuellen Streams angezeigt.
- Die Benutzeroberoberfläche bietet auch eine Anzeige des Videostreams mit Statusinformationen, wie beispielsweise:
    - Verbindungsstatus
    - Streaminformationen
    - Trefferstatistiken

8. Qualitätsziele

- Ziel ist die Vermeidung bzw. Minimierung von Falschalarmen, um menschlichen Arbeitsaufwand zu minimieren und Speicherplatz zu sparen. Die genaue Zahl hängt jedoch sehr stark von der Konfiguration und der Qualität des Videostreams ab.
- Da die verfügbaren Videostreams durchgängig ausgewertet werden sollen und die Hessdalen-Phänomene häufig nur für einen kurzen Zeitraum auftreten ist es wichtig, dass die Software mindestens 14 Tage zuverlässig rund um die Uhr läuft.
- Bei der Programmierung wird Wert auf Modularität und ausreichende Dokumentation gelegt, um Wartbarkeit und Erweiterbarkeit zu erleichtern.

9. Ergänzungen

- Das Projekt wird in der Programmiersprache Java realisiert, für die grafische Oberfläche soll JavaFX verwendet werden.
- Für die Analyse und Verarbeitung des Videostreams wird die freie Programmbibliothek OpenCV und dazu bestehende Erweiterungen verwendet.
- Das Projekt ist bis zum 6. Februar 2017 fertig zu stellen.