Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

Professur für Raumfahrttechnik
Lehrstuhl für Informatik VIII

Prof. Dr.-Ing. Hakan Kayal

Bachelorarbeit

Entwicklung, Implementierung und Test von Algorithmen zur Erkennung von Unregelmäßigkeiten in den Bilddaten von Meteosat Wettersatelliten

Vorgelegt von:

Peter Kettig

Matrikelnummer: 1876978

Prüfer: Prof. Dr.-Ing. Hakan Kayal

Betreuer: M.-Space Tech. Borja Garcia

Würzburg, den 10.08.2015

Danksagung

Hiermit möchte ich mich bei allen herzlich bedanken, die mich während der Anfertigung meiner Bachelorarbeit unterstützt haben.

Mein Dank gilt Prof. Dr.-Ing. Hakan Kayal, der mir die Bearbeitung dieses interessanten Themas ermöglicht hat. Ich danke Ihm auch für die freundliche und konstruktive Unterstützung während der Bearbeitungszeit. Des Weiteren möchte ich M.-Space Tech. Borja Garcia danken, der mich während der gesamten Bearbeitungszeit als Betreuer mit seiner herausragenden Expertise bei der Recherche und Fragen unterstützt hat.
Vielen Dank für Zeit und Mühen, die Sie in meine Arbeit investiert haben.

Auch muss ich mich bei meinen Korrekturlesern Frederik und Sophia Meissner sowie meinen Eltern bedanken, die viel Zeit in die Korrektur meiner Arbeit investiert haben. Zahlreiche Kommata, Satzstellungen und Rechtschreibfehler flogen dank ihrer Hilfe hinaus.
Meinen Eltern und meiner Freundin Sophia möchte ich darüber hinaus für die immerwährende Unterstützung in jeglicher Form während meines gesamten Studium danken.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CNES Centre national d'études spatiales

EUMETSAT European Organisation for the Exploitation of Meteorological Satellites

FOV Field of View

GSD Ground Sampling Distance

GUI Graphical User Interface

IFOV Instantaneous Field of View

LTP Lunar Transient Phenomena

LUT Look Up Table

MSG Meteosat Second Generation

NASA National Aeronautics and Space Administration

NEO Near Earth Object

NOAA National Oceanic and Atmospheric Administration

OpenCV Open Source Computer Vision

SETI Search for extraterrestrial Intelligence

SOHO Solar and Heliospheric Observatory

SSA Space Situational Awareness

UTC Universal Time Coordinated

1 Kurzfassung

Ziel dieser Arbeit ist die Entwicklung einer Arbeit, die mit Hilfe der Open Source Computer Vision und anderen C++ Bibliotheken die Bilddaten des EUMETSAT Wettersatelliten Meteosat-10 selbstständig auf Unregelmäßigkeiten auswertet. Dazu wurden fünf Algorithmen entwickelt, die den Kern der Arbeit darstellen: Zwei Algorithmen, die auf helle bzw. dunkle Flecken reagieren und drei, die geometrische Formen innerhalb des Bildes erkennen können.

Gefundene Ereignisse der Algorithmen werden lokal gespeichert und innerhalb einer Benutzeroberfläche angezeigt, die darüber hinaus Einstellmöglichkeiten für die Algorithmen bietet, sowie aktuelle Programmparameter anzeigt. Ebenso wurde ein Email Client eingebunden, der bei einer Detektion automatisch eine Nachricht mit dem Bild im Anhang versendet. Das System soll die bisherigen Programme in der Bodenstation der Universität Würzburg ergänzen und wurde für diesen Anwendungsfall optimiert.

In der Evaluierung des Systems wird gezeigt, dass es dazu in der Lage ist die Bilder autonom auszuwerten und damit Unregelmäßigkeiten zu erkennen. Weiterhin wird die Treffergenauigkeit im regulären Betrieb untersucht. Hier braucht es jedoch aufgrund der wenigen Vorkommnisse während der Entwicklung weitere Untersuchungen bzw. reale Ereignisse um das Programm diesbezüglich einer genaueren Evaluation zu unterziehen.

2 Einleitung

2.1 Wetterbeobachtung mit Satelliten

Viele Bereiche in unserer heutigen Gesellschaft wurden im Laufe der Zeit durch technische Erfindungen revolutioniert. So ist die Wettervorhersage per Smartphone aber ein nicht mehr wegzudenkender Bestandteil des täglichen Lebens. Um die Genauigkeit und Reichweite solcher Vorhersagen zu erhöhen wurden in den letzten Jahren eine Vielzahl von Satelliten in den Weltraum gesendet, die für die unterschiedlichen Regionen der Welt Daten sammeln und zur Erde schicken.
Aber Wettervorhersagen sind nur ein kleiner Teil des vielfältigen Aufgabengebietes von Umweltsatelliten. Auch die Veränderung des Erdklimas, der Erdgeographie, der Einfluss des Menschen auf das Klima und die Folgen des globalen Klimawandels sind Anwendungen, die heutzutage mit Satellitendaten erforscht werden.
Daraus werden beispielsweise weitere Charakteristiken von Vegetationen oder Landvermessungen ([Conway, 1997], S. 1) abgeleitet. Vor allem der Blick "von außen" auf die Erde, in diesem Fall aus einer Bahnhöhe von 400 bis hin zu etwa 36000 km, hilft Wissenschaftlern Systeme besser zu verstehen. Außerdem können Satelliten je nach gewähltem Orbit Bereiche der Erde über einen längeren Zeitraum verfolgen und so Veränderungen besser sichtbar machen. Mit dem neu gewonnenen Wissen über unseren Heimatplaneten wurden viele Bereiche der Wissenschaft erst möglich und Zusammenhänge zwischen Ozeanen, der Atmosphäre und Landmassen erstmalig sichtbar. Das hilft wiederum bei Entscheidungen und Vorhersagen bezüglich dieser Systeme. Darüber hinaus zeichnet sich seit einigen Jahren ein weiterer Trend ab:
Waren die Daten der Satelliten früher ausschließlich den Raumfahrtorganisationen und einer begrenzten Anzahl Wissenschaftlern vorenthalten, ist es Dank immer schneller werdender Verbreitung von Daten heute jedem möglich diese Fakten Einzusehen und mit ihnen zu arbeiten. Auf diese Weise können auch Lehranstalten und private Forschungseinrichtungen davon profitieren.

2.2 Überblick über Meteosat-10 und Eumetcast-System

Da der Betrieb von Satelliten aber auch heute noch teuer, zeitaufwändig und risikoreich ist, gibt es bisher wenige Betreiber. So werden Europa und Afrika hauptsächlich von Eumetsat bzw. deren Satelliten abgedeckt, einer zwischenstaatlichen Organisation, die 1986 gegründet wurde und zur Zeit 30 Mitgliedsstaaten hat, darunter auch Deutsch-

land[1]. Die Spitze ihrer Satellitenflotte besteht dabei aus vier geostationären Meteosat Wettersatelliten, unterstützt von drei Metop Satelliten in Polaren Orbits sowie Jason-2, einem gemeinsamen Projekt von Eumetsat, NOAA, CNES und der NASA zur Meeresbeobachtung.
Meteosat-10 ist dabei zusammen mit dem erst kürzlich gestarteten MSG-4[2] der Modernste aus dieser Reihe. An Bord befinden sich mehrere Nutzlasten, wobei als Primär-Nutzlast das SEVIRI-System mitgeführt wird. Dahinter verbirgt sich ein hochauflösender Bildgeber, der in insgesamt 12 Spektralbereichen Aufnahmen der Erde macht.

Meteosat-10 sendet kontinuierlich alle 15 Minuten 13 neue Bilder zur Erde, was im Folgenden als ein 'Bilderset' (bzw. kurz 'Set') bezeichnet wird. Der Satellit arbeitet dabei in UTC (Universal Time Coordinated, koordinierte Weltzeit), weshalb im Folgenden ebenso diese Zeitskala eingehalten wird. In diesem Set enthalten sind elf Bilder mit einer Ground-Sampling-Distance (GSD, Bodenauflösung eines Pixels) von 3 km und 2 Bilder mit einer GSD von 1 km (HRV-Kanal, High-Resolution-Visible Spectrum). Die elf Bilder teilen sich wiederum in zwei Bilder im sichtbaren (VIS, visible) Bereich, sowie neun Aufnahmen im Infrarot (IR) Bereich auf. Die neun Kanäle im IR-Band sind so eingestellt, dass zwei den Wasserdampf (WV, water vaporous) in der Atmosphäre zwischen 5 und 10 km und jeweils einer den Ozon- und Kohlenstoffdioxidgehalt messen. Tabelle 1 gibt einen Überblick über die Kanäle, sowie ihren charakteristischen Wellenlängen. Durch seine Position in einem geostationären Orbit bei 0°E ist der Satellit in der Lage zu jeder Tages- und Nachtzeit Bilder

Channel	Bands	Centre Wavel..	Spectral Band (99% energy limits)
		µm	µm
HRV	Visible &	(0.75)	broadband (peak within 0.6 - 0.9)
VIS0.6		0.635	0.56 - 0.71
VIS0.8	Near	0.81	0.74 - 0.88
IR1.6	IR	1.64	1.50 - 1.78
IR3.9	Window	3.92	3.48 - 4.36 (98% energy limits)
IR8.7		8.70	8.30 - 9.10 (98% energy limits)
IR10.8		10.80	9.80 - 11.80 (98% energy limits)
IR12.0		12.00	11.00 - 13.00 (98% energy limits)
IR6.2	Water Vapour	6.25	5.35 - 7.15
IR7.3		7.35	6.85 - 7.85 (98% energy limits)
IR9.7	Ozone	9.66	9.38 - 9.94
IR13.4	Carbon-dioxide	13.40	12.40 - 14.40 (96% energy limits)

Tabelle 1: Eigenschaften der Kanäle von Meteosat-10 (Quelle: Eumetsat)

[1]Siehe: `http://www.eumetsat.int/website/home/AboutUs/WhoWeAre/index.html`, Aufgerufen am: 17.07.15
[2]Siehe `http://www.eumetsat.int/website/home/News/DAT_2696903.html`, Aufgerufen am: 01.08.2015

vom europäischen und afrikanischen Kontinent, Teilen des Atlantiks, Südamerikas sowie der arabischen Halbinsel aufzunehmen. Ein Bild wird zeilenweise zusammengesetzt, wobei das Nachführen von SEVIRI automatisch durch die Eigendrehung des Satelliten geschieht, die ihn gleichzeitig spinstabilisiert. Die Aufnahmen von Meteosat-10 werden zunächst an das zentrale Rechenzentrum von Eumetsat in Darmstadt geschickt, wo die Bilder zur sog. Level-1.5-Data prozessiert werden (siehe [Müller, 2007]). Dazu wird der Teil des Bildes weggeschnitten, der nicht die Erdscheibe darstellt, ebenso wie ein Zentrieren der Erde auf dem Bild. So wird sichergestellt, dass dem Endnutzer jederzeit der gleiche Bildausschnitt bereitgestellt wird. Anschließend werden die Bilder an das EumetCast-System übertragen, welches die Aufnahmen über ein Netz von Kommunikationssatelliten global verteilen kann.

2.3 Vorstellung des Lehrstuhls und dessen Bodenstation

Um die vom Eumetcast System vesandten Daten empfangen zu können benötigt jeder Nutzer eine Bodenstation, wie sie der Lehrstuhl für Informatik VIII an der Universität Würzburg besitzt. Diese besteht grundlegend aus einer Parabolantenne mit LNB (Low-Noise-Block, Rauscharmer Signalumsetzer) sowie einem Empfangs-PC mit DVB-Karte, auf dem die Daten entschlüsselt und angezeigt werden können.
Da für den Empfang der Daten keinerlei Restriktionen gesetzt sind (Jeder der eine Sat-Antenne hat, kann mit passender Ausrichtung auch Eumetcast-Signale empfangen), muss für eine kontrollierte Verteilung der Bilder eine Verschlüsselung eingesetzt werden, die es an dieser Stelle wieder zu entschlüsseln gilt. Dafür setzt Eumetcast auf eine Autorisierungs-Software (TelliCast), die an einen physikalischen Datenträger (dem Eumetcast Uni Key) gekoppelt ist. Damit ist jedoch nur gewährleistet, dass die Bilder entschlüsselt werden - tatsächlich zusammengesetzt werden sie von David Taylor's MSG Data Manager.
Dieser setzt die einzelnen Datenpakete, die kontinuierlich während der 15 Minuten für ein Set kommen, zu einem Bild zusammen und löscht anschließend die Source-Dateien. Ebenso kann dort eingestellt werden, welche der Kanäle bzw. Satelliten empfangen werden sollen, denn er ist ebenso für Meteosat-7 und 9 einsetzbar.

2.4 Hintergrund der Arbeit

Zeitweise beinhalten die Aufnahmen dieser Satelliten aber auch Unregelmäßigkeiten. Zu diesen sog. Himmelsphänomenen gehören beispielsweise die Polarlichter. Das erste mal erwähnt wurden sie in Schriftstücken um etwa 2600 v. Chr. aus Regionen des heutigen China, während ihr Ursprung erst Anfang des 20. Jahrhunderts eindeutig geklärt werden konnte[3]. Doch bis heute gibt es Erscheinungen in der Atmosphäre, die die Wissenschaft noch nicht erklären kann. Nicht nur auf unserem Planeten wird nach ihnen gesucht - auch auf fernen Himmelskörpern, wie dem Zwergplaneten Ceres, dessen helle Flecken für Raumfahrtorganisation weiterhin rätselhaft sind[4]. Meistens wird die Suche nach dem Ursprung aber dadurch erschwert, dass viele Phänomene nur in bestimmten Situationen auftreten.
Um Klarheit über Unregelmäßigkeiten in der Erdatmosphäre zu schaffen ist es deshalb sinnvoll, die empfangenen Bilddaten von Meteosat-10 heran zu ziehen und diese autonom auf Unregelmäßigkeiten zu untersuchen. Dazu sollten unterschiedliche Algorithmen vorliegen, die nach den spezifischen Eigenschaften von Anomalien, wie etwa hellen Flecken oder bestimmten Formen, suchen und diese in den Bildern markieren, so dass sie später durch den Menschen untersucht werden können.

3 Aufgabenstellung

Entwicklung, Implementierung und Test von Algorithmen zur Erkennung von Unregelmäßigkeiten in den Bilddaten von Meteosat Wettersatelliten

Ziel dieser Arbeit ist es, ein Programm zu entwickeln, welches auf der Bodenstation des Lehrstuhls für Informatik VIII betrieben werden kann und die empfangenen Bilder von Meteosat-10 automatisch auf Unregelmäßigkeiten untersucht. Dazu sollen verschiedene Algorithmen entwickelt, implementiert und getestet werden, die die Aufnahmen anhand festgelegter Eigenschaften untersuchen. Darüber hinaus soll bei einer Detektion eine Meldung ausgegeben werden, die den Nutzer darüber informiert. Ebenso ist für das Programm eine grafische Oberfläche (GUI) zu implementieren, die die Ergebnisse und

[3] Siehe: `https://www.nasa.gov/mission_pages/themis/auroras/aurora_history.html`, Aufgerufen am: 17.07.15
[4] Siehe `https://www.nasa.gov/jpl/dawn/ceres-bright-spots-come-back-into-view`, Aufgerufen am 01.08.2015

den aktuellen Zustand adäquat präsentiert. Damit die Algorithmen korrekt ausgeführt werden, sind diese möglichst effizient in das Programm einzubauen.
Die Arbeit an der Entwicklung ist ausführlich zu dokumentieren und evaluieren. Dazu gehört ebenso eine Betrachtung bereits existierender Systeme und Methoden. In der Evaluation wird zudem verglichen, inwiefern die gestellten Anforderungen erreicht wurden.

4 Stand der Technik

Da die Suche nach Himmelsphänomenen bereits seit längerem existiert, werden im Rahmen dieser Arbeit unterschiedliche Projekte dargestellt, die sich ebenfalls damit befassen und mit dem zu Entwickelnden verglichen. Dazu wird zuerst auf die Observation von Lunar Transient Phenomena auf dem Mond eingegangen, anschließend werden das SOHO-Projekt der ESA sowie die SkyCam der Universität Würzburg näher betrachtet. Diese Projekte beschäftigen sich zwar alle mit den Entdeckung von Unregelmäßigkeiten auf bzw. bei Himmelskörpern, jedoch ist die SkyCam das einzige Instrument, welches ebenso wie die geplante Software für den Einsatz bei Aufnahmen der Erde konzipiert wurde.

4.1 Lunar Transient Phenomena Observation

Lunar Transient Phenomena (LTP) sind Leuchterscheinungen auf der Oberfläche des Mondes. Charakteristisch für ein solches Phänomen sind gelbe bis rot erscheinende Punkte, die von einigen Sekunden bis hin zu mehreren Stunden teilweise sogar für das bloße Auge sichtbar sind ([Roa und Felipe, 2012]).
Ihre Entstehung ist bis heute nicht eindeutig geklärt, deshalb wird die Detektion dieser Anomalien weiterhin vorangetrieben. Ein Beispiel für ein solches Projekt ist die im Rahmen eines Praktikums am Lehrstuhl für Informatik VIII der Universität Würzburg von Helge Mohn entwickelte "Moonspy"-Software. Grundlage für die Beobachtung der LTP ist eine hochauflösende Kamera, deren Field of View (FOV, Sichtfeld des Aufnahmegeräts) so gewählt wurde, dass sie den gesamten Mond, egal ob im Apogäum oder Perigäum, aufnehmen kann.[5]. Kombiniert mit einer Sternenkamera auf der Vorrichtung wird das System exakt auf den Mond ausgerichtet. Es ergibt sich eine Anordnung, die

[5]Siehe `http://www8.informatik.uni-wuerzburg.de/mitarbeiter/kayal0/studentische_aktivitaeten/tlp_observation/`, Aufgerufen am 02.08.2015

den Mond mit 50 Bildern pro Sekunde und einer GSD von 5 km aufnehmen kann.
Die entwickelte "Moonspy"-Software erhält die Daten des Systems und prüft anhand mehrerer Frames (Bilder), ob eine unregelmäßige Helligkeitsänderung vorliegt und nimmt in diesem Fall automatisch das Videomaterial auf, um es so später analysieren zu können.

4.2 ESA Solar and Heliospheric Observatory

Aber Unregelmäßigkeiten existieren nicht nur auf der Erde und ihrem Trabanten, viel häufiger kommen sie auf der Sonne vor und beeinflussen in vielerlei Hinsicht unser Alltagsleben, etwa wenn durch erhöhte Sonnenaktivität Kommunikationssysteme gestört werden. Um solche und weitere unregelmäßig vorkommende Ereignisse auf und innerhalb der Sonne sichtbar zu machen, wurde 1995 das Solar and Heliospheric Observatory (SOHO) aus einer Kooperation von ESA und NASA gestartet. Es umkreist die Sonne im erdzugewandten Teil in 1.5 Millionen Kilometer Entfernung im Lagrange-Punkt L1. An Bord befinden sich zwölf Instrumente, die kontinuierlich den Zustand der Sonne aufzuzeichnen und zu analysieren, darunter mehrere Spektrometer und ein Teleskop. Auf Basis dieser Daten können unter anderem Vorhersagen für das Sonnenwetter gegeben werden.
Neun dieser 12 Instrumente entstanden dabei aus einer Kollaboration von Europäischen Wissenschaftlern. Mit ihnen konnte erstmals die Struktur von Sonnenflecken unter der Oberfläche sichtbar gemacht werden [Fleck, 2013].
Die Liste an Erkenntnissen aus den Daten von SOHO soll aber noch nicht enden, denn trotz seiner relativ langen Lebenszeit soll es noch bis 2016 operativ bleiben[6].

4.3 SkyCAM

Dass die Beobachtung von Himmelsphänomenen nicht immer vom Weltraum aus geschehen muss, zeigt die SkyCAM des Lehrstuhls für Informationstechnik für Luft- und Raumfahrt der JMU-Würzburg. Sie ist eine autonome Observationsplattform, um sich bewegende Objekte am Himmel zu detektieren. Dazu gehören sowohl Flugobjekte (z.B. Vögel, Flugzeuge) bzw. Wolken, als auch unregelmäßige Ereignisse, wie etwa Meteoriten oder Leuchterscheinungen in der Atmosphäre.

[6]Siehe `http://sci.esa.int/director-desk/51944-esa-science-missions-continue-in-overtime`, Aufgerufen am 02.08.2015

Basis für die Untersuchung ist eine einfache Webcam, die an einen Windows XP Computer angeschlossen ist und Richtung Himmel zeigt. Sie hat eine Auflösung von 1280x720 Pixeln bei einer Bildwiederholrate von 30fps und ein FOV von 68.5° (Siehe [Kayal, 2015], S. 19). Die SkyCAM ist Basis für mehrere Algorithmen, die anhand von mehreren charakteristiken bewegende Objekte erkennen und analysieren können. Wird eine Unregelmäßigkeit entdeckt, so gibt die implementierte Software automatisch eine Meldung aus.
Hintergrund dieser Arbeit ist das Space Situational Awareness (SSA) Programm der ESA und anderer Raumfahrtorganisationen zur Überwachung und Verfolgung von Weltraumobjekten, hauptsächlich im Nahfeld der Erde. Dazu zählen drei Komponenten (vgl. [Kayal, 2015], S.20):

- Weltraumschrott
- Weltraumwetter
- Near Earth Objects (NEOs)

Diese stellen für operative Satelliten und die Erde selbst eine konstante Bedrohung dar, weshalb jedes gefundene Objekt katalogisiert und dessen Risiken für andere Objekte ermittelt werden. Ein großes Problem bei der Verfolgung solcher Objekte ist, dass es mehrere Millionen gibt. Davon haben die meisten einen Durchmesser unter 10cm, die aber nur sehr schwierig von der Erde aus zu entdecken sind. Doch auch größere Objekte sind noch nicht vollständig analysiert und erfordern deshalb, dass der Himmel kontinuierlich nach ihnen abgesucht wird. Einen Beitrag dazu leistet die SkyCAM, die in der Lage ist Tag und Nacht nach solchen Objekten zu suchen, die etwa beim Verglühen ähnlich in Erscheinung treten wie Meteoriten.

Abbildung 1: Die SkyCAM der Universität Würzburg (Quelle: JMU Würzburg)

Diese Plattform stellt für die in dieser Arbeit entwickelte Software eine gute Grundlage dar, an der sich das in dieser Arbeit Entwickelte Konzept orientiert - Ziel beider ist es, mit eigens entwickelten und implementierten Algorithmen unbekannte Elemente auf Bilddaten auszumachen. Jedoch wird diese Arbeit sich mit der *entgegengesetzen* Perspektive beschäftigen - der aus dem Weltraum.

5 Betrachtung einzelner Elemente in den Aufnahmen von Wettersatelliten

Um überhaupt Unregelmäßigkeiten zu erkennen, muss zunächst klar sein, welche Objekte auf Aufnahmen von Wettersatelliten vorzufinden sind. So können Regelmäßigkeiten erkannt und später gezielt gefiltert werden[7].
Es ist aufgrund der gegebenen GSD möglich, eine Grenze anzugeben, ab der Gegenstände nur noch teilweise oder gar nicht mehr dargestellt werden können. Dazu gilt Gleichung 1 (nach [Ley et al., 2009]):

$$GSD = 2 * H_0 * \tan\left(\frac{IFOV}{2}\right) \tag{1}$$

Und für so für die GSD in Abhängigkeit von einer Höhe H:

$$GSD(H) = 2 * (H_0 - H) * \tan\left(\frac{IFOV}{2}\right) \tag{2}$$

Für das Instantaneous Field of View (IFOV) - der Winkel, den ein Pixel vom gesamten FOV einnimmt (vgl. [Ley et al., 2009]) gilt:

$$IFOV = \frac{FOV}{N_{Pix}} \tag{3}$$

Wobei N_{Pix} die Breite/Höhe des Bildes in Pixeln darstellt. Da die Aufnahmen von Meteosat-10 quadratisch sind, ist keine weitere Unterscheidung nötig. Wird nun Gl. 3 in 2 eingestzt, so ergibt sich für die GSD in Abhängigkeit von der Höhe:

$$GSD(H) = 2 * (H_0 - H) * \tan\left(\frac{FOV}{2 * N_{Pix}}\right) \tag{4}$$

Zu beachten ist zudem, dass der Nullpunkt von H der senkrechten Schnittebene des Nadir, dem Fußpunkt gegenüber dem Zenit, entspricht - Die GSD eines Punktes am äußeren Rand der Erdscheibe am Äquator entspricht also nicht der eines Punktes genau in der Mitte dieser, da ihr Höhenunterschied zueinander dem Erdradius[8] entspricht. Besser beobachtbar wird ein Objekt, je weiter es sich von der Erdoberfläche entfernt befindet. So könnte es durchaus möglich sein etwa große Flugzeuge, die 20km über dem Boden fliegen oder die ISS in 400km Höhe bzw. sogar andere Satelliten zu entdecken.

[7] Dabei sei jedoch angemerkt, dass der Umfang dieser Arbeit nicht ausreicht, um alles aufzuzählen, was sich darin finden lässt

[8] $r_E = 6.378$ km (nach WGS-84)

Für alle Fälle wurde dabei von der GSD des zwölften Kanals, also 1 km auf Meereshöhe am Äquator, ausgegangen.
Aufschluss über die Ergebnisse gibt Tabelle 2. Es ist zu erkennen, dass die GSD bei geringer werdendem Abstand größer wird. Jedoch reicht sie bei Weitem nicht aus, um kleine Gegenstände, wie Flugzeuge oder sogar die ISS mit einer strukturellen Länge von 109m ([9]) erkennbar zu machen. Auch Satelliten bieten nicht die geforderte Größe, um überhaupt als ein Pixel sichtbar zu sein, sofern sie sich nicht direkt vor dem Satelliten befinden.

Objekt	**Höhe [km]**	**GSD(H) [km]**
Flugzeug	20	1.00
ISS	400	0.99
GPS-Satellit	20200	0.43

Tabelle 2: GSD in der Höhe von Flugzeugen, der der ISS sowie GPS-Satelliten

5.1 Regelmäßig auftretende Vorgänge

Die wohl am häufigsten auffindbaren Elemente in den Aufnahmen von Meteosat-10 sind die drei Bestandteile aus Ozeanen bzw. anderen Gewässern, Landmassen sowie Wolkengebilden. Im Folgenden soll nicht auf deren meteorologische Interpretation eingegangen werden, sondern vielmehr auf Eigenschaften, die dazu führen können, dass sie als "regelmäßig" bzw. "unregelmäßig" eingestuft werden können. Dazu wird mit Abbildung 2 exemplarisch ein Satellitenbild auf den Kanälen 1-11 gezeigt und verglichen, wie sich die drei Elemente auf den

Clouds		**Surface features—bare areas/soils**	
Cumulonimbus, large and tall	92	Snow, freshly fallen	75–90
Cumulonimbus, small, tops at 6 km	86	Snow, 3 to 7 days old	40–70
Cirrostratus, thick, with lower clouds	74	White Sands, New Mexico	60
Cumulus with stratocumulus	69	Sand dune, dry	35–45
Stratocumulus	68	Soil, dry light sand	25–45
Stratus, thick (0.5 km), over ocean	64	Soil, dry clay or gray	20–35
Stratocumulus masses, with cloud sheet, over ocean	60	Sand dune, wet	20–30
Stratus, thin, over ocean	42	Concrete, dry	17–20
Cirrus, alone, over land	36	Soil, moist gray	10–20
Cirrostratus, alone, over land	32	Soil, dark	5–15
Cumulus, fair weather	29	Road, blacktop	5–10
Water features		**Vegetative zones**	
Sunglint on Gulf of Mexico	17	Desert	25–30
Lake, Great Salt lake, Utah	9	Savanna, dry season	25–30
Ocean, Gulf of Mexico	9	Crops	15–25
Ocean, Pacific	7	Savanna, wet season	15–20
		Tundra	15–20
		Chaparral	15–20
		Meadows, green	10–20
		Forest, deciduous	10–20
		Forest, coniferous	5–15

Tabelle 3: Approximiertes Albedo für verschiedene Oberflächen der Erde (Quelle: [Conway, 1997])

[9]Siehe `http://www.nasa.gov/mission_pages/station/main/onthestation/facts_and_figures.html`, Aufgerufen am 03.08.2015

unterschiedlichen Wellenlängen der Kanäle verhalten:

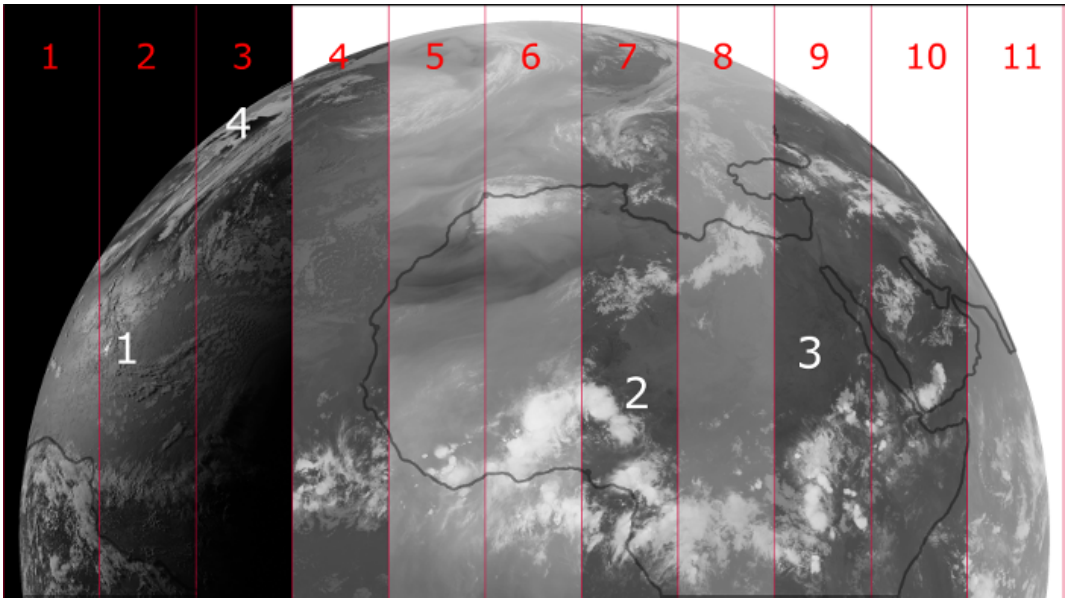

Abbildung 2: Zusammengesetztes Bild vom 23.05. 20:00h aus Kanälen 1-11 mit skizzierter Kontur der Landmassen sowie möglichen Ursachen für (Un-) Regelmäßigkeiten (1-4)

Punkt 1 zeigt auf eine Sonnenreflexion auf Wasser und Wolken, die vor allem im sichtbaren Bereich erkennbar ist (Kanal 1-2). Ihre Position ist abhängig von Uhrzeit und aktueller Deklination der Sonne. Da dies eine Regelmäßigkeit darstellt, sollten sie nicht durch einen Algorithmus, der helle Flecken erkennen kann, fälschlicherweise als Unregelmäßigkeit eingestuft werden. Dazu ist es sinnvoll ihre Ausmaße und Position exakt zu berechnen. Jedoch ist eine Berechnung ihrer Größe und Helligkeit technisch nicht möglich, da die verschiedenen Oberflächen, an denen das Licht reflektiert wird, unterschiedliche Eigenschaften haben. Einerseits ist das Albedo, also das Reflexionsvermögen von z.B. Vegetationen, Wasser und Wolken unterschiedlich (Tabelle 3) und für den Betrachter nicht immer eindeutig erkennbar, um welchen Untergrund es sich bei einer Reflexion handelt.
Andererseits ist die Brechung der Lichtstrahlen diffus. Meistens ist die Oberfläche von Ozeanen oder Wolken rau. Dadurch ergibt sich ein Rückwerfen der Strahlen in verschiedenste Richtungen, weshalb eine genaue Berechnung der zurückgeworfenen Strahlen, als Vektoren interpretiert, nicht möglich ist. Es sollte deshalb bei der Implementierung ei-

nes Helle-Flecken-Algorithmus darauf geachtet werden, dass er zwar helle Flecken wie eine Sonnenreflexion erkennt, jedoch diese anschließend auch wieder zu filtern weiß.
Punkt 2 in Abb. 2 zeigt auf eine Wolkenformation. Es ist deutlich erkennbar, dass sie in mehreren Kanälen sichtbar ist, aber überall andere Helligkeitswerte besitzt. Hauptsächlich sind Wolken eine Regelmäßigkeit, jedoch kann es vorkommen, dass Wolken ungewöhnliche Formen annehmen - etwa, wenn sie die Form einer geometrischen Figur besitzen. Es wäre deshalb sinnvoll, die Bilder auf bestimmte Formen zu untersuchen.
Punkt 3 liegt auf der Landfläche von Afrika. Aufgrund der GSD von 1 bzw. 3 km müssen Objekte, die sich auf dem Boden befinden eine große Fläche haben, um als Unregelmäßigkeit erkannt zu werden. Deshalb ist ihre Möglichkeit Anomalien aufzuzeigen ebenfalls relativ gering, aber nicht ausgeschlossen. Darüber hinaus emittieren Landflächen vor allem in den WV-Kanälen praktisch nichts, weshalb sie auf deren Bildern als dunkle Fläche erscheinen (Ebenso Punkt 4). Ein Algorithmus, der dunkle Flecken findet, sollte diese deshalb ähnlich wie die Sonnenreflexionen herausfiltern können.

5.2 Unregelmäßigkeiten mit Beispiel und Herkunft einzelner Ereignisse

Nachdem erläutert wurde, welche regelmäßigen Vorgänge es in den Bildern von Meteosat-10 gibt, sollen nun einige Unregelmäßigkeiten und deren Herkunft dargestellt werden. Diese stellen aber nur eine Auswahl an Objekten dar, die im Bild vorkommen können. Es existieren noch viele weitere Anomalien, die aber nur sehr selten dokumentiert wurden und deshalb noch weitgehend unerforscht sind. Auf solche Ereignisse wird im letzten Teilkapitel dieses Abschnittes eingegangen.

Abbildung 3: Midnight Effect auf einem Bild von Meteosat-6 von ’97 (Quelle: Eumetsat 2015)

5.2.1 Bildfehler des optischen Systems

Optische Bildfehler stellen im Bereich der Anomalien ein noch relativ häufiges Vorkommnis dar. Unter ihnen versteht man Fehler, die bei der Bildverarbeitung und -aufnahme stattgefunden haben sowie im weitesten Sinne auch durch Übertragungsfehler hervorgerufene Änderungen im Bild. Häufig handelt es sich aber um aus direkter oder indirekter Sonneneinstrah-

lung resultierende Strukturen. Das SEVIRI-Instrument ist zwar relativ gut innerhalb von Meteosat-10 vor diesen Effekte geschützt und bedingt durch die Eigendrehung des Satelliten entsteht ein automatischer Schutz vor sog. Streulicht. Es kann dennoch vorkommen, dass Sonnenstrahlen in das Instrument treffen und so ungewöhnliche Strukturen hervorrufen. Ein Beispiel für ein solches Ereignis zeigt eine Meteosat Aufnahme von 1997, als durch den Midnight-Effekt mehrere dunkle und helle Flecken sowie bogenartige Strukturen entstanden sind[10]. Der Effekt entsteht, wenn die Sonne für den Satelliten noch teilweise hinter der Erde sichtbar ist, was immer um Mitternacht der Fall ist. In dieser Konstellation fällt Sonnenlicht direkt in das Radiometer und wird durch die mechanische Struktur mehrfach reflektiert, bis es irgendwann auf den Sensor trifft (Abbildung 3).

5.2.2 NEOs und andere Körper

Ein weiterer Grund für Anomalien können Near-Earth-Objects (NEOs) sein, die sich in der näheren Erdumgebung befinden und beim Wiedereintritt verglühen, wodurch ein charakteristischer Schweif entsteht. Es ist bereits bekannt, dass Meteoriten ein solches Verhalten zeigen, jedoch gilt dies auch für Weltraumschrott. Ein Beispiel für ein solches Ereignis zeigt den Eintritt des Meteors von Tscheljabinsk am 15.Februar 2013 (Abbildung 4). Meteosat-10 konnte mit seinem 15-Minütigem Intervall den Eintritt des Meteors festhalten, wobei ein Schweif deutlich erkennbar ist. Da dieser immer gerade entlang der Eintrittsbahn des Meteoriten verläuft, ist es ein Ziel für das zu entwickelnde Programm, Bilder auf gerade Strukturen zu prüfen.

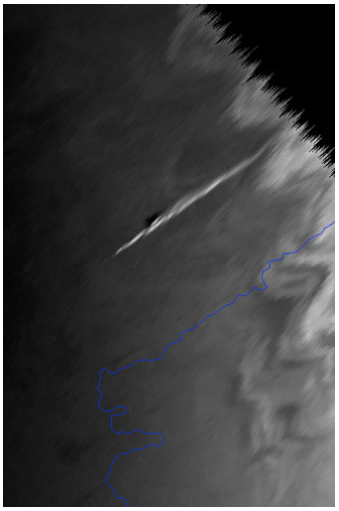

Abbildung 4: Meteor auf einem Bild von Meteosat-10 (Quelle: Eumetsat 2015)

5.2.3 Leuchterscheinungen in der Atmosphäre

Unter Leuchterscheinungen am Himmel fallen bereits bekannte Phänomene, wie Gewitterblitze, Regenbögen sowie die bereits erwähnten Polarlichter. Es gibt darüber hinaus weitere Beobachtungen, die nicht in dieses Schema passen. Dazu gehören die sog. Sprites (dt.: Kobolde), als Blitze aus einer Gewitterwolke

[10]Siehe `http://www.eumetsat.int/website/home/Data/ServiceStatus/AnomaliesonMeteosatImages/index.html`, Aufgerufen am 03.08.2015

in Richtung Weltraum. Sie können bis zu mehreren hundert Kilometern hoch reichen und dabei unterschiedliche Farben annehmen. Sie wurden erst 1994 durch Zufall entdeckt[11].

Es gibt darüber hinaus noch andere Ereignisse, die ebenfalls eine Leuchterscheinung in der Atmosphäre darstellen. Allen ist aber gemeinsam, dass sie nur von sehr kurzer Dauer und deshalb schwer festzuhalten sind. Meteosat-10 besitzt für dieses Einsatzgebiet eigentlich nicht die richtige Aufnahmestuktur[12] - dennoch sollte dieses Szenario nicht ausgeschlossen werden, da ein Fund einer solchen Anomalie durchaus wissenschaftlich relevant wäre.

5.3 Nicht erklärbare Erscheinungen und SETI

Im weiteren Zusammenhang gibt es auch Phänomene, für die es weder Erklärungen noch genug Bildmaterial gibt, um sie näher zu untersuchen. Dazu gehören Sichtungen von UFO-ähnlichen Strukturen am Himmel. Um die eher subjektiven Schilderungen von Ereignissen zu untersuchen, wurden in den vergangenen Jahrzehnten mehrere Organisationen gegründet, die sich mit dem Thema SETI (Search for extraterrestrial Intelligence) beschäftigen.

Häufig ist ein als UFO betitelter Fund nur eine spezielle Wolkenformation, die in Kombination mit Sonnenlicht eine seltsame Struktur annimmt, wie in einem erst kürzlich aufgenommenen Beispiel zu sehen[13]. Manche dieser Fälle lassen sich mit der Hilfe von Satellitenaufnahmen aufklären, andererseits können so auch neue Objekte gefunden werden. Dadurch stellen Satelliten ein wichtiges Instrument bei der Suche nach Erscheinungen dar.

6 Grundlagen der Digitalen Bildverarbeitung

Um die Bilder von Meteosat-10 analysieren zu können, müssen die Bilder digital verarbeitet werden. Was darunter zu verstehen ist, soll in diesem Kapitel erläutert werden.

Es ist allgemein schwierig eine einheitliche Begriffsdefinition für die digitale Bildverarbeitung zu finden, da die Schwelle, ab wann ein Bild verarbeitet wird und wann

[11] Siehe `http://www.nasa.gov/mission_pages/sunearth/news/gallery/BigRed-Sprite.html`, Aufgerufen am 03.08.2015

[12] Die Bilder werden zeilenweise aufgenommen. Das führt dazu, dass kurzzeitige Ereignisse nur selten sichtbar werden

[13] Siehe `http://metro.co.uk/2015/07/07/is-this-a-ufo-disguised-as-a-cloud-video-shows-it-behaving-very-oddly-5284677/`, Aufgerufen am 03.08.2015

nicht unterschiedlich interpretiert werden kann. Letztlich umfasst der Begriff aber nach [Erhardt, 2008] (S.2)

> "eine Vielzahl von Prozessen, deren gemeinsames Ziel es ist, die Gewinnung nützlicher Parameter aus einem Bild oder einer Folge von Bildern zu ermöglichen."

Digitale Bildverabeitung lässt sich in drei Unterkategorien einteilen (vgl. [Erhardt, 2008], S.2):

- Bildbearbeitung:

  Darunter wird auch die Bildaufbereitung verstanden. Diese wird angewandt, wenn das Bild für den Betrachter in einem nicht optimalen Zustand vorliegt, also etwa zu Dunkel oder verrauscht ist. Um dies zu verbessern, wird das Bild künstlich aufgehellt oder ein Filter verwendet. Das Ergebnis der Operation bei optimaler Anwendung ist ein Bild, welches die gewünschten Informationen besser darstellt und sich so besser weiterverarbeiten lässt.

- Bildtransformation:

  Das Ziel der Bildtransformation ist es, die Aufnahme in einen für den Computer intepretierbaren Zustand zu bringen. Dazu gehört etwa die Fouriertransformation oder auch eine Vergrößerung oder Verkleinerung des Bildes.

- Bildauswertung

  Nachdem das Foto mehrere Stufen der Vorprozessierung durchlaufen hat, werden in diesem Schritt die relevanten Parameter entnommen. Im Falle eines Satellitenbildes mit einem hellen Fleck als Unregelmäßigkeit sind das die Helligkeitswerte des Bildes im Bereich, wo der Fleck sich befindet oder auch die Größe von diesem; also der größte und kleinste x bzw. y-Wert mit einer bestimmten Helligkeit größer als ein festgelegter Schwellwert.

6.1 Vorgehensweise beim Arbeiten mit Computer Vision

In der Bildverarbeitung wird "Computer Vision" als Teilbereich angesehen und frei als "*Bildverstehen* oder *Bilderkennen*" ([Erhardt, 2008], S.3) übersetzt. Der Begriff kommt vor allem aus dem Bereich der Roboter (Robotik), da diese oftmals auf ein optisches

System angewiesen sind und entsprechend zuverlässige Algorithmen zur Erkennung der jeweiligen Zielobjekte brauchen.
Es ist hierbei elementar, dass der zu beschreibende Algorithmus die Beschaffenheit seines Ziels bzw. bestimme Parameter von eben diesem kennt. Ein roter Fleck auf einem Bild charakterisiert sich zum Beispiel durch seine Größe, eine bestimmte Helligkeit in den drei Kanälen R,G,B sowie seine Position in kartesischen Koordinaten. Damit der Algorithmus diesen und ähnliche Flecken findet, müssen entsprechende Operationen ausgeführt werden, um diesen Bildbereich von den anderen zu unterscheiden. Meistens greift man dabei auf sogenannte *Threshold-Operationen* zurück. Diese Methode stellt eine der Grundideen der digitalen Bildverarbeitung dar und wird auch in dieser Arbeit für die Entwicklung der Algorithmen mehrfach verwendet. Aus diesem Grund wird dieses Verfahren kurz erläutert.
Die Anwendung eines "Thresholds" (zu deutsch: Schwellwert) entscheidet für jeden Punkt bzw. Bereich eines Bildes, ob der Wert einer bestimmten Ausprägung (also z.B. Sättigung oder Helligkeit eines Kanals) größer oder kleiner als eine gewählte Grenze ist (Gl. 5).

$$Pixel.at(x, y).Brightness > Threshold_{Brightness} \tag{5}$$

Nun gibt es insgesamt sechs Threshold Anwendungen, um mit Werten größer bzw. kleiner als der Schwellwert zu verfahren (nach [Bradski und Kaehler, 2008], S. 136):

- Binary Threshold:

  Alle Werte größer als der Schwellwert werden auf einen Maximalwert max gesetzt, alle kleiner gleich werden zu 0. Da es nur zwei Ergebnisse gibt, wird diese Methode entsprechend als binäre-Schwellwertoperation bezeichnet (Gl. 6).

$$\begin{aligned} if(Pixel.at(x, y).Brightness > Threshold_{Brightness})val = 1 \\ else\ val = max \end{aligned} \tag{6}$$

- Binary Threshold Inverted:

  Wie oben beschrieben ist lediglich die Wertezuweisung vertauscht. Also alle

Werte größer als der Schwellwert werden zu 0, der Rest zu max (Gl. 7).

$$if(Pixel.at(x,y).Brightness > Threshold_{Brightness})val = 0$$
$$else\ val = max \tag{7}$$

- Truncate:

  Bei "Truncate" (zu deutsch: abschneiden) werden Werte größer als der Schwellwert gleich dem Schwellwert gesetzt. So wird die maximale Ergebnismenge gekürzt, die die Werte innerhalb eines Bildes annehmen können. Es gilt Gl. 8.

$$if(Pixel.at(x,y).Brightness > Threshold_{Brightness})val = Threshold_{Brightness}$$
$$else\ val = Pixel.at(x,y) \tag{8}$$

- Threshold to Zero Inverted:

  Hier werden sämtliche Werte größer als der Schwellwert auf 0 gesetzt, der Rest wird beibehalten. Auch hier wird wie bei Truncate die Ergebnismenge reduziert (Gl. 9).

$$if(Pixel.at(x,y).Brightness > Threshold_{Brightness})val = 0$$
$$else\ val = Pixel.at(x,y) \tag{9}$$

- Threshold to Zero:

  Die letzte Methode stellt eine Inversion der oben vorgestellten dar. Alle Pixel kleiner als der Schwellwert, werden auf 0 gesetzt, der Rest wird ohne Bearbeitung weitergegeben. Diese Methode wird in dieser Arbeit am Häufigsten von den eben genannten verwendet. Sie stellt zum Beispiel den Ausgangspunkt für die Helle-Flecken-Erkennung dar (Siehe Kapitel 8.2.1). Es gilt Gl. 10.

$$if(Pixel.at(x,y).Brightness > Threshold_{Brightness})val = Pixel.at(x,y)$$
$$else\ val = 0 \tag{10}$$

Nachdem eines dieser Verfahren durchgeführt wurde, wird die bearbeitete Aufnahme

weiteren Operationen unterzogen, bis ausschließlich der gewünschte Bereich herausgefiltert ist. Häufig ist dabei der Extraktionsprozess der Daten nur ein Teilaspekt. Das finale Ergebnis ist meistens ein Anderes: Etwa ein Roboter, der einen farbigen Ball (roter Fleck) erst erkennen muss, um ihm dann zu folgen.
Die Herausforderungen bei dieser Arbeitsmethode sind einerseits korrekte Schwellwerte für die besagten Threshold-Operationen zu finden und andererseits die richtigen Operationen aneinander zu reihen, sodass am Ende ausschließlich der gewünschte Bereich extrahiert wird.

6.2 Arbeiten mit OpenCV

Die im vorherigen Kapitel beschriebenen Techniken gelten allgemein für jede CV-Bibliothek bzw. generell für die meisten Bildverarbeitungsprogramme. Da in dieser Arbeit aber speziell mit OpenCV in der Version 3.0.0-rc1 gearbeitet wird, soll im Folgenden auf die Arbeitsweise mit dieser C++ Library eingegangen werden.
Die Basis von OpenCV ist mathematisch geprägt, denn alle internen Datenstrukturen repräsentieren mathematische Objekte. Die beiden am häufigsten verwendeten sollen nun erläutert werden.

- **Matrix**

  Damit Bilder verarbeitet werden können, besitzt OpenCV seit Version 2.0 den Datentyp `Mat`. Er speichert Bilder in einem $n \times m$ großen, zweidimensionalen Array, wobei n die Anzahl der Spalten und m die Anzahl der Reihen darstellen. Je nach dem, wie viele Kanäle ein Bild hat wird in jedes Feld entweder ein `uchar` oder ein Vektor (siehe unten) geschrieben. Um Bilder der Reihe nach durchzugehen, sind also einfach zwei `for`-Schleifen notwendig, die die Anzahl der Reihen bzw. Spalten inkrementieren. Da dieses zeilenweise Durchgehen auch in C++ eine schnelle Operation darstellt, ist das Durchgehen eines Bildes von $(0, 0)$ bis (n, m) entsprechend effizient.
  Jede Matrix besteht dazu noch aus einem Header, ohne den die Datenstruktur nicht als solche identifizierbar ist. Er beinhaltet wichtige Informationen über den Zustand der Matrix, etwa die Anzahl der Kanäle, die Bittiefe und Ausmaße eines Bildes. Um ein neues, schwarzes Bild zu erstellen legt OpenCV also zunächst ein Objekt `Mat` mit den im Konstruktor definierten Werten im Header an und füllt diese dann mit Nullen (für die Farbe 'schwarz') auf.

Ein wichtiger Vorteil bei der Verwendung dieses Datentyps ist, dass der benötigte Speicher automatisch belegt und bei Nichtbenutzung auch wieder freigegeben wird, was ein Überlaufen (overflow) des Arbeitsspeichers und den Absturz des Programms verhindert. Darüber hinaus sind alle Bilder gleich handhabbar bzw. können sogar miteinander interagieren, wie z.B. das Anwenden eines Masken-Bildes auf ein Anderes.

- **Vektor**

  Der zweite wichtige Datencontainer ist der sog. Vektor. Er stellt eine Spezialform der Matrix mit einer Größe von $n \times 1$ dar. Er wird immer dann benötigt, sobald in einem Feld einer Matrize mehr als ein Wert gespeichert werden muss, die Anzahl der Kanäle also > 1 ist. Bei RGB-Bildern benötigt man demnach einen 3×1 Vektor, der je nach Bittiefe Daten vom Typ `uchar` bis hin zu `uint64` enthält. So ist es sehr einfach möglich, die einzelnen Werte einer Matrix zu adressieren ohne aufwändige Rechenoperationen durchführen zu müssen.

OpenCV bietet darüber hinaus bereits ein integriertes I/O-Modul mit dem Namen HighGUI zum Öffnen und Speichern von Bildern, weshalb auf einen externen Code verzichtet werden kann. Da die zu verarbeitenden Aufnahmen lediglich 2D-Bilder und kein Videomaterial sind, kann auf einen Großteil des Umfangs von OpenCV verzichtet werden. So reduziert sich die Bibliothek letzten Endes auf die Module *Core*, welches die Basisfunktionalität bietet, *ImgProc* zum Prozessieren der Bilder sowie *HighGUI* mit *ImgCodecs* zum Öffnen und Speichern der Bilder mit jeweiligem Codec.

7 Randbedingungen und Anforderungen an das Programm

Da das zu entwickelnde Programm auf einer bereits bestehenden Bodenstation laufen soll, welche bereits eine Reihe von Programmen enthält, sind die Rahmenbedingungen, dieser Bodenstation unter allen Umständen einzuhalten (Siehe Tabelle 4). So kommen die Bilder mit demselben Zeitstempel wie in Kapitel 2.2 erwähnt alle 15 Minuten (C010) auf jeweils 12 Kanälen an (C020) und sind stets im .jpeg Format gespeichert (C030). Ebenso bearbeitet Eumetsat die Bilder bereits soweit, dass nur noch die Erdscheibe zu sehen ist (C040) und durch die Konstruktion des SEVIRI Instruments liegen alle Bilder nur in Graustufen mit 8-bit Tiefe (1 Kanal) vor (C060).

Nr.	**Randbedingung**	**Wert**	**Priorität**[1]	**Flexibilität**[1]	**Quelle**
C010	Meteosat-10 Bilder kommen kontinuierlich alle 15-Minuten	-	1	10	Titel d. Arbeit[2]
C020	Bilder derselben Uhrzeit kommen auf 12 Kanälen	-	1	10	Titel d. Arbeit[2]
C030	Bilder sind in im .jpeg Format und haben eindeutigen Namen	[Date] [Hour] -msg-ch[Ch-Nr] .jpg	1	10	C010
C040	Bilder sind bereits bearbeitet worden	L1.5 Data	2	8	MSG Data Format Desc.[3]
C050	Bilder eines Tages werden in gemeinsamem Ordner gespeichert	-	3	5	C010
C060	Bilder sind in Graustufen gespeichert und haben 8-bit Tiefe	-	3	6	C010

[1] 1 = Höchste; 10 = Niedrigste
[2] siehe Kapitel 3
[3] siehe: [Müller, 2007]

Tabelle 4: Randbedingungen

Als Hauptanforderung (Tabelle 5) an das Programm wird die grundsätzliche Lauffähigkeit

auf der Bodenstation vorgegeben (R010), während die Benutzeroberfläche und der interne Programmablauf weitgehend unspezifiziert bleiben. Dies erleichtert den späteren Entwicklungsprozess insofern, dass lediglich die Algorithmen effizient in das System eingebunden werden müssen (R040 und R050) und dabei automatisch die Bilder öffnet und nach Bedarf speichert (R020). Da es sich bei der Bodenstation nicht um ein Echtzeitsystem handelt, wird auf Angaben zur Ressourcennutzung und des Speicherbedarfs verzichtet. Weitere Aufschlüsse darüber wird erst die Evaluation zeigen.

Nr.	**Anforderung**	**Wert**	**Priorität**[1]	**Flexibilität**[1]	**Quelle**
R010	Lauffähigkeit auf Bodenstation	-	1	10	Titel d. Arbeit[2]
R030	Software soll Meteosat-10 Bilder automatisch erkennen und verarbeiten und gefundene Ereignisse in eigenem Ordner speichern	-	1	10	C010
R040	Software soll helle Flecken erkennen können	Helligkeit > 250	1	10	Titel d. Arbeit[2]
R050	Software soll Formen erkennen können	Kreise, Vier-, Fünf- und Sechsecke sowie Linien	1	10	Titel d. Arbeit[2]
R060	Start und Stopp sowie Neustart Funktion ohne Programm beenden zu müssen	-	3	5	R010
R070	Ausgabe von Dateiname des geöffneten Bildes auf interner Konsole mit Zeitstempel	-	3	7	R010
R080	Aktuelles und letztes Bild mit Unregelmäßigkeit sollen innerhalb des Programms zu öffnen sein	-	2	8	R010

[1] 1 = Höchste; 10 = Niedrigste

[2] siehe Kapitel 3

Tabelle 5: Anforderungen

8 Konzept

8.1 Überblick

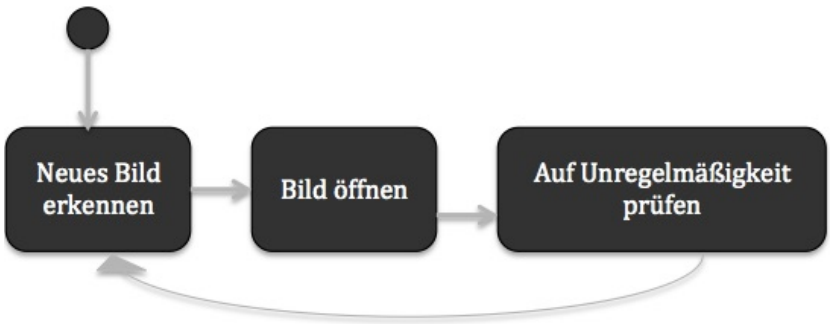

Abbildung 5: Verarbeitungsweg eines Bildes

Um Unregelmäßigkeiten (vgl. Kapitel 5.2) finden zu können, soll ein Programm entwickelt werden, welches auf der Bodenstation die Bilder autonom öffnet und auswertet. Dazu werden eine Reihe unterschiedlicher Algorithmen erstellt, die am Ausgangsbild mit mehreren Operationen Anomalien herausfiltern. Damit Regelmäßigkeiten, wie die aus Kapitel 5.1 nicht erkannt werden, werden mehrere Korrekturalgorithmen angewandt, die eine möglichst hohe Trefferrate bei gleichzeitig wenigen Falschmeldungen haben sollen, um gefundene Phänomene nicht zu verwerfen. Eine Falschmeldung wird im Folgenden als eine Detektion definiert, die aufgrund fehlender Unregelmäßigkeit als Grundlage dennoch als solche erkannt wird.
Damit alle Algorithmen korrekt ausgeführt werden, ist ein Thread-Konzept zu entwerfen, welches mit User-Interface auf dem Zielcomputer läuft und auf die Bilddaten zugreifen kann, sowie gefundene Bilder speichert. Abbildung 5 skizziert für das Konzept den Verlauf eines Bildes innerhalb des Programms. Es ist darüber hinaus auf den Ablauf und Kontrollfluss des Programms sowie auf möglichst effiziente Algorithmen zu achten. Dies ist notwendig, um die Rechenleistung beim Verarbeiten minimal zu halten, wodurch eine Auswertung der Bilder gewährleistet wird.

8.2 Vorstellung und Entwicklung der Algorithmen zur Erkennung von Unregelmäßigkeiten

Nachdem dargestellt wurde, woher die Aufnahmen kommen, welche Eigenschaften sie aufweisen (Siehe Kapitel 2.2) und welche Elemente sich in den Bildern vorfinden lassen

(Siehe Kapitel 5), geht es im Folgenden darum, Algorithmen zu konstruieren, die die Unregelmäßigkeiten zuverlässig finden. Dazu wurden insgesamt fünf Vorgehensweisen entworfen, die teilweise auf denselben Operationen beruhen.

8.2.1 Helle- und Dunkle-Flecken-Erkennung

Zunächst wird ein ungewichteter Algorithmus konstruiert, der Ansammlungen von hellen Pixeln auf kleinem Raum (im Folgenden: heller Fleck) erkennen soll. Der Ansatz ist hier, wie in Kapitel 6.1 erklärt eine Threshold-to-Zero Operation. Der Algorithmus scannt das Bild zeilenweise von $(0, 0)$ bis (x_n, y_n) (siehe Abb. 6) ab und entscheidet für jeden Punkt, ob dieser über dem Schwellwert liegt. Da ein einzelner Pixel noch nicht hinreichend aussagekräftig und evtl. ein Fehler des optischen Systems ist, werden für diesen Fall zusätzlich die Nachbarpixel untersucht. Die Anzahl heller bzw. dunkler Nachbarpixel, die das Zentrum benötigt, um als heller bzw. dunkler Pixel durchzugehen, wird dynamisch berechnet. Das bedeutet, dass für einen hellen Zentrumspixel bereits wenige benachbarte helle Pixel ausreichend sind, während für einen, der nur knapp über dem Schwellwert liegt, mehrere Pixel.
Ebenfalls wird eine Gewichtung der Pixel durchgeführt. Dazu wird zunächst der Bereich definiert, der helle oder dunkle Pixel enthält und dessen optischer Schwerpunkt ermittelt (Abb. 7). Um den Schwerpunkt herauszufinden, verläuft der Algorithmus entlang der Kontur des Bereiches und speichert Konturpixel in einem Array dynamischer Größe, da der Umfang der Kontur vorab nicht bekannt ist. Anschließend wird der Mittelwert über alle Helligkeitswerte der im Bereich liegenden Pixel gebildet und geprüft, ob dieser größer, kleiner bzw. gleich dem Schwellwert ist.

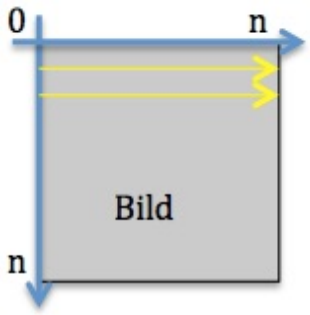

Abbildung 6: Durchlauf eines Bildes

Beide Varianten reduzieren die Fleck-Findung auf einen Pixel und Helligkeitswert. Dabei ergibt sich in beiden Fällen das Problem, dass sie in der nächsten Zeile erneut auf dieselbe Pixelformation treffen, wie in der Zeile zuvor. Um doppelte Treffer zu vermeiden, ist daher der zuvor bereits bearbeitete Bereich zu maskieren - also für weitere Betrachtung ausgeschlossen. Die erste Methode ist dabei einfach zu implementieren. Variante zwei ist zwar etwas komplexer und benötigt mehr Rechenleistung, verspricht

Abbildung 7: Konturenmethode mit Mittelpunkt

aber eine höhere Treffergenauigkeit.

8.2.2 Formen-Erkennung

Formen-Erkennung stellt neben der Hellen- bzw. Dunklen-Fleck-Erkennung die zweite Art von Algorithmen dar, um Unregelmäßigkeiten zu erkennen. Dabei gibt es für jede Form unterschiedliche Herangehensweisen, weshalb diese hier gesondert betrachtet werden. Da alle von derselben Operation ausgehen, dem Canny-Operator (nach [Canny, 1986], S.679-698), wird dieser im Folgenden kurz erläutert. Das Ziel dieses Algorithmus ist es, Kanten innerhalb eines Bildes zu finden. Diese sind vorhanden, wenn auf kurzem Raum ein abrupter Farb- bzw. Helligkeitswechsel stattfindet. Zunächst wird ein Gauß-Filter auf das Bild angewandt, der dazu dient Rauschen zu minimieren. Daraufhin berechnet der Algorithmus die partiellen Ableitungen mit Hilfe zweier Faltungsmatrizen einzelner Pixel in x- und y-Richtung, wodurch horizontale und vertikale Kanten betont werden. Daraus entstehen zwei Bilder, wobei eines die Kanten in y-Richtung und das andere die in x-Richtung enthält. Mit Hilfe dieser partiellen Ableitungen g_x und g_y lässt sich die Richtung θ des Gradienten einer Kante durch Gl. 11 ausdrücken.

$$\theta = arctan\left(\frac{g_y}{g_x}\right) \tag{11}$$

Da die Richtung benachbarter Pixel aber nur ein Vielfaches von 45° betragen kann wird der Winkel auf einen dieser Werte gerundet. Es wird anschließend die absolute Kantenstärke G mittels der euklidischen Norm berechnet. Um Rechenzeit zu minimieren verwendet OpenCV aber eine Approximation:

$$G = |g_x(x,y)| + |g_y(x,y)| \tag{12}$$

Um im entstehenden Binärbild eine Linie mit der Dicke von einem Pixel zu erhalten, wird die sog. Non-Maximum-Suppresion angewandt, die die benachbarten Pixel entlang des Gradienten vergleicht und nur den größten der Werte herausnimmt. Im letzten Schritt wird die sog. Hystere angewandt, die wieder eine Threshold-Operation mit zwei Schwellwerten darstellt. Die empfohlene Einstellung hierfür ist lt. Canny ein Verhältnis von 2:1 bis maximal 3:1 (siehe [Canny, 1986]) von oberem zu unterem Schwellwert. Liegt ein Pixel über beiden Schwellwerten, so wird er immer als Konturpixel akzeptiert. Dazwischen nur, wenn er mit einem Pixel verbunden ist, der bereits akzeptiert wurde. Alles was darunter ist wird zurückgewiesen.
Ein so entstandenes Binärbild mit Schwellwerten von 700 bzw. 1400 wird in Abbildung 8 dargestellt. Es ist anzumerken, dass für die folgenden Algorithmen stets unterschiedliche Schwellwerte für den Canny-Operator genommen wurden, da die Bilder wegen ihres hohen Detailgrades bei niedrigen Schwellwerten sehr viele Kanten enthalten. Dies führt in den meisten Fällen aber zu Falschmeldungen. Wie die Schwellwerte ermittelt wurden, soll in der Implementierung gezeigt werden.

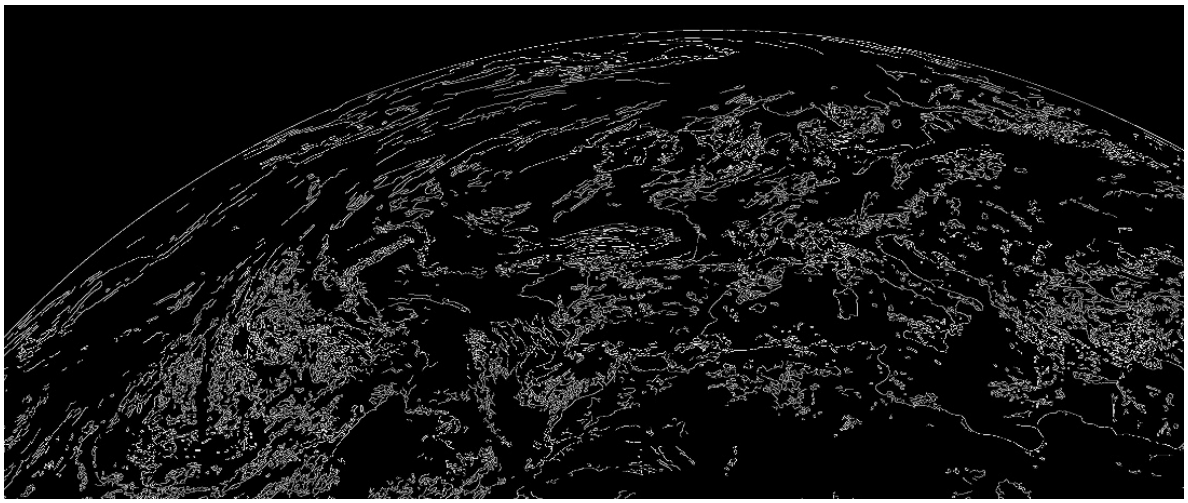

Abbildung 8: Canny-Algorithmus auf Satellitenbild angewandt

- **Vier-, Fünf- und Sechseckerkennung**

  Dieser Algorithmus dient der Erkennung elementarer geometrischer Formen in einem Bild und ist der Einzige, der keine Eigenentwicklung darstellt (Siehe [Nash, 2012]). Die Formen charakterisieren sich einerseits durch ihre Anzahl an Kanten und andererseits über den Betrag jeder ihrer Innenwinkel (nach [Nash, 2012]):

– Viereck:
  Betrag jedes Innenwinkels zwischen 85° und 97°
– Fünfeck:
  Betrag jedes Innenwinkels zwischen 105° und 110°
– Sechseck:
  Betrag jedes Innenwinkels zwischen 116° und 123°

Um nun die Form erkennen zu können, wird zunächst, wie oben genannt, der Canny-Operator angewandt. Innerhalb des Binärbildes wird daraufhin nach geschlossenen polygonalen Konturen gesucht[14] und für jede dieser Konturen alle Innenwinkel berechnet. Ist die Anzahl der Ecken gleich den jeweiligen für ein Mehreck und sind dazu noch alle Innenwinkel im oben angegebenen Bereich, wird die Kontur gespeichert und später markiert.
Ein Problem für die spätere Implementierung stellt die Winkelmessung dar: Je kleiner eine Form wird, desto stärker kann die Winkelmessung verfälscht werden, bis es schließlich zu so großen Abweichungen kommt, dass sie gar nicht mehr erkannt werden kann. Dies ist auch der Grund, warum keine Formen mit mehr als sechs Ecken geprüft werden: Die Fehler bei Winkelmessungen lassen die Form schnell als eine andere erscheinen, ebenso wie die Tatsache, dass der Betrag der Winkel nahezu gleich groß ist. Wie groß eine Form sein muss, um sie sicher erkennen zu können, wird eine der Aufgaben für die Evaluation sein.

- **Linienerkennung**

  Die Linienerkennung basiert auf dem 1962 von Paul V.C. Hough entwickelten Hough-Algorithmus (Siehe [Dawson-Howe, 2014], S. 109). Dieser geht ebenfalls vom Canny-Algorithmus aus, arbeitet demnach auch mit einem Binärbild. Der Grundgedanke ist dabei, dass eine Gerade in Polarkoordinaten dargestellt wird (Gleichung 13).

$$y = \left(-\frac{\cos\theta}{\sin\theta} x + \frac{r}{\sin\theta} \right) \tag{13}$$

  Und so für jede Linie durch die Punkte x,y:

$$r_\theta = x_0 * \cos\theta + y_0 * \sin\theta \tag{14}$$

[14] Die Innenwinkelsumme darf also maximal $(n-2) * 180°$ betragen, wobei n die Anzahl der Kanten ist

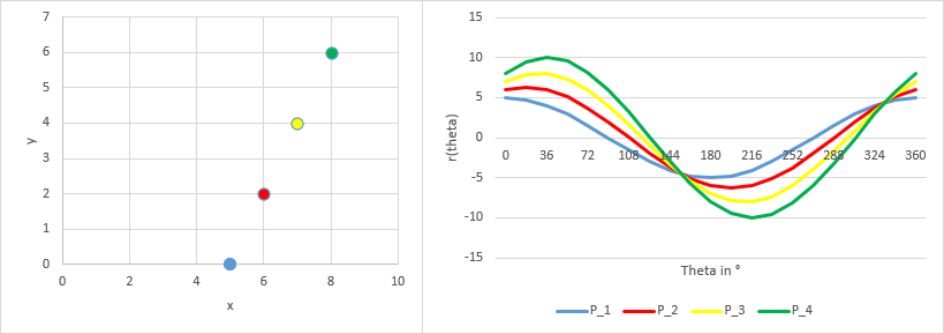

Abbildung 9: Die Hough Transformation von vier Punkten vom (x,y)-Raum in den (θ,r_θ)-Raum. Die Schnittpunkte der Kurven stellen die Geraden dar, die durch alle vier Punkte geht. Es gibt deshalb zwei Schnittpunkte, weil eine volle 360° Drehung angenommen wurde (Und Geraden, die im 180° Winkel liegen sind effektiv die Gleichen, [Dawson-Howe, 2014])

Da es unendlich viele Geraden durch einen Punkt gibt und es rechnerisch nicht effizient ist für jedes Punktepaar zu prüfen, wie viele Punkte auf der Gerade liegen, werden x- und y-Werte eines Punktes nicht mehr als Variable, sondern als Parameter behandelt. Stattdessen wird θ als variabel angesehen, ebenso wie das daraus folgende r_θ. So kann für jeden Punkt auf einer Kontur r_θ berechnet (Abbildung 9) und an der Stelle (θ, r_θ) einer Matrix, die der Größe des Bildes entspricht, der Wert um eins imkrementiert werden. Eine Gerade ist dann gegeben, wenn an einer Stelle der Matrix die Anzahl der Inkrementierungen über einem festgelegten Schwellwert liegt.[15]
Nachdem erfolgreich Linien gefunden wurden, werden diese gespeichert und durch eine Subroutine im Zielbild markiert.

- **Kreis-Detektion**

  Dieser Algorithmus dient der Identifizierung einer Kontur als Kreis. Dazu wird eine Variante des bereits erklärten Hough-Algorithmus angewandt, diesmal jedoch für Kreise. Demnach lässt sich ein Kreis C durch seinen Mittelpunkt in kartesischen Koordinaten sowie den Radius beschreiben:

$$C : (x_{zentrum}, y_{zentrum}, r) \tag{15}$$

  Ansonsten gilt wie zuvor, dass ein Kreis gegeben ist, wenn eine bestimmte Anzahl an Punkten auf dessen Bogen liegt. Eine Stärke der Methode ist, dass

[15] Die Einstellung der Schwellwerte für Canny und Hough-Transformation werden im Kapitel 9 behandelt.

der Kreis nicht komplett im Bild enthalten sein muss, es reicht bereits der Teil eines Kreisbogens.
Auch hier sind die Schwellwerte für die Canny- und Hough-Algorithmen zu finden. Ebenso ist es für die Implementierung eine Aufgabe, Kreise mit unrealistischer Größe herauszufiltern. Dazu gehören etwa Kreise mit $r \leq 1Px$ oder $r > r_{Erde}$.

8.3 Algorithmen zur Filterung von Sonnenreflexionen

Damit die in Kapitel 8.2.1 vorgestellten Algorithmen eine möglichst geringe Zahl an Falschmeldungen herausgeben, müssen Regelmäßigkeiten ausgeblendet werden (vgl. Kapitel 5.1). Eine Hauptquelle für Falschmeldungen stellen Sonnenreflexionen an der Erdoberfläche dar, die aufgrund der diffusen Brechung von Licht an dieser und den Wolken schwierig zu berechnen sind. Es ist eine andere - möglichst effiziente - Lösung zu finden und implementieren.
Hierzu wird ein Bereich innerhalb einer temporären Schwarz-Weiss-Matrix konstruiert, in dem für Tag im Jahr, Uhrzeit sowie pro Kanal die häufigsten Sonnenreflexionen finden lassen. Dieser Bereich ist möglichst klein zu halten, um dennoch Unregelmäßigkeiten jederzeit erkennen zu können. So wird er in den Morgen- und Abendstunden am Rand der Erdscheibe zu finden sein und tagsüber zentral, aber stets mit der Uhrzeit wandernd.
Auch wenn Heller- und Dunkler-Fleck-Algorithmus sich ähneln, so benötigen sie trotzdem unterschiedliche Routinen oder zumindest andere Ausgangs- bzw. Schwellwerte, um Reflexionen zu erkennen. Um die Schwellwerte zur Laufzeit anpassen zu können, benötigt das Programm ein User Interface, welches im folgenden Kapitel erläutert wird.

8.4 GUI und Worker Thread Prinzip

Um sämtliche Algorithmen aus Abschnitt 8.2 und 8.3 aufrufen zu können, wird eine Programmstruktur benötigt, die eine effiziente Bearbeitung der Tasks erlaubt. Hierzu wird das GUI- und Worker-Thread-Prinzip angewandt. Threads sind in Betriebsystemen eine der Grundideen der Parallelverarbeitung ([Mandl, 2014], S. 83). Deren übergeordnete Recheneinheit stellt der Prozess dar. Alle Threads teilen sich den gemeinsamen Adressraum eines Prozesses und greifen damit auf dieselben globalen Variablen zu, während Threads untereinander nur über definierte Interfaces kommunizieren. Der

Worker Thread stellt dabei einen Thread dar, der die für das Programm notwendigen Rechenoperationen durchführt. Als Beispiel dafür sei das Öffnen und Verarbeiten der Bilder genannt. Die ermittelten Daten werden über ein Interface an den GUI-Thread weitergegeben, der die Daten adäquat präsentiert und so z.B. das gerade verarbeitete Bild anzeigt. Um eine klare Unterteilung zwischen beiden Objekten zu schaffen, werden Datenstrukturen so konstruiert, dass sie nur von einer der beiden Parteien interpretiert werden können. Diese sind aber nicht zu verwechseln mit den globalen Variablen, auf die beide zugreifen können.

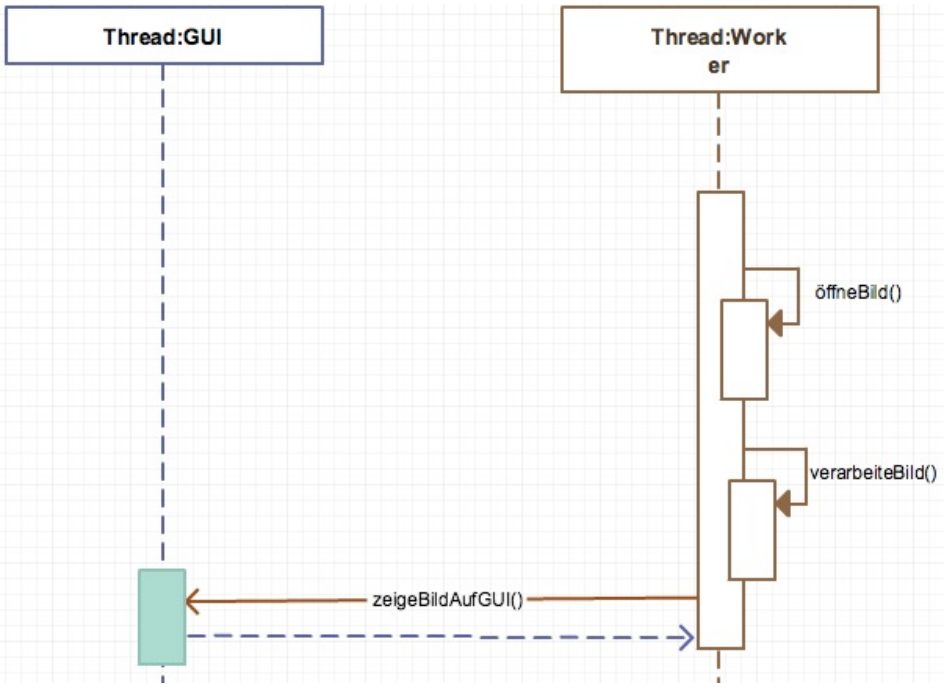

Abbildung 10: Skizziertes UML-Sequenzdiagramm zum Verhalten von GUI und Worker Thread beim Verarbeiten von Bildern

Abbildung 10 zeigt, wie die interne Kommunikation bei diesem Konzept gestaltet ist. Einen weiteren Vorteil bei diesem Vorgehen stellt die Modularität dar, mit der es dem Entwickler einfach gemacht wird dem bestehenden System schnell neue Worker-Threads hinzuzufügen. Jedoch besitzt dieses Konzept auch Grenzen. Dazu gehört, dass bei aufwändigen Rechenoperationen der GUI-Thread nicht schnell genug auf Anfragen reagieren kann und Kommandos erst mit erheblicher zeitlicher Verzögerung abgearbeitet werden. Ist darüber hinaus alles innerhalb eines Threads, müssten keinerlei Kommunikationsschnittstellen implementiert werden, was zu einer geringeren Anzahl an Lines-of-Code (LOC) führt.

Ebenso können mehrere Threads auf dieselbe Variablen zugreifen und ungewollte Zustände provozieren[16]. Hierzu wird das Mutex-Konzept implementiert, welches um die Stelle der Variablenänderung einen sog. kritischen Abschnitt definiert. Betritt ein Thread diesen Abschnitt wird er für die Anderen gesperrt ($mutex.lock()$) und beim Verlassen wieder geöffnet ($mutex.unlock()$). Dadurch wird gewährleistet, dass sämtliche Änderungen einer Variable angewandt werden ohne, dass eine davon verloren geht.

9 Implementierung

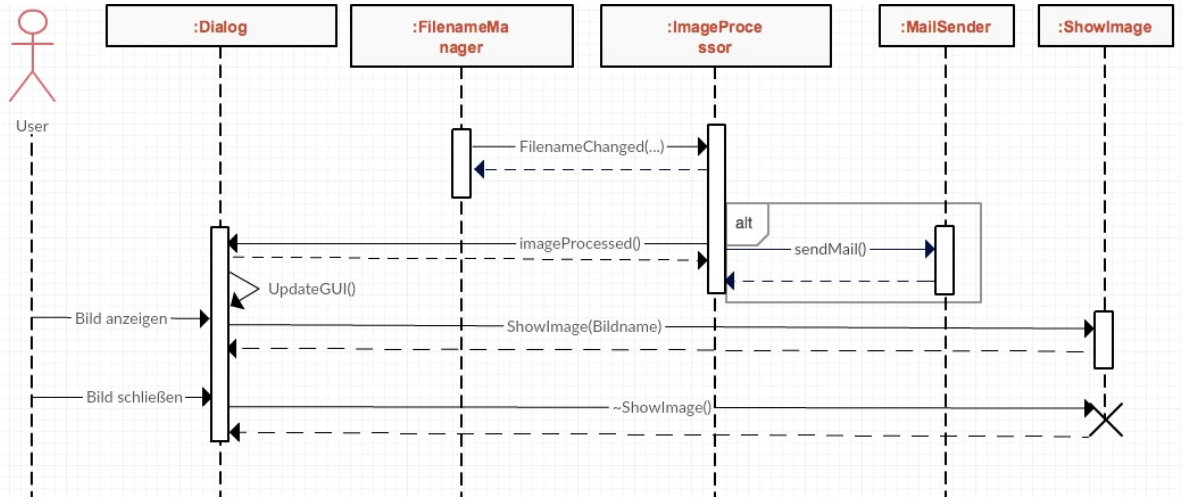

Abbildung 11: Interner Nachrichtenverlauf der Threads für den Fall, dass der User das aktuelle Bild anzeigen lässt

Nachdem die Struktur des Programms und der Algorithmen im Konzept erarbeitet wurde, ist es an dieser Stelle die Aufgabe des Entwicklers diese mit der Struktur von OpenCV und C++ zu vereinen. Der Fokus dieses Kapitels liegt auf den Problemen, die während der Implementierung auftraten und wie sie gelöst wurden sowie auf der Parametereinstellung der Algorithmen, wie z.B. Schwellwerte.
Als Richtlinie für die Einstellung der Parameter gilt dabei, dass alle tatsächlichen Unregelmäßigkeiten erkannt werden. Somit ist stets darauf zu achten, dass Schwellwerte und maskierte Bereiche minimal einzustellen sind.
Dafür wird zunächst auf die Implementierung der Algorithmen eingegangen bzw. deren Methoden zum Filtern von Regelmäßigkeiten. Anschließend werden das User-Interface

[16]Es kann beispielsweise vorkommen, dass zwei Threads gleichzeitig dieselben Variablen beschreiben und sie anschließend wieder abfragen. Dabei wird eine der beiden Änderungen zwangsweise verloren gehen. Dies kann bei gleicher Ausgangssituation zu einem unterschiedlichen Ergebnis führen. Diese Art von Zufall ist aber unerwünscht.

und die einzelnen Threads erläutert, die einen reibungslosen Ablauf aller Funktionen bieten sollen. In dieser Sektion wird auch auf die interne Kommunikation der Threads eingegangen, welche in Abbildung 11 dargestellt ist. Um die grafische Benutzeroberfläche und das Thread Konzept zu realisieren wird für diese Zwecke die C++ Bibliothek QT 5.2.0 in der Community-Version verwendet[17].

9.1 Anwendung der Algorithmen zur Erkennung von Unregelmäßigkeiten

9.1.1 Formen-Algorithmen

Der erste implementierte Formendetektions-Algorithmus ist der zur Erkennung von Vier- bzw. Fünf- und Sechsecken. Die Grundlage hierfür stellt ein bereits existierender Algorithmus dar, der aus der OpenCV-Tutorial Sektion entstammt. Ursprünglich konnte dieser Algorithmus nur Vierecke erkennen, jedoch wurde er von der OpenCV-Community so erweitert, dass er zustäzlich noch Fünf- und Sechsecke erkennen kann (Siehe [Nash, 2012]).
Da ein Satellitenbild aufgrund der hohen Auflösung und seines dementsprechend hohen Detailgrades viele Möglichkeiten zur Erkennung von Formen bietet, sind die Schwellwerte zur Konturfindung festzulegen. Dabei werden diese so lange erhöht, bis innerhalb der Testbilder keine Falschmeldungen mehr vorkommen. Dadurch wird gewährleistet, dass die Methode möglichst alle Formen erkennt und gleichzeitig nur wenige falsche Treffer meldet. Abbildung 8 zeigt bereits, wie ein Konturbild aussieht, welches mit dem Canny-Algorithmus und einem Schwellwert von 700 zu 1400 erstellt wurde. Es ist zu erkennen, dass aufgrund einiger Wolkenformationen noch viele Konturen erkennbar sind, die potenzielle Falschtreffer hervorrufen können. Damit sich auch diese Bereiche auflösen, ist der Schwellwert weiter anzuheben. Dadurch ergibt sich die Gefahr, dass Konturen der zu findenden Formen nur noch als Teil erkennbar sind und somit nicht erkannt werden. Aufschluss darüber, wie gut die Detektion damit noch funktioniert, ist Teil der Evaluation.
Anschließend ist die Liniendetektion einzubauen. Dazu wird zunächst der Standard-Hough Algorithmus aus der OpenCV Bibliothek implementiert. Dieser besitzt jedoch die Eigenschaft, dass er für eine Linie nicht den Start und Endpunkt in kartesischen Koordinaten speichert, sondern die Parameter r_θ und θ. So eingestellt werden nur Tangen-

[17]Siehe: `http://www.qt.io/download/`, aufgerufen am 25.07.15

ten an der Erdscheibe und keine Linien innerhalb dieser entdeckt. Deshalb wird der sog. probabilistische Hough-Algorithmus angewandt. Er ist aufgrund einiger Vereinfachungen bei der Linienberechnung nicht nur effizienter als der Standard-Hough-Algorithmus, sondern gibt darüber hinaus auch die Extrema (x_0, y_0, x_1, y_1) der detektierten Linien als `Vec4i` (4×1 Int-Vektor) aus. So kann für Start- und Endpunkt der Linie getestet werden, ob die Linie aus der Erdscheibe herausreicht. Ist dies der Fall, so wird die Linie verworfen, wenn nicht, wird sie akzeptiert und ins Zielbild eingezeichnet. Abbildung 12 zeigt Linien, die mit dem Algorithmus erkannt wurden.

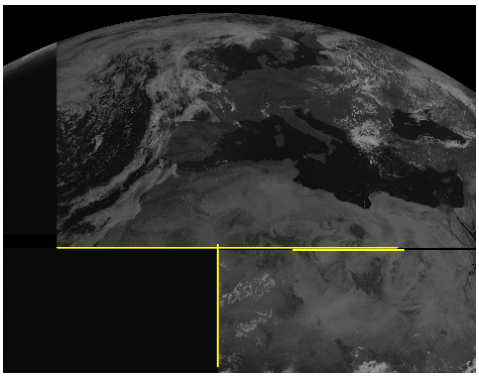

Abbildung 12: Fund einer Linie am Rand eines Bildes von Kanal 12

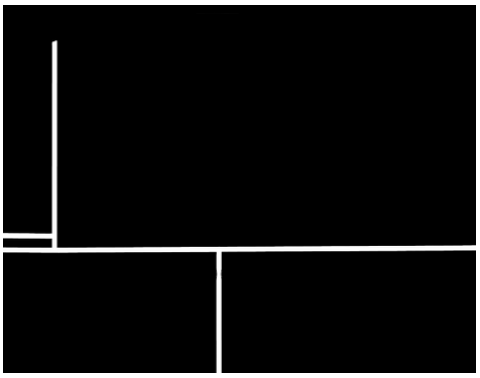

Abbildung 13: Maske für Ränder von Kanal 12 in diesem Bereich (weiß = maskierter Bereich)

Der Fund dieser Linien stellt ein kritisches Ereignis dar, denn ihr Ursprung resultiert daraus, dass in den Bildern auf Kanal 12 nur die Kontinente aus zwei großen Bildteilen zusammengesetzt sind. An den Kanten dieser Stücke wird deshalb immer eine Linie vorliegen. Da die Bildteile stets an derselben Stelle aufhören, wird der Konturbereich für Kanal 12 maskiert. Die so entstandene Maske ist in Abbildung 13 für diesen Ausschnitt erkennbar.

Der Kreisdetektor ist der letzte zu implementierende Formenalgorithmus. Hier wird das Standard-Hough-Verfahren für Kreise angewandt und keine erweiterte Version. Das erste Ergebnis liefert für die meisten Bilder die Erdscheibe als Kreis. Da dies ebenfalls eine Regelmäßigkeit darstellt, ist dieser und andere ungewollte Kreise gezielt zu verwerfen:

$$\textbf{if}\,(r_{Kreis} \geq r_{Erde} \;\; || \;\; r_{Kreis} < 5Px \;\;)\;\; \mathrm{skipCurrentElement}\,()$$

Im obigen Codeausschnitt ist darüber hinaus zu erkennen, dass Kreise mit einem Radius kleiner 5 Pixel verworfen werden, da sonst zu viele Falschmeldungen gefunden werden. Damit ist auch bereits eine untere Grenze für den Kreisalgorithmus festgelegt: Ist der Radius eines Kreises kleiner als 5 Pixel, wird er nicht als solcher markiert.

9.1.2 Heller- und Dunkler-Fleck-Algorithmus

Zunächst wird der Helle-Fleck-Algorithmus implementiert. Dazu werden zuerst die in Kapitel 8.2.1 vorgestellten Methoden zur Markierung der hellen Flecken ausgewählt. Bei einem Vergleich der Gewichteten gegenüber der Ungewichteten zeigt sich, dass beide dieselbe Treffergenauigkeit aufweisen. Dies ist bei genauerer Untersuchung nicht überraschend: Die Methoden arbeiten bis zu dem Punkt, an dem geprüft wird, ob ein heller Fleck vorliegt unterschiedlich, aber arbeiten danach gleich weiter. Die kritische Threshold Operation, die für eine hohe Treffergenauigkeit verantwortlich ist, kommt erst während der Prüfung bzw. danach zur Ausführung. Deshalb sind beide Methoden im Grunde gleich, was die Wahl auf die ungewichtete Methode fallen lässt, da sie etwas weniger Rechenaufwand benötigt als die Gewichtete.
Für den dunklen Fleck kann dieselbe Methode angewandt werden. Dazu wird das Bild vorher mit einem Look-Up-Table (LUT) pixelweise invertiert. So erscheinen sehr dunkle Pixel als helle Pixel, wodurch der Helle-Fleck-Algorithmus wieder angewandt werden kann. Es sind zuletzt einige Schwellwerte für diesen Fall anzupassen, so dass die Anzahl der Falschmeldungen gering bleibt, aber immer noch alle dunklen Flecken gefunden werden können. Dabei besitzt der Helle-Fleck-Algorithmus einen Standardschwellwert für weiße Flecken von 250, der Dunkle-Fleck-Algorithmus minimal höher bei 253.
Ebenso ergibt sich für den Dunklen-Fleck-Algorithmus ein Problem: Auf Bildern der VIS-Kanäle reflektieren viele Teile von Land und Wasser nur sehr wenig Licht dieser Wellenlänge, so dass der Bereich auf dem Bild schwarz erscheint. Da diese Bereiche nicht vorhersagbar sind und der Algorithmus damit in diesen Kanälen ausschließlich Falschmeldungen liefert, besteht lediglich die Möglichkeit diese Kanäle wegzulassen. So operiert der Algorithmus nur im Infrarot-Bereich.
Darüber hinaus liefern beide Algorithmen eine große Zahl an Falschmeldungen für Sonnenreflexionen, vor allem in den Morgen- und Abendstunden. Aus diesem Grund soll im nächsten Kapitel erläutert werden, wie damit umgegangen wurde.

9.2 Erkennung von Regelmäßigkeiten

Damit Heller- und Dunker-Fleck Algorithmus korrekt funktionieren und gleichzeitig die Fehlerrate minimal bleibt, muss ein Mechanismus eingebaut werden, der für beide Algorithmen die Einflüsse der Sonne erkennt und sie herausfiltert. Ein Problem bei der Suche nach einer Lösung stellt die OpenCV-Bibliothek dar: Sie ist hauptsächlich für

zweidimensionale Bilder ausgelegt, obwohl der Ursprung des Problems jedoch aus dem dreidimensionalen Raum kommt und letzten Endes auf ein zweidimensionales Bild projiziert wird. Aus diesem Grund wird es einen Genauigkeitskompromiss geben müssen, um mit OpenCV weiterarbeiten zu können. Ob die Präzision dennoch ausreicht wird sich erst im Verlauf der Arbeit zeigen.
Es gibt Ansätze, die trotz der eben genannten Einschränkungen Erfolg versprechen. Es ist bekannt, dass die Sonne im Laufe eines Jahres von 23,45°, dem nördlichen Wendekreis (am 21.6.), auf -23,45° am südlichen Wendekreis (am 21.12.) wandert. Daraus kann man für die der Deklination δ an einem Tag im Jahr ($dayOfYear$, 01.01. = 1) Gleichung 16 aufstellen.

$$\delta = -23,45 * \cos\left(\frac{2 * \pi}{365} * (dayOfYear + 10)\right) \tag{16}$$

Der Verlauf einer Sonnenrelexion lässt sich empirisch ermitteln und wird in Abbildung 14 für den 23.05.2015 sowie 09. und 10.06.2015 dargestellt. Der Vergleich der Position der Punkte derselben Uhrzeit macht sichtbar, dass die Deklination innerhalb dieses Zeitintervalls um einige Grad gestiegen ist - so liegen die Punkte des 23.05. deutlich weiter von denen des 10.06. entfernt, auch wenn bei der Beobachtung dieser Reflexionen eine Streuung durch Messungenauigkeit vorhanden ist. Von weitaus größerer Relevanz ist die Erkenntnis, dass der Verlauf der Reflexion tagsüber durch einen Ellipsenbogen beschrieben werden kann, auch wenn zum Rand der Erde hin, in den Morgen- und Abendstunden, eine starke Verfälschung erkennbar ist. Zunächst werden aber nur Reflexionen, die tagsüber entstanden sind betrachtet.

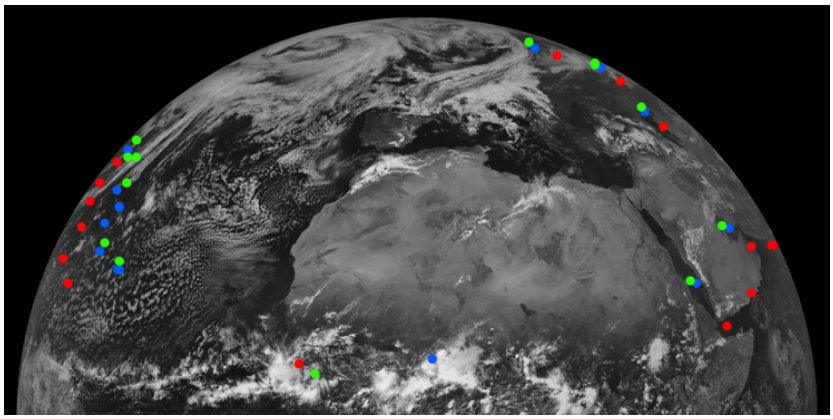

Abbildung 14: Sonnenreflexionen an drei unterschiedlichen Tagen: 23.05. (rot), 09.06. (blau) und 10.06. (grün)

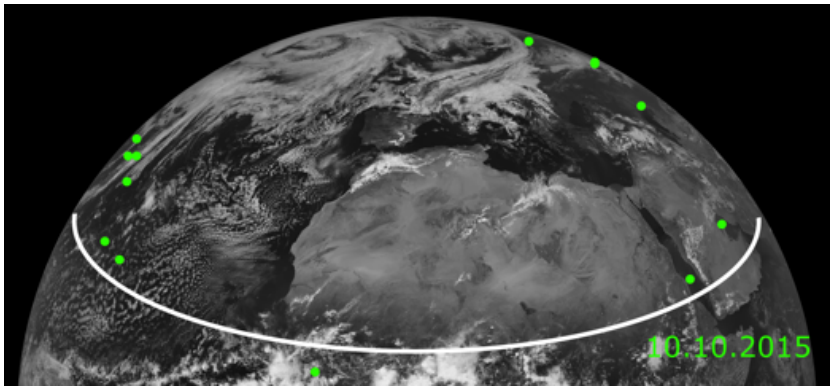

Abbildung 15: Annäherung des Pfades der Sonnenreflexion am Tag durch einen Ellipsenbogen für den 10.06.

OpenCV besitzt die Möglichkeit Ellipsen(-bögen) zu zeichnen und benötigt dazu folgende Parameter:

- Den Mittelpunkt der Ellipse in kartesischen Koordinaten
- Die Größe der beiden Halbachsen
- Den Start- und Endwinkel des Ellipsenbogens (0° bis 360° bei einer vollen Ellipse), ausgehend vom Punkt auf der Ellipse mit dem größten x-Wert.

Nachdem für die drei Tage jeweils eine passende Ellipse gefunden (Siehe Abb.15) werden konnte, sollte sich ebenso für den Rest der Tage im Jahr eine Vorhersage für den Bogenverlauf ermitteln lassen, da dafür keine Aufnahmen vorliegen. Es ist dazu hilfreich die Extremsituationen zu betrachten, wenn die Deklination gleich 0° bzw. ± 23,45° ist.

So ergibt sich für die Tagundnachtgleiche im Frühling bzw. Herbst eine Position genau im Mittelpunkt des Bildes. Dies ist aber nur möglich, da dank der Vorverarbeitung von Meteosat (Siehe [Müller, 2007]) bereits gegeben ist, dass die Erde zentriert im Bild liegt. Darüber hinaus entspricht die große Halbachse der Hälfte des Erddurchmessers, die Kleine wird zu 0. Da der Verlauf der Reflexion nur ein halber Ellipsenbogen ist, geht die Ellipse von 0° bis 180°.
Für die beiden Wendekreise gilt eine ähnliche Überlegung. Die Position ändert sich lediglich in positive bzw. negative x-Richtung um einen Wert Δx, der von der Deklination bzw. des Tages im Jahr abhängt, entlang der y-Achse hat sie einen konstanten Wert. Die große Halbachse wird kleiner, je weiter man sich vom Äquator entfernt, während die Kleine sich vergrößert. Der Bogen der Ellipse bleibt für die nördliche Hemisphäre

weiterhin bei 0° bis 180° ändert sich bei der Überschreitung des Äquators nach Süden aber auf 180° bis 360°, da nun die obere Hälfte des Bogens dargestellt werden muss.

Für die Tage, die zwischen den vier Extrempunkten liegen wird ein linearer Anstieg bzw. Abfall der jeweiligen Werte angenommen. Hier sind die bereits vorliegenden Bilder hilfreich, denn sie wurden zwischen Frühlings-Tagundnachtgleiche sowie Sommersonnenwende aufgenommen, liegen damit genau in diesem Bereich und dienen somit als Testfall. Es ergeben sich somit für die unterschiedlichen Tage folgende Gleichungen für die x-Position der Ellipse in Abhängigkeit des Tages im Jahr (day_{year}). Innerhalb des Intervalls von 21.03. (81. Tag) bis 21.06. (172. Tag):

day −=80;

$x(day_{year}) = Size_{Image}/2 - day_{year} * \left(\frac{Size_{Image}/2 - Min_{x-Position}}{172-81-1}\right)$

Sowie zwischen 22.06. (173. Tag) und 23.09. (266. Tag):

day −=172;

$x(day_{year}) = Min_{x-Position} + day_{year} * \left(\frac{Size_{Image}/2 - Min_{x-Position}}{266-173-1}\right)$

Zuletzt zwischen 24.09. (267. Tag) und 20.03. (80. Tag), wobei der 356. Tag den Wendepunkt des Sonnenstandes im Winter markiert:

```
if(day > 356){
    day = 356 - (day - 356);
}else if(day > 0 && day < 81){
    day += 365;
    day = 356 - (day - 356);
}
```

day −=266;

$x(day = Size_{Image}/2 + day_{year} * \left(\frac{Size_{Image}/2 - Min_{x-Position}}{356-267-1}\right)$

Analog zu diesem Schema werden die Gleichungen für die Größe der Halbachsen formuliert und implementiert. Um nun mit Hilfe dieser Gleichungen ein geeignetes Ausschlusskriterium für eine Sonnenreflexion zu finden, ist zunächst zu berücksichtigen, dass der Bereich, der für Sonnenreflexionen infrage kommt, möglichst klein gehalten werden muss, um eventuelle Unregelmäßigkeiten auch innerhalb der Bahn der Sonnenreflexion erkennen zu können. Es ist an dieser Stelle möglich über den ganzen Tag den Bereich des Ellipsenbogens einer gewissen Dicke auszugrenzen.
Da die Reflexionen derselben Uhrzeit aber immer im selben Bereich der Erde erscheinen,

kann der Ellipsenbogen auf einen geringen Teil seiner selbst reduziert werden, sodass je nach aktueller Uhrzeit nur ein bestimmter Bereich der Ellipse angezeigt wird. Hierzu wird der Punkt in Polarkoordinaten ($d_{P(x,y)}$, α) transformiert und eine Korridorgröße definiert, die angibt, wie groß der Bereich vom Bogen für die Maskierung der Reflexion ist. Dessen obere und untere Grenze wandern im Laufe eines Tages von Osten nach Westen abhängig von der aktuellen Uhrzeit. Damit eine Sonnenreflexion erkannt wird, muss der Winkel α des Punktes gegenüber dem Ellipsenmittelpunkt innerhalb der Grenzen liegen, genauso wie die Distanz $d_{P(x,y)}$ innerhalb der Ellipsenkontur.
Zuletzt wird noch eine Dicke der Ellipse d definiert, die in die temporäre Masken-Matrix gezeichnet wird, so dass letzten Endes eine einfache Vergleichsoperation Aufschluss darüber gibt, ob der eben gefundene helle Fleck innerhalb des Bereichs liegt und somit eine Sonnenreflexion ist. So ergeben sich folgende Aussagen, die für eine Sonnenreflexion alle wahr sein müssen:

- $d - r < d_{P(x,y)} < r$
- $Grenze_{Unten}[^\circ] < \alpha < Grenze_{Oben}[^\circ]$
- Bild ist zwischen 2:00h und 22:00h aufgenommen

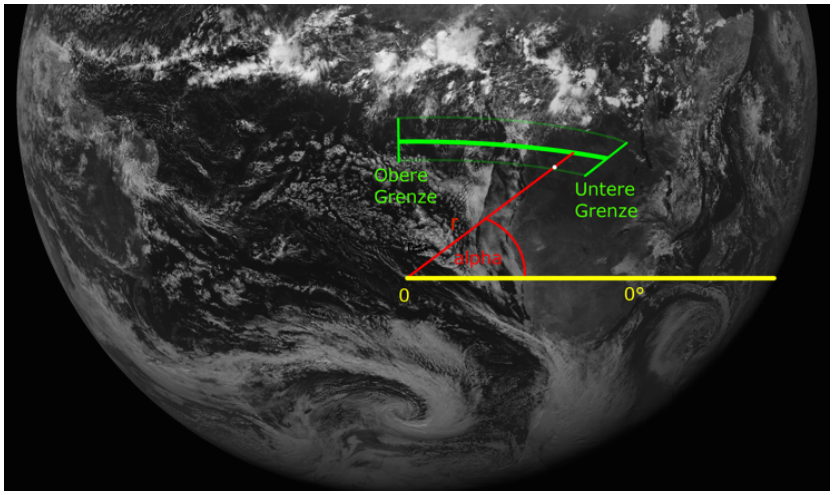

Abbildung 16: `isOnSunTrajectory()`-Test für eine fiktive Reflexion im Winter um 11:00 Uhr

Abbildung 16 zeigt die Funktionsweise der resultierenden `isOnSunTrajectory()`-Methode für eine fiktive Reflexion im Winter/Herbst bei 11:00 Uhr. Es ist zu erkennen, dass der Ellipsenbogen nur einen für diese Uhrzeit charakteristischen Ausschnitt anzeigt.

Darüber hinaus liegt der Bogen aufgrund der Deklination in der südlichen Hemisphäre und zeigt so nur Ausschnitte von 180° bis 360°.
Für Reflexionen morgens und abends kann ein ähnliches Verfahren angewandt werden, jedoch wird hier als Referenz die Erdkontur angenommen. Dazu wird zunächst mit einem `PointPolygonTest()` geprüft, ob der Punkt innerhalb der Erdkontur liegt und wenn ja, wie groß seine Distanz zu dieser ist. Ist sie mehr als die Dicke d vom Scheibenrand entfernt, so kann es sich um keine Reflexion handeln. Ebenso wird wieder ein Korridor festgelegt, in dem sich der Winkel α befinden muss, um als Sonnenreflexion eingestuft zu werden.

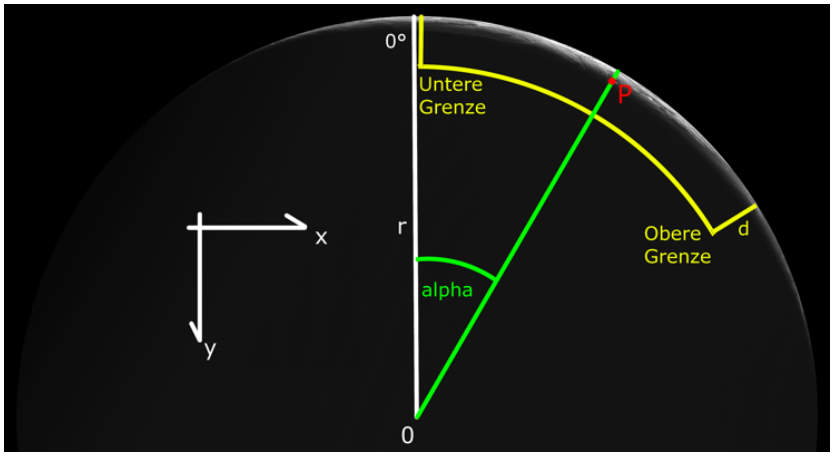

Abbildung 17: `isSunReflection()`-Test für einen Punkt P am Rand der Erdscheibe um 1:15 Uhr

Abbildung 17 zeigt die Parameter und die aus ihnen aufgespannte Fläche am Rand der Erdscheibe für einen Test einer Reflexion um 1:15h im Sommer. Es ist anzumerken, dass auch hier die Deklination auf die Lage von oberer und unterer Grenze addiert wird und sich somit eine Abhängigkeit von den Parametern

- Datum (`DayOfYear` und damit Deklination δ)
- Uhrzeit (Stunden und Minuten, keine Sekunden)
- sowie (x, y)- bzw. $(d_{P(x,y)}, \alpha)$-Position der Reflexion

ergibt.
Damit der Algorithmus nur Regelmäßigkeiten am Morgen und Abend filtert ist festgelegt, dass er von 19:00h bis 5:00h morgens aktiv ist. Eine Unterscheidung von Sommer- und Winterzeit ist nicht notwendig, da die UTC als Standard-Zeitskala angenommen

wird. Den Rest des Tages übernimmt die `isOnSunTrajectory()`-Methode.
Damit können helle Flecken, die durch eine Sonnenreflexion entstanden sind, gefiltert werden. Für den Dunklen-Fleck-Algorithmus gelten allerdings andere Voraussetzungen. Da dieser nur im Infrarot- bzw. WV-Bereich arbeitet, ergeben sich häufig über Landmassen (speziell Wüsten) dunkle Flecken. Ursache dafür ist der geringe Wassergehalt über trockenen Flächen und somit ein geringes Reflexions-Potential im WV-Bereich.
Hier liegt erneut eine Regelmäßigkeit vor, die es von Unregelmäßigkeiten zu unterscheiden gilt. Die Lösung stellt eine Maske dar, die nur Landflächen enthält und je nach aktueller Uhrzeit und Tag im Jahr (siehe oben) einen Teil davon zur Prüfung zur Verfügung stellt. Ebenso wie bei den Methoden zuvor gibt es dazu variable Parameter, die nachträglich noch anpassbar sind. Durch deren Variation kann die Größe des Bereichs beliebig verändert werden. Möglich ist dies durch die XML-Datei, die die entsprechenden Parameter enthält (Siehe Kapitel 9.3).
Jedoch führt die Anwendung der Land-Maske dazu, dass ein sehr großer Bereich der Kanal 5-8 Bilder maskiert wird, der aber nicht die in Kapitel 9.2 beschriebenen dunklen Bereiche besitzt. Somit wird für diese vier Kanäle die `isLandmass()`-Filtermethode weggelassen, was einen höheren Bereich für Unregelmäßigkeiten und somit eine bessere Trefferwahrscheinlichkeit bedeutet.

9.3 Aufbau der GUI

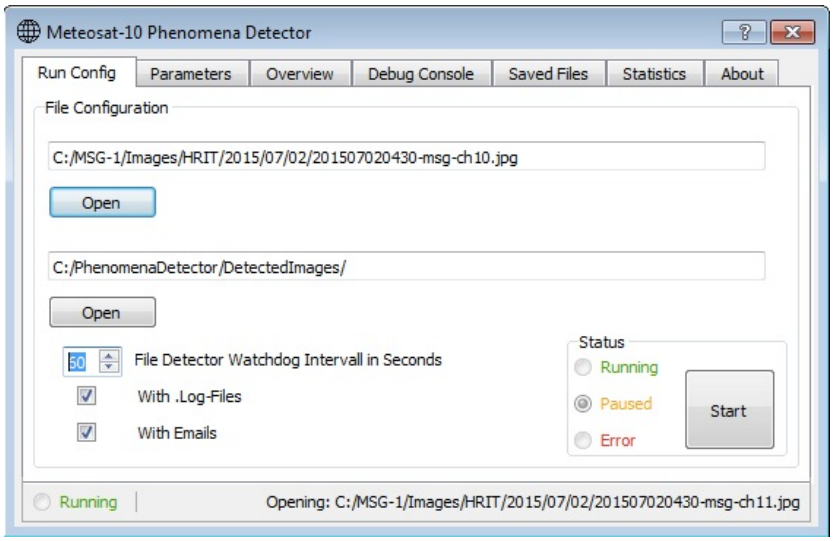

Abbildung 18: Konfigurations-Tab der GUI

Um dem Benutzer des Programms einen möglichst intuitiven Umgang mit der Suche

nach Unregelmäßigkeiten zu ermöglichen wird im Folgenden auf den Aufbau des Graphical User Interface (GUI) eingegangen. Dazu wird, wie schon erwähnt, die Bibliothek QT 5.2.0 verwendet. Sie bietet für C++ einen großen Umfang an Bausteinen für Benutzeroberflächen, darunter auch das Anzeigen von Bildern im OpenCV-Format `Mat`, wodurch es für diesen Einsatzfall geeignet ist. Aus Lizenzgründen muss das Programm darüber hinaus dynamisch kompiliert werden und innerhalb des Programms eine Referenz auf die QT Website vorhanden sein.
Das User Interface ist dabei die Schnittstelle zwischen Eingaben des Benutzers und den Worker Threads, wie aus Abbildung 11 zu entnehmen ist. So soll ein geordneter Ablauf beim Arbeiten mit dem Programm gewährleistet sein. Das Einstellen von Parametern zur Laufzeit läuft grundsätzlich so ab, dass der GUI-Thread die Eingabe entgegen nimmt und anschließend an den eigentlichen Empfänger weitergibt. Ebenso stellt der GUI-Thread den Main-Thread dar, der alle Worker Threads beinhaltet. Dadurch erben aber die Worker nicht vom Main-Thread, sondern der Main-Thread hat volle Kontrolle über untergeordnete Instanzen. So wird es überhaupt möglich, dass beispielsweise Bilder dem Benutzer angezeigt werden können, die eigentlich aus einem Sub-Thread kommen.
Das Programm wird in nur einem Fenster konzipiert, in dem alle Einstellungen und Daten sichtbar sind. Dabei ist die Oberfläche in sog. "Tab"-Seiten unterteilt wie man sie aus Browsern kennt. Im ersten Tab sollen dabei grundlegende Konfigurationsparameter eingestellt werden (Abbildung 18), damit das Programm gestartet werden kann. Dazu sind der Pfad zu einem Bildverzeichnis beliebigen Datums von Meteosat-10 sowie das Zielverzeichnis für gespeicherte Bilder und log-Files, in dem auch die Masken und eine Parameter-Datei auffindbar sind, notwendig. Ebenso kann hier optional das Erstellen von log-Files bzw. der Mailversand (de-)aktiviert werden. Zuletzt befindet sich auf dieser Seite noch der Startbutton, der erst aktiviert ist, wenn die beiden Pfadeingaben korrekt sind und sämtliche Konfigurationsdateien und Ordner gefunden wurden.
Darüber hinaus beinhaltet das User Interface zwei Konsolen. Eine, um Debug-Nachrichten zu lesen und eine, um Namen, Pfad und Datum einer gespeicherten Datei anzuzeigen. Nachrichten innerhalb der Konsolen fangen stets mit dem Zeitstempel an, an dem sie veröffentlicht wurden, um Ereignisse besser zuordnen zu können. In diesen beiden Tabs finden sich dazu noch Counter für die gesamte Anzahl der verarbeiteten sowie gespeicherten Bilder und ein zunächst roter Radio-Button, der bei einem Fund für eine Dauer von 24 Stunden grün wird und so eine Unregelmäßigkeit signalisieren soll, sich nach 24

Stunden aber wieder zurücksetzt. Zuletzt ist auf den beiden Seiten noch ein untergeordnetes User-Interface eingebaut, welches das aktuelle bzw. zuletzt gespeicherte Bild in einem neuen Anzeigefenster darstellt (ShowImage-Thread, siehe Kapitel 9.4).
Ebenso findet sich eine Seite mit Informationen über den aktuellen Zustand des Programms. Dazu gehört etwa die Laufzeit seit Programmstart und die Windows-spezifische Prozess ID. Darüber hinaus existiert eine zweite Informationsseite, die etwas weniger Parameter enthält, dafür aber auch das aktuell verarbeitete Bild anzeigt. Diese Seite soll im regulären Betrieb als Hauptanzeigeseite verwendet werden, da sie alle relevanten Daten auf einer Seite vereint.
Ergänzt wird das User Interface noch von einem Tab, in dem zentrale Parameter der Algorithmen zur Laufzeit geändert werden können. Dazu existiert eine XML-Datei, die diese Parameter speichert und so die Daten auch über einen Neustart des Programms hinaus speichern und wieder öffnen kann. Damit soll gewährleistet sein, dass stets eine Optimierung des Programms stattfindet. Ebenso können durch einen Abgleich mit einem XML-Schema, welches nach der Richtlinie des W3C[18] angefertigt wurde, auch andere XML-Dateien mit gleicher Struktur eingelesen werden. So können unterschiedliche Programmkonfigurationen geladen und gespeichert werden, wodurch eine höhere Flexibilität gegenüber Anpassungen ermöglicht wird.

9.4 Aufbau der Worker-Threads

Da das Programm mit dem Öffnen, Speichern und dem Verarbeiten von Bildern sowie der Anzeige aktueller Daten einen recht großen Umfang besitzt, ist es sinnvoll einzelne Aufgabengebiete zu unterteilen. Damit diese Aufgaben schnell erledigt werden können wird das in Kapitel 8.4 angekündigte GUI-Worker-Thread Prinzip angewandt. Zunächst werden der ImageProcessor-Thread und der FilenameManager-Thread implementiert. Ersterer ist dient dazu das Bild zu öffnen, transformiert es in einen für die Algorithmen interpretierbaren Zustand und ruft anschließend sämtliche Algorithmen auf. Bei der Transformation (siehe Kapitel 6) ist zu erwähnen, dass dabei sämtliche Bilder in ein Bild der Größe 3712×3712 übertragen werden, welches der Standardgröße der Bilder von Kanal 1-11 entspricht. Der FilenameManager überprüft in einem einstellbaren Intervall, ob ein neues Bilderset eingetroffen ist und öffnet die Bilder sukzessiv. Da es vorkommen kann, dass die Bilder eines Zeitpunkts etwa aufgrund von Wartungsarbeiten nicht übertragen werden, besitzt der Thread eine Timeout-Funktion, die nach

[18] Siehe `http://www.w3.org/XML/Schema`, aufgerufen am 28.07.2015

15-Minuten automatisch die Uhrzeit, zu der nach Bildern gesucht wird, inkrementiert. So wird sichergestellt, dass das Programm nicht zu lange auf Bilder einer Uhrzeit wartet und stattdessen dort weiterarbeitet, wo wieder Bilder empfangen wurden. Somit liegt die Maximalgrenze des Intervalls für die nach einem Bild gesucht wird bei

$$\lfloor \frac{15min}{13Bilder} \rfloor = 69 \frac{sec}{Bild}. \tag{17}$$

Ein Problem bei der Suche nach neuen Dateinamen stellt die in den Randbedingungen spezifizierte Namenkonvention (Tabelle 4, C030) dar. Da sich solche Konventionen ändern können, wird versucht Dateinamen-unabhängig zu arbeiten. Dazu wird mit einer weiteren externen Bibliothek *Boost*[19], die einen integrierten Filesystem-Manager mitbringt, sowie mittels der QFilesystemWatcher-Klasse von QT getestet, wie sich das Überwachen von Dateisystemen verhält. Beide Methoden funktionieren fehlerfrei bei wenigen Änderungen über eine Zeitspanne. Dieses Verhalten ändert sich jedoch, wenn viele Änderungen über einen kurzen Zeitraum stattfinden. Dies ist der Fall, wenn die Bilder eines neuen Sets fertig sind und bedeutet, dass innerhalb von einer Sekunde 13 Dateien mit Größen zwischen einigen hundert KB und mehreren MB ins Verzeichnis geschrieben werden. Beide Dateisysteme erkennen zwar, dass Dateien hinzugefügt wurden, aber nicht, wie viele und vor allem nicht welche Dateien[20]. Das führt zwangsweise dazu, dass anhand der festgelegten Namensrichtlinie geprüft werden muss, welche Dateien hinzugefügt wurden.
Somit verlieren die Dateisysteme ihre Wirkung, da für ein Prüfen der Dateien mit demselben Namensschema kein Dateimanager mehr benötigt wird. Im weiteren Verlauf der Arbeit wird deshalb darauf verzichtet und eine unabhängige Version geschaffen, die lediglich auf Dateinamen beruht.
Für zwei weitere Nebenoperationen wurden ebenfalls Threads geschaffen: Mailversand und Bildanzeige. Ersterer organisiert das Versenden der Mail mit dem von Jakub Piwowarczyk und David Johns entwickelten CSMTP-Mail Client mit SSL/TLS (siehe [Piwowarczyk und Johns, 2012]). Der zweite Thread wird dann instanziert, wenn der Benutzer ein Bild im Programm anzeigen lassen möchte. Daraufhin wird die angegebene Datei geöffnet und in einem separaten Fenster angezeigt. Beim Schließen des Fensters wird die aktuelle Instanz des Threads zerstört und der belegte Arbeitsspeicher freige-

[19] Siehe `http://www.boost.org/`, aufgerufen am 25.07.2015
[20] Siehe hierzu `http://doc.qt.io/qt-4.8/qfilesystemwatcher.html`, aufgerufen am 25.07.2015

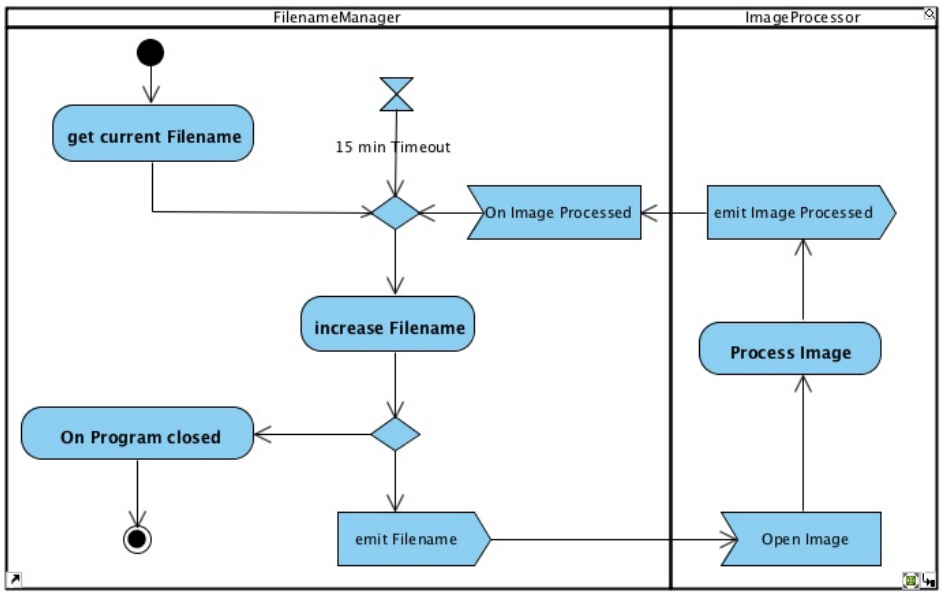

Abbildung 19: Aktivitätsdiagramm für FilenameManager und ImageProcessor

geben. Dies optimiert die Speichernutzung.

9.5 Routineablauf beim Eintreffen eines neuen Bildes

Dadurch, dass das Programm dauerhaft auf der Bodenstation problemlos betrieben werden soll, kommt dem Routineablauf beim Eintreffen eines neuen Bildes eine wichtige Rolle zu, welcher im folgenden anhand von Aktivitätsdiagrammen erläutert werden soll.

Zunächst muss das Bild geöffnet werden, was wiederum erfordert, dass dessen Name bekannt ist. Dazu arbeitet das Programm bei der Suche nach neuen Dateien mit einer internen Datenstruktur, die Datum, Uhrzeit und Kanal enthält und so abhängig vom Namen des Vorgängerbildes jeweils eine der Größen inkrementiert. Daraufhin wird der Dateiname aus den drei Parametern zusammengesetzt und an den ImageProcessor-Thread übergeben. Dieser versucht mit dem übergebenen Namen diese Datei zu öffnen. Schlägt die Operation fehl, gibt der FilenameManager zunächst keine weiteren Dateinamen aus, sodass auf das aktuelle Bild gewartet wird. Wie oben erwähnt müssen aber zeitweise ganze Bildersets übersprungen werden, da diese nicht empfangen werden konnten. Dazu existiert ein Timeout, sodass entweder nach 15-Minuten oder nach dem erfolgreichen Verarbeiten der Datei wieder ein neuer Dateiname ausgegeben wird. Abbildung 19 zeigt diesen Ablauf in einem Aktivitätsdiagramm.

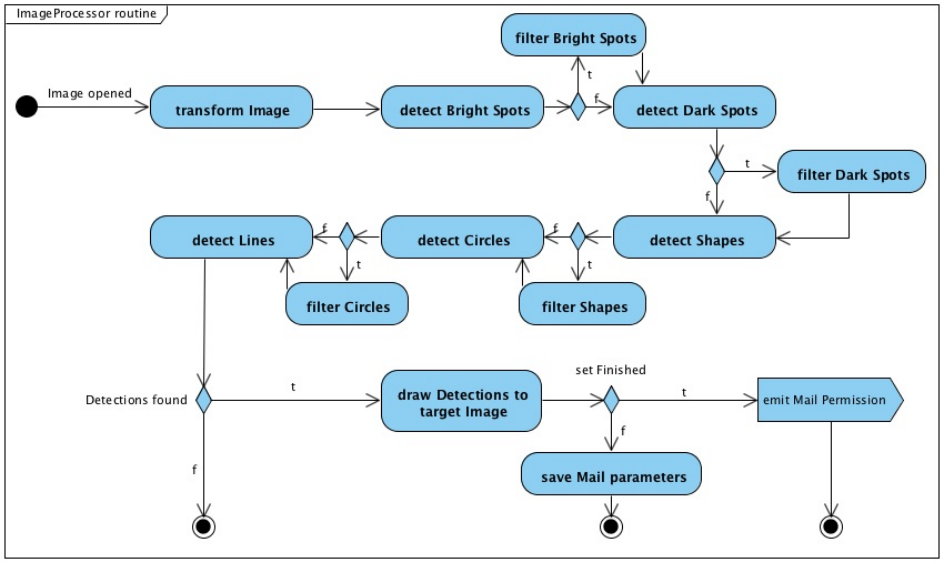

Abbildung 20: Aktivitätsdiagramm für die Bildverarbeitung im ImageProcessor

Innerhalb des ImageProcessors werden die Operationen sequentiell abgearbeitet (Siehe Abbildung 20). In Abhängigkeit davon, ob bei einem der Algorithmen eine Unregelmäßigkeit gefunden wurde, ist die Validität des Fundes zu überprüfen, indem die Ergebnisse u.a. durch die Sonnenreflexion-Methoden gefiltert werden. Wird einer dieser Tests bestanden, müssen die Daten für einen Mailversand gespeichert werden. Handelt es sich zusätzlich um das letzte Bild in einem Set wird ein Signal mit den Informationen über die Unregelmäßigkeiten an den Mail Thread emittiert.

Das User Interface wird nach einem erfolgreichen Durchlauf aktualisiert. Dazu gehört ein Update der Anzeigen in sämtlichen Tabs sowie der Übernahme von geänderten Parametern, wie etwa die für die Algorithmen. Ebenso können anschließend wieder Nutzereingaben getätigt werden, beispielsweise Start und Stop des Programms.

9.6 Implementierung auf bestehender Bodenstation

Da im Laufe der Entwicklung eine Testumgebung geschaffen wurde, die der Bodenstation gleicht, kann eine schnelle Portierung auf diese möglich gemacht werden. Um die Software einzurichten benötigt es zwei Komponenten. Diese bestehen aus dem Programm selbst, inklusive zusätzlicher Bibliotheken und dem Konfigurations- bzw. Arbeitsverzeichnis. Während des Betriebes muss gewährleistet sein, dass diese Daten noch dort liegen, wo sie bei der Initialisierung gespeichert waren - sonst kommt es zu unkontrollierbaren Fehlermeldungen und Abstürzen. Darüber hinaus ist der Ort des Zusatzverzeich-

nisses jedoch variabel, sodass das Programm auch auf Bilder von anderen Festplatten oder sogar über das Netzwerk zugreifen bzw. speichern kann.
Ebenso ist zu erwähnen, dass die Bodenstation neben diesem Programm auch noch Andere betreibt, was zu einer hohen Ressourcennutzung im Normalbetrieb führt. Dass das Programm in diesem Betrieb lauffähig ist, wird in der Evaluation gezeigt.

10 Evaluation

10.1 Überblick

In der Evaluation wird zunächst auf die Ressourcennutzung und Laufzeit einzelner Funktionen eingegangen. Danach wird die Treffergenauigkeit der Algorithmen getestet und erläutert, warum darunter auch Falschmeldungen zu finden sind und auf die Hypothese eingegangen, ob Algorithmen mit gleichen Vorgehensweisen auch dieselben Ergebnisse hervorbringen. Dabei wird auch auf die Minimalgröße für Detektionen der jeweiligen Algorithmen eingegangen. Zuletzt werden die tatsächlichen Funde im Laufe der Entwicklungszeit ausgewertet und geprüft, ob die in der Implementierung gesetzte Richtlinie, dass Unregelmäßigkeiten immer gefunden werden sollen, auch wenn dies bedeutet, dass dadurch auch mehr Regelmäßigkeiten nicht gefiltert werden, eingehalten werden konnte.

10.2 Performancetests des Programms auf der Testumgebung und der Bodenstation

Die Laufzeit der Bildprozessierung wurde mit Hilfe eines Timers gemessen, der Millisekunden erfasst. Dabei wurde der Stand des Timers vor dem Aufrufen der Funktion in einer Variable gespeichert und nach dem Ende der Funktion die Differenz zwischen dem gespeicherten Timerstand und dem aktuellen Timerstand gebildet. Somit ergibt sich eine valide Schätzung für die Laufzeit der jeweiligen Funktionen. Tabelle 6 zeigt die minimale, maximale, die durchschnittliche Zeit sowie den Anteil am Durchschnitt einer Funktion im ImageProcessor.
Es ist erkennbar, dass ein sehr großer Teil der Laufzeit auf die Transformation des Bildes fällt, während die Algorithmen samt Filterfunktionen dennoch den größten Anteil an der Gesamtzeit haben. Für den Fall, dass ein Algorithmus nichts findet, ist die Zeit

zum Speichern eines Bildes entsprechend 0, sodass der Gesamtanteil aufgrund der Bildung des arithmetischen Mittelwerts gering bleibt, aber in Einzelfällen zu signifikanter Verzögerung führen kann. Auch die Gesamtlaufzeit kann mit 2,13 Sekunden zu 4,70 Sekunden um mehr als das Doppelte variieren, was auf den erhöhten Aufwand bei Bildern von Kanal 12 zurückzuführen ist.

Funktion	**Min [ms]**	**Mittel [ms]**	**Max [ms]**	**Anteil [%]**
Öffnen	157	158.2	167	5.2
Transformation	611	1048.8	1241	34.3
Heller Fleck	100	114.0	250	3.7
Dunkler Fleck	120	135.1	317	4.4
Formenerkennung	407	540.2	821	17.6
Linienerkennung	482	671.4	1046	21.9
Kreiserkennung	253	344.0	614	11.2
Speichern	0	50.1	241	1.6
Gesamt	2130	3061.8	4697	100.0

Tabelle 6: Laufzeit des ImageProcessors

Ebenso wurde die Ressourcennutzung für das gesamte Programm ermittelt. Eine Betrachtung der einzelnen Funktionen ist hier nicht möglich, da Windows die Speicher- und CPU-Nutzung lediglich für das gesamte Programm angibt. Ermittelt wurden diese mit dem Windows-internen Resource Monitor, der vor dem Start des Programms initialisiert wurde. Abbildung 21 zeigt den Verlauf von CPU-Nutzung in % für den gesamten Prozessor und die Belegung des physikalischen Speichers in MB. Als Testsystem wurden ein AMD Phenom II X4 965 BE mit 4×3.0GHz sowie 10GB 1333MHz DDR3 RAM auf Windows 8.1 Pro 64-bit verwendet.
Für den Zeitpunkt t=0 ist das Programm bereits geöffnet, jedoch nicht initialisiert bzw. gestartet. Nachdem dies geschehen ist, kann ein deutlicher Anstieg beider Kurven beobachtet werden, der daraus resultiert, dass das Programm in diesem Moment sämtliche Masken, den Konfigurationsfile sowie Threads öffnet bzw. initialisiert. Im Anschluss pendelt sich der Verbrauch bei ca. 230MB Arbeitsspeicher und 20% CPU Nutzung ein und erhöht sich für kurze Zeit, wenn ein neues Bild geöffnet wird, abhängig davon wie groß das Bild ist. Beim direkten Vergleich beider Graphen ist deutlich zu erkennen, dass die Häufigkeit der Peaks bei 20s gegenüber denen bei 10s beim Doppelten liegt. Das bedeutet, dass das Programm in beiden Fällen die meiste Zeit im Idle-Modus

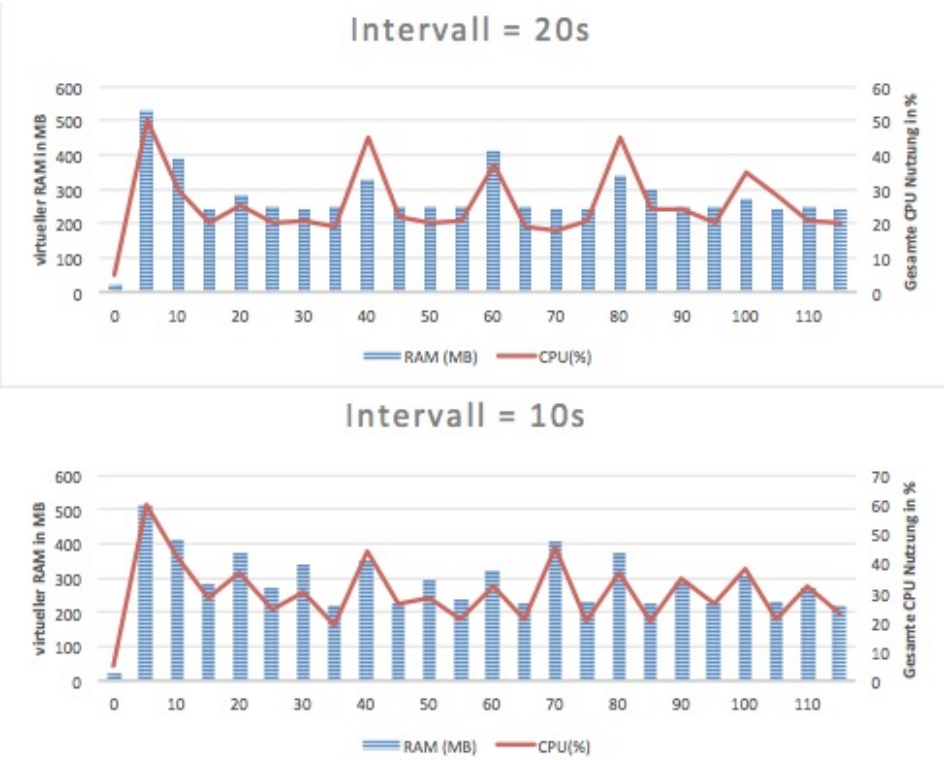

Abbildung 21: CPU- und RAM-Auslastung bei einem Watchdog-Intervall von 10s bzw. 20s

verbringt und diesen nur für neue Bilder verlässt. Eine Optimierung gerade für den Idle-Modus würde dem Programm einen deutlichen Performanceschub bei gleichzeitiger Ressourcenschonung geben. Jedoch ist der auf der Bodenstation verbaute Prozessor leistungsfähiger gegenüber dem in diesem Test verwendeten, sodass der verbrauchte Anteil an der gesamten Rechenleistung für den Einsatzfall noch gesenkt wird.

10.3 Evaluation der Treffergenauigkeit der Algorithmen

Im Folgenden soll anhand der aufgezeichneten Bilddaten ermittelt werden, wie gut die Algorithmen Unregelmäßigkeiten finden. Dabei sind von den insgesamt 12.825 Dateien aus insgesamt vier Monaten zwei der Bildersets mit Unregelmäßigkeiten behaftet, die von hellen bzw. dunklen Flecken resultieren. Um eine bessere Aussage über die Treffergenauigkeit zu geben werden deshalb noch artifiziell angelegte Testszenarien durchlaufen und geprüft, ob die Algorithmen diese erkennen. Solche Aufnahmen werden mit der Bildverarbeitungssoftware GIMP bearbeitet.

Zunächst wird der Helle-Fleck-Algorithmus näher betrachtet. Dieser konnte, wie auch der Dunkle-Fleck-Algorithmus mit echten Unregelmäßigkeiten getestet werden. Wie zu

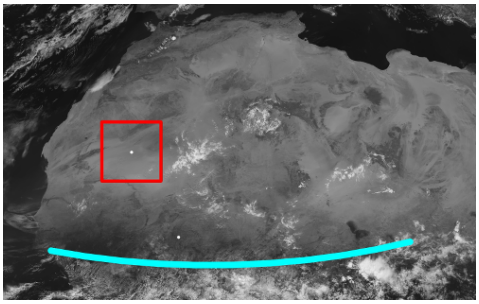

Abbildung 22: Detektion zweier Flecken, wobei der zweite wieder verworfen wird (blau = Bahn der Sonnenreflexion)

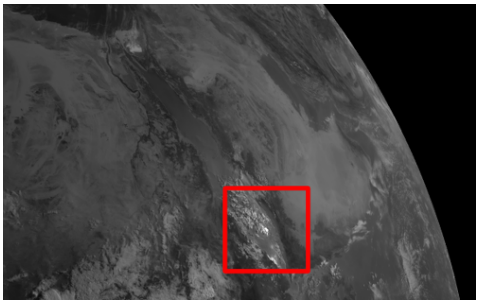

Abbildung 23: Sonnenreflexion einer Wolke als Falschmeldung

erwarten zeigt sich, dass der Fleck bei Annäherung an die Sonnenreflexionsbahn entsprechend herausgefiltert wird. In Abbildung 22 ist zu erkennen, wie zwei künstlich hinzugefügte Flecken auf den Bild erkannt bzw. wieder gefiltert werden, da einer der beiden auf der aktuellen Reflexionsbahn dieses Tages bzw. dieser Uhrzeit liegt.
Insbesondere in den Morgen- und Abendstunden war das Ziel die auftretenden Sonnenreflexionen gezielt auszulassen. Dies konnte teilweise erreicht werden. Jedoch kann aufgrund fehlender Testaufnahmen eines Jahres nicht immer gewährleistet sein, dass die Ellipse auch tatsächlich dort anliegt, wo der aktuelle Pfad der Sonnenreflexion ist. So ergeben sich zeitweise Falschmeldungen, wie in Abbildung 23 dargestellt.

Die Hypothese, dass der Dunkle-Fleck-Algorithmus aufgrund seiner Ähnlichkeit zum Hellen-Fleck-Algorithmus auch ähnliche Ergebnisse liefert, bestätigt sich. Die Anwendung der Land-Maske führt jedoch dazu, dass ein großer Bereich entsteht, in dem potenzielle Unregelmäßigkeiten als Sonnenreflexion gefiltert werden. Eine Verkleinerung des Bereichs durch Variation der `isLandmass()`-Parameter ist deshalb für die Zukunft erforderlich. Abbildung 24 zeigt eine künstliche Detektion in einem hellen und dunklen Bereich, wobei diese sich weit am Rand befinden. Beide werden erkannt, was zeigt, dass der Algorithmus für die meisten Uhrzeiten auch Pixel am Rand erkennen kann, kontrastunabhängig ist und bis zu einem gewissen Grad auch unabhängig von den umgebenden Pixeln arbeitet. Dabei ist zu erwähnen, dass die direkten Nachbarpixel einer Detektion ebenfalls stets in die Threshold-Operation mit einbezogen werden und niemals

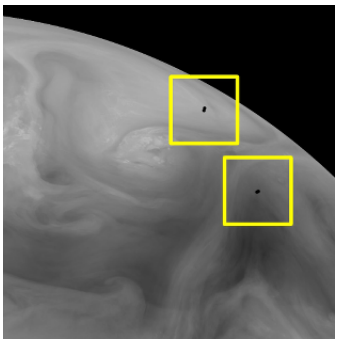

Abbildung 24: Detektion von dunklen Flecken am Rand eines Bildes von Kanal 5

ausschließlich der zentrale Pixel betrachtet wird. Wären die Punkte außerdem zu einer anderen Uhrzeit vorgekommen, so könnte die `isSunReflection()`-Funktion dazu führen, dass beide nicht als Unregelmäßigkeit erkannt werden.

Der Formenalgorithmus besitzt, wie auch die Folgenden keinerlei reale Aufnahmen zum Testen, weshalb dieser ausschließlich auf künstlich produzierten Bildern beruht und auch für diese optimiert wurde. Damit der Algorithmus Formen findet, aber gleichzeitig wenige Fehlmeldungen liefert, ist der Schwellwert für den Canny-Operator während der Implementierung angehoben worden. Die Evaluierung dieses Schrittes zeigt, der Schwellwert teilweise noch höher hätte liegen können. Darauf wurde verzichtet, um mögliche Unregelmäßigkeiten nicht zu übersehen. Abbildung 25 zeigt ein Beispiel für eine günstige Detektion bei kontrastreichem Hintergrund, während rechts davon ein Fehlschlag in einem kontrastschwachen Bereich dokumentiert wird. In Ersterer deutlich erkennbar ist die Farben- und Größenunabhängigkeit des Algorithmus, was ihn ebenso für andere Einsatzgebiete potenziell interessant macht.

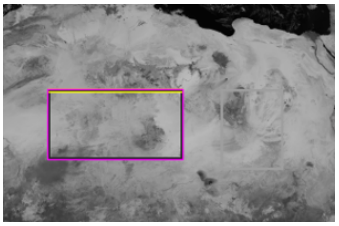

Abbildung 25: Erfolgreiche und fehlgeschlagene Detektion eines Rechtecks

Der Linienalgorithmus kann nach seiner Optimierung zu einem probabilistischen Hough-Algorithmus zwischen Linien innerhalb und außerhalb der Erdscheibe unterscheiden. Dies ist wichtig, da damit ein großer Teil der fälschlicherweise gefundenen Linien wegfällt. Er ist in der Lage Linien innerhalb der Erdscheibe zu erkennen, wie exemplarisch in Abbildung 26 zu erkennen ist. Dabei ist er auch kontrastabhängig, jedoch aufgrund seines signifikant geringeren Canny-Schwellwertes (von 200:600) gegenüber dem Formenalgorithmus (1000:2000) deutlich weniger anfällig auf Kontraständerungen. Dabei kann es dazu kommen, dass der Algorithmus Geraden an Stellen erkennt, an denen aber keine vorliegen.

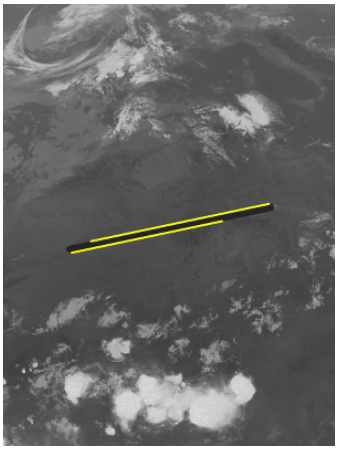

Abbildung 26: Detektion zweier Geraden an einer dunklen Linie

Die Ursache dieses Problems stellt die grundlegende Herangehensweise an die Liniensuche dar. Der Hough-Algorithmus sucht innerhalb eines Bildes Linien, auf denen möglichst viele Konturpunkte liegen, egal woher diese Kontur stammt. Für den Ein-

satzfall ist diese Herangehensweise zwar auch zielführend, jedoch wäre eine Suche nach geraden Konturen (wie sie entlang einer Linie vorkommen) vorteilhafter gewesen. Diese Problematik äußert sich vor allem dann, wenn Sonnenreflexionen innerhalb des Bildes existieren, die hohe Kontraständerungen auf einem kleinen Bereich und damit viele harte Kanten hervorrufen. Dann registriert der Algorithmus diese und findet mehrere Linien entlang dieser Konturen.
Um die Häufigkeit solcher Falschmeldungen gering zu halten, sind die Schwellwerte angepasst worden, was wiederum dazu führt, dass manche Geraden gar nicht erkannt werden. Dadurch, dass die Parameter des Programms aber anpassbar sind können die zwei zentralen Schwellwerte dennoch so eingestellt werden, dass es weniger Falschmeldungen gibt. Der Erste ist dabei deutlich kleiner, als etwa der für die Formenerkennung (siehe Oben), wodurch grundsätzlich, wie in Abbildung 8 erkennbar, mehr Konturen gefunden werden. Um dabei die Anzahl der gefundenen Linien gering zu halten wird der zweite Schwellwert[21] erhöht. So ergibt sich, dass eine Linie erst dann gefunden wird, wenn fast alle Punkte auf dieser auch vorhanden sind - wie es bei einer geraden Kontur immer der Fall ist.
Der Kreis-Algorithmus besitzt aufgrund seiner strukturellen Ähnlichkeit zur Linienerkennung dasselbe Problem, jedoch nicht ganz so stark ausgeprägt, da selbst Binärbilder mit vielen Kanten nur selten so viele Punkte auf einem Kreisbogen haben, wie der Schwellwert des Algorithmus (Standardeinstellung des Programms sind 50 Punkte) es vorgibt. Ein Vorteil dieses Algorithmus ist dabei auch, dass er Kreise finden kann, die nicht komplett sichtbar sind, so lange zumindest ein Teil der Kontur erkannt wird. Dies ist aber nur bei einer gewissen Mindestgröße gegeben, worauf im folgenden Kapitel eingegangen wird.

10.4 Minimale Größe der Formen und Flecken für eine Detektion

Um den minimalen Durchmesser d für die einzelnen Formen zu ermitteln wurden in ein Testbild kleiner werdende Formen mit gleicher Orientierung und Helligkeit vor einen kontrastreichen bzw. -armen Hintergrund gesetzt und getestet, bis zu welchem Durchmesser d bzw. Radius r die Form als solche erkannt wird. Für die hohe Kontrastdifferenz der Hellwerte (bzw. Dunkelstufen) zwischen Form und Hintergrund wurden dabei 100, für die niedrige 10 angenommen.
Diese Messung ist notwendig, da beispielsweise die Winkelberechnung des Mehreck-

[21] Für die Anzahl der Punkte, die auf einer Linie sein müssen, um als solche erkannt zu werden

Algorithmus stärker verfälscht wird, je kleiner die Form wird, da Pixel nur in Winkeln als Vielfaches von 45° zueinander liegen können. Auch bei den Fleckenalgorithmen kann es vorkommen, dass bei kleiner werdendem hellen bzw. dunklen Bereich dieser nicht mehr erkannt wird, weshalb hierfür ebenfalls der Minimalradius evaluiert wird.

Detektierte Form	**Hoher Kontrast [px]**	**Niedriger Kontrast [px]**
Kreise	r = 20	r = 52
Fünfecke	d= 77	-
Sechsecke	d = 80	-
Vierecke	d = 25	-
Heller Fleck	r = 4	r = 4
Dunkler Fleck	r = 5	r = 5

Tabelle 7: Minimale Größe von Formen bzw. Flecken, um für die Algorithmen detektiert zu werden

Der Kreisalgorithmus liefert, wie bereits in Kapitel 9.1.1 angegeben, für Radien kleiner als 5 Pixel keine verwertbaren Ergebnisse mehr, weshalb dieser Bereich ausgegrenzt wurde. Auch darüber lässt sich kein Kreis finden, erst bei einem Radius von 20 Pixeln werden Kreise erkannt. Für Teile eines Kreisbogens gilt, dass mindestens 55 Konturpunkte auf diesem liegen müssen, um als Kreis detektiert werden zu können, wobei der Wert nachträglich anpassbar ist. Bei geringem Kontrastverhältnis liefert der Algorithmus erst bei mehr als doppelt so großen Kreisen ein Ergebnis.
Der Formenalgorithmus für Fünf- und Sechsecke ist derjenige, der am meisten von der zunehmenden Verpixelung der Formen leidet, da diese auf eine korrekte Winkelmessung aufbauen. Folgend werden die Formen auch nur bis zu einer Minimalgröße bis zu 80 Pixeln erkannt. Bei geringem Kontrast kann der Algorithmus aufgrund fehlender Kanten keine Ergebnisse liefern.
Der Formenalgorithmus für Vierecke liefert dagegen bei hohem Kontrast ein besseres Ergebnis, da er als Einziger von der Verfälschung der Winkelmessung weitestgehend ausgenommen ist (Seine Innenwinkel betragen im Optimalfall 90°). So werden die rechtwinkligen Innenwinkel immer erkannt und die minimale Größe des Bereichs lediglich durch die implementierte Flächenbegrenzung limitiert. Allerdings verschlechtert sich auch er,wie die anderen Formenalgorithmen, wenn sich das Kontrastverhältnis verschlechtert.

Dies gründet darauf, dass mit dem Canny-Operator als Ausgangspunkt alle Algorithmen auf einer korrekten Konturfindung aufbauen. Werden die Kanten unschärfer und damit das Kontrastverhältnis kleiner, werden diese nicht mehr gefunden. Da dies ein grundlegendes Problem darstellt, ist der Formenalgorithmus für Vier-, Fünf- und Sechsecke für diesen Anwendungsfall ungeeignet.
Helle und dunkle Flecken sind bei der Größe des Bereichs lediglich dadurch limitiert, dass je nach Helligkeitswert ein Bereich von minimal 4×4 Pixeln benötigt wird. Dieser Bereich ist somit signifikant kleiner als der, der für die Formen benötigt wird. Ebenso arbeitet der Algorithmus kontrastunabhängig, weshalb er auch bei unscharfen Aufnahmen noch Unregelmäßigkeiten erkennen kann, vorausgesetzt die Helligkeitswerte bleiben gleich.

10.5 Übersicht über gefundene Unregelmäßigkeiten

Die insgesamt fünf Bilder der zwei Unregelmäßigkeiten sollen in diesem Kapitel vorgestellt werden, wobei auch auf deren Ursprung eingegangen wird.
Die erste Unregelmäßigkeit stellt die vom 28.04.2015 dar und ist ein heller Fleck auf den sichtbaren Kanälen 1, 2 sowie dem HRV-Kanal 12f. Seine Position ist über dem östlichen Teil des afrikanischen Kontinents in einem relativ wolkenarmen Gebiet, was diese als Ursache unwahrscheinlich macht. Jedoch liegt der Punkt zentriert über einem Flusslauf bzw einem kleineren See, was eine Reflexion an der Wasseroberfläche vermuten lässt. Durch seine längliche Form und schwarze Ränder kann es jedoch ebenso um einen Bildfehler des optischen Systems handeln. Eine Überprüfung durch den implementierten Hellen-Fleck-Algorithmus mit anschließender Filterung nach Sonnenreflexionen zeigt jedoch deutlich, dass der Fleck zentral in dem Bereich für Sonnenreflexionen zu dieser Uhrzeit liegt. Dennoch lassen sich andere Ursachen für den Ursprung dieses hellen Flecks nicht ausschließen.
Darüber hinaus wurden mehrfach dunkle Flecken ebenfalls auf dem östlichen Teil von Afrika entdeckt, die alle die gleiche Position und Größe haben. Ein Vergleich mit einer Karte macht sichtbar, dass die Kontur des Punktes genau der des "Lake Tana" entspricht. Es liegt also nahe, dass es sich bei diesem dunklen Fleck um den See handelt, der bei passendem Sonnenstand nur eine geringe Menge an Licht in einem speziellen Infrarotbereich reflektiert, wodurch in diesem Bereich des Bildes ein dunkler Fleck entsteht.

Warum dies die beiden einzigen Unregelmäßigkeiten sind, die während Entwicklung, Implementierung und Test gefunden werden konnten, soll im Folgenden diskutiert werden.

11 Diskussion und Ausblick

11.1 Diskussion der Evaluation

Die Evaluation hat zunächst gezeigt, dass das Programm unter den gegebenen Umständen in der Lage ist auf der Bodenstation für einen längeren Zeitraum die empfangenen Bilder auszuwerten bzw. die Ergebnisse adäquat zu präsentieren. Darüber hinaus konnte festgestellt werden, dass es ebenso auf anderen Betriebssystemen und Architekturen funktioniert. Dies zeigt, dass das Programm mit geringem Aufwand auch auf weitere Systeme portierbar ist. Jedoch konnte durch den großen Umfang der Software nur begrenzt sichergestellt werden, dass das Programm optimal resourcensparend arbeitet, sodass in diesem Bereich eine Optimierung der internen Prozesse gerade für kleiner dimensionierte Systeme notwendig ist.
Im Weiteren konnte darüber hinaus belegt werden, dass es möglich ist die Bilder von Meteosat-10 mit eigens entwickelten Algorithmen auf Unregelmäßigkeiten zu untersuchen. Die Evaluation der Treffergenauigkeit hat jedoch auch gezeigt, dass es einerseits eine schwierige Aufgabe darstellt Unregelmäßigkeiten mit konkreten Werten zu charakterisieren, wodurch sich ein Zustand der ständigen Anpassung einstellt. Eine nachträgliche Optimierung konnte durch eine Parameterdatei im XML Format realisiert werden, die sämtliche zentralen Parameter enthält.
Andererseits wurde die in Kapitel 5.2 aufgestellte Hypothese belegt, dass die meisten Himmelsphänomene nur sehr selten auf Bildern festgehalten werden können, was die geringen Anzahl an Detektionen aus Kapitel 10.5 belegt. Das macht deutlich, dass eine genaue Untersuchung der Herkunft und Ausprägung einzelner Unregelmäßigkeiten nur dann möglich ist, wenn man über einen langen Zeitraum unter gleichen Bedingungen einen möglichst großen Bereich der Atmosphäre überwacht.

11.2 Ausblick und Erweiterbarkeit

Die Software bietet aufgrund ihrer flexibel gestalteten Struktur gleich mehrere Möglichkeiten diese relativ einfach zu ändern bzw. vergrößern. Es ist deshalb denkbar, dass der MSG-PhenomenaDetector in zukünftigen Projekten um Funktionen, wie etwa die Folgenden, erweitert wird.
So bietet das Programm die optimale Grundlage für ein Archivierungssystem der Meteosat-10 Wetteraufnahmen. Das hierzu nötige IO-Interface ist aufgrund der Einbindung von OpenCV und Qt bereits gegeben, es fehlt lediglich eine Funktion, die den Status der Festplatte überwacht und nach Bedarf alte Bilder löscht. Ebenso sind sämtliche Algorithmen auch auf beliebige andere Aufnahmen von Wettersatelliten anwendbar, wie etwa Meteosat-7 oder 9. Auch ein Einsatz auf Bildern der NOAA-Satelliten in Polarorbits wäre denkbar, jedoch setzt dies eine Anpassung der Sonnenreflexionsalgorithmen voraus, da diese dann nicht mehr von einer statischen Position des Sonnenstandes ausgehen können.
Darüber hinaus kann das Programm mit wenigen Änderungen auf anderen Architekturen bzw. Betriebssystemen lauffähig gemacht werden. Das so entstandene System, etwa ein Kleinstrechner, wie der Raspberry-Pi, könnte dazu auch über ein Netzwerk auf die Bilder zugreifen, sodass keine direkte Kopplung von Programm und Bodenstationsrechner notwendig ist.
Das im Rahmen dieser Arbeit entwickelte Programm stellt einen ersten Schritt zur Erforschung von Himmelsphänomenen mit Satellitendaten dar und bietet somit eine Basis zur weiteren Analyse dieser Anomalien. Da es sich hierbei um eine Grundlagenarbeit handelt, bietet die Software eine Plattform um die Implementierung neuer Algorithmen möglich zu machen. Dabei ist es dem Entwickler freigestellt, ob er die bestehenden und erfolgreich angewandten Algorithmen um neue Funktionen ergänzt oder versucht neue Ideen einzubauen, um so auf anderem Wege Unregelmäßigkeiten sichtbar zu machen.
Zusammenfassend lässt sich feststellen, dass mit der entwickelten Software “MSG-PhenomenaDetector” die Auswertung von Unregelmäßigkeiten in den Bilddaten von Meteosat Wettersatelliten erfolgreich durchführbar ist.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Bradski und Kaehler, 2008] Bradski, G. und Kaehler, A. (2008). *Learning OpenCV - Computer Vision with the OpenCV Library*. O'Reilly Media.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*.

[Conway, 1997] Conway, E. D. (1997). *An Introduction to Satellite Image Interpretation*. The Johns Hopkins University Press.

[Dawson-Howe, 2014] Dawson-Howe, K. (2014). *A practical introduction to Computer Vision with OpenCV*. John Wiley and Sons Ltd.

[Erhardt, 2008] Erhardt, A. (2008). *Einführung in die digitale Bildverarbeitung*. Vieweg+Teubner.

[Fleck, 2013] Fleck, B. (2013). Staring at the sun - soho factsheet. Technical report, European Space Agency, ESA.

[Kayal, 2015] Kayal, H. (2015). Multi sensor plattform. Lehrstuhl für Informatik VIII (Hrsg.): Presentation about ADS-B, EumetCast, NOAA and SkyCAM.

[Ley et al., 2009] Ley, W., Wittmann, K., und Hallmann, W. (2009). *Handbook of Space Technology*. John Wiley and Sons Ltd., 1. auflage edition.

[Mandl, 2014] Mandl, P. (2014). *Grundkurs Betriebssysteme, 4. Auflage*. Springer Vieweg.

[Müller, 2007] Müller, J. (2007). *MSG Level 1.5 Image Data Format Description*. Eumetsat. Verfügbar auf: `http://www.eumetsat.int/website/wcm/idc/idcplg?IdcService=GET_FILE&dDocName=PDF_TEN_05105_MSG_IMG_DATA&RevisionSelectionMethod=LatestReleased&Rendition=Web`, version 7, aufgerufen am 18.07.2015.

[Nash, 2012] Nash, B. (2012). *Detecting simple shapes in an image*. Verfügbar auf `http://opencv-code.com/tutorials/detecting-simple-shapes-in-an-image/`, aufgerufen am 19.07.2015.

[Piwowarczyk und Johns, 2012] Piwowarczyk, J. und Johns, D. (2012). *SMTP Client*. Verfügbar auf `http://www.codeproject.com/Articles/98355/SMTP-Client-with-SSL-TLS`, aufgerufen am 25.07.2015.

[Roa und Felipe, 2012] Roa, C. und Felipe, A. (2012). Activity in the lunar surface: Transient lunar phenomena. *Tumbaga journal of the University of Tolima - Vol 1, No 7*.

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift