

Diplomarbeit

ALARM PRE-RECORDING

Ausgeführt im Schuljahr 2018/19 von:

Jennifer Romirer-Maierhofer 5BE
Veronika Stangl 5BE

Betreuer:

Ing. Josef Feichtl, BEd
Ing. Josef Feichtl, BEd

Abgabedatum: 05.04.2019

Pinkafeld, am 04.04.2019

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Pinkafeld, am

Verfasser / Verfasserinnen:

Jennifer Romirer-Maierhofer

.....

Veronika Stangl

.....

Kurzfassung

Diese Diplomarbeit beinhaltet die Entwicklung eines Kamerasystems zur Überwachung von SPS-gesteuerten Maschinen.

Im Betrieb sind auftretende Fehler einer Maschine und ihre Störursachen oft nur schwer rekonstruierbar, wodurch Fehler immer wieder auftreten. Durch ein automatisch aufgenommenes Video zum und vor einem Störfall sollen auftretende Fehler identifizierbar und Störungsursachen erkennbar werden. Durch diese Rekonstruktionsmöglichkeit kann die Effizienz von Maschinen erheblich gesteigert werden.

Hierfür wurde ein Kamerasystem, bestehend aus Raspberry Pi und Webcam, entwickelt. Es wurde eine Möglichkeit geschaffen, bei SPS-Störmeldungen automatisch ein Video zu speichern, um die Fehlersituation später visuell darstellen zu können. Die Kamerasysteme haben flexible Vor- und Nachlaufzeiten, um die Situation vor und nach der Störung darstellen zu können. Diese Zeiten sind unabhängig voneinander einstellbar.

Darüber hinaus wurde ein Benutzerprogramm in der Programmiersprache C# implementiert, das die Möglichkeit bietet, bei unterschiedlichen Fehlermeldungen einer Maschinensteuerung das Video spezifischer Kameras zu speichern und anzuzeigen. Zudem können von diesem Benutzerprogramm aus sämtliche Verbindungen hergestellt und weitere relevante Einstellungen vorgenommen werden.

Abstract

This thesis includes the development of a camera-system to supervise SPS-controlled machines.

Failures on machines and their cause of malfunction that happen during production are hard to reconstruct, thus failures happen again and again. Through an automatically recorded video at and before the occurring failure, upcoming failures can be identified and the cause of malfunction recognized.

This opportunity to reconstruct causes for failure can increase the efficiency of machines considerably.

A camera-system, consisting of a Raspberry Pi and a webcam, was developed. The opportunity to automatically save videos at the event of SPS-fault, and thus to visualize the failure situation, was created. The camera-systems have flexible lead and follow-up times, and can illustrate the situation at and before the event of fault. These times can be configured independently.

The additional user program was implemented in the coding language C#, this provides the opportunity to store and illustrate videos of specific cameras at the event of different failures at a machine control. Moreover the user program has the ability to create all connections and conduct further relevant configurations.

Personaldaten

Verfasser:

ROMIRER-MAIERHOFER Jennifer
Obere Hauptstraße 76
7422 Riedlingsdorf
jenny-romirer@gmx.at



Schule:

HTBLA Pinkafeld
Abteilung Elektronik und Technische Informatik

Aufgabenbereich:

Software

Betreuer:

Ing. Josef Feichtl, BEd

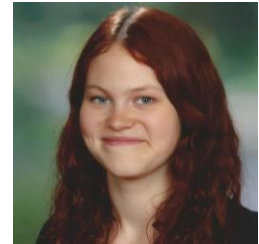
Datum:

Haupttermin Schuljahr 2018/19

Personaldaten

Verfasser:

STANGL Veronika
Günserstraße 44
2860 Kirchschlag
veronika99@gmx.at



Schule:

HTBLA Pinkafeld
Abteilung Elektronik und Technische Informatik

Aufgabenbereich:

Hardware und Software

Betreuer:

Ing. Josef Feichtl, BEd

Datum:

Haupttermin Schuljahr 2018/19

Inhaltsverzeichnis

KAPITELAUFTeilUNG	10
1 AUFGABENSTELLUNG	11
1.1 Ausgangssituation und Relevanz.....	11
1.2 Ziele.....	12
1.3 Begriffssammlung	12
2 PROJEKTPLANUNG	14
3 PROJEKTREALISIERUNG	18
3.1 Gesamtaufbau.....	18
4 AUSWAHL DES KAMERASYSTEMS.....	19
4.1 Alarm Pre-Recording Kamera	19
4.2 Ausschlusskriterien	22
5 KAMERASYSTEM MIT RASPBERRY PI UND WEBCAM	23
5.1 Kennwerte der WebCam.....	24
5.2 Verbindungseinstellungen	25
5.2.1 RaspberryPi-Verbindung zum Leitrechner	25
5.2.2 Leitrechner-Verbindung zum Raspberry Pi	25
5.2.3 Putty Theorie und Verwendung	25
5.2.4 WinSCP Theorie.....	26
5.3 OpenCV Theorie.....	29
5.3.1 Benötigte Installationen	30
5.3.2 Konfigurationen.....	30
5.4 Kameraprogramm.....	31
5.4.1 CSV-File auslesen	33
5.4.2 File-Polling	33
5.4.3 File erzeugen	34
5.4.4 Kamera und Video.....	35
5.5 Autostart.....	37
6 AGLINK4 – SPS VERBINDUNG	39
6.1 AGLINK Bibliothek.....	41

6.2	<i>SPS Verbindung</i>	41
6.3	<i>SPS Datenbausteine auslesen</i>	43
7	BENUTZERPROGRAMM – ALARM PRE- RECORDING - EINSTELLUNGEN	44
7.1	<i>Einführung</i>	44
7.1.1	Flussdiagramm – Visio	44
7.2	<i>Grafische Oberfläche</i>	47
7.2.1	Laden der Oberfläche	48
7.2.2	Windows Panels öffnen	48
7.2.3	Videoaufnahme starten	50
7.2.4	Ändern der Meldetextzeile	51
7.2.5	Leeren der Meldezeile	52
7.2.6	Beenden	53
7.3	<i>Einstellungen</i>	53
7.3.1	Variablen	53
7.3.2	Speichern in einer CSV Datei	54
7.3.3	Speichersystem	54
7.3.4	Einstellungen speichern – Hintergründe	56
7.3.5	Laden aller Konfigurationen aus dem Einstellungsfile	59
7.3.6	Die Konfigurationsdaten auf den Standardwert zurücksetzen	61
7.3.7	Ändern der Größe eines Arrays	61
7.3.8	Neuladen des Datensatzes bei einer Kameraanzahländerung	62
7.3.9	Neuladen des Datensatzes bei einer Datenbitänderung	62
7.4	<i>Projekte laden oder erstellen</i>	63
7.4.1	Projekt erstellen	63
7.4.2	Projekt laden	65
7.5	<i>Hauptprogramm-Statusanzeige</i>	66
7.5.1	Start der Videoaufnahme-Form	67
7.6	<i>Kameraanzahl</i>	67
7.6.1	Speichern der Kameraanzahl	68
7.7	<i>Datenbaustein – Meldebits</i>	70
7.8	<i>SPS Verbindung</i>	72
7.8.1	SPS-Verbindungskonfigurator öffnen	72
7.8.2	Verbindung mit der SPS	75
7.8.3	Verbindung mit der SPS trennen	76
7.9	<i>Videoanzeige mittels Windows Media Player</i>	77
7.9.1	Das Suchen eines Videos im Projektordnerverzeichnis	79
7.9.2	Änderung der ausgewählten Videos	80
7.10	<i>Vor- und Nachlaufzeiten</i>	81
7.10.1	Speichern der Vor- und Nachlaufzeiten	82
7.11	<i>Zuweisung der Meldebits</i>	84
7.11.1	Meldungszuweisungen speichern	86
8	BENUTZERPROGRAMM - VIDEOAUFNAHME STARTEN	89

8.1	<i>Videoaufnahme</i>	89
8.1.1	Erstes Laden der grafischen Oberfläche	89
8.1.2	Vor- und Nachlaufzeiten senden	90
8.1.3	Meldezeile	90
8.1.4	Fehler: Cross-Threads.....	90
8.1.5	Beenden	92
8.1.6	Zurück zu den Haupteinstellungen	92
8.1.7	Schließen der gesamten Form.....	93
8.2	<i>Kameraverbindung</i>	94
8.2.1	Verbindungsaufbau mit dem Raspberry Pi	94
8.2.2	Verbindung mit dem Raspberry Pi trennen	95
8.2.3	Verbindung mit dem Raspberry Pi erneut aufbauen.....	96
8.2.4	Dateien auf den Raspberry Pi laden.....	96
8.2.5	Dateien vom Raspberry Pi laden	96
8.2.6	Verbindung mit dem Raspberry Pi herstellen und alle Dateien senden	97
8.2.7	Dateien vom Raspberry Pi entfernen	97
8.2.8	Überprüfung, ob der Raspberry Pi verbunden ist	97
8.3	<i>Verbindungsaufbau mit den Kameras</i>	97
8.4	<i>Senden der Dateien an den Raspberry Pi</i>	98
8.4.1	Senden der Kameraaufnahmezeiten.....	98
8.4.2	Videoaufnahme am Raspberry Pi anfragen	99
8.5	<i>Automatisierter Ablauf - Datenbaustein</i>	99
8.5.1	Einführung	99
8.5.2	Initialisieren des Ablaufes	101
8.5.3	Sekundliches Abfragen der Datenbits.....	101
8.5.4	Datenbits aus der SPS lesen.....	104
8.5.5	Video downloaden	105
8.6	<i>Kamerastatus</i>	107
8.7	<i>Videoanzeige mittels Windows Media Player</i>	107
9	BEDIENUNGSANLEITUNG BENUTZERPROGRAMM	108
10	REALAUFBAU	117
11	ZUSAMMENFASSUNG	120
11.1	<i>Ergebnisse</i>	120
11.2	<i>Erkenntnisse</i>	120
11.3	<i>Zukunftsblick</i>	121
12	ABBILDUNGSVERZEICHNIS	122
13	LITERATURVERZEICHNIS	126

Kapitelaufteilung

<i>Aufgabenstellung:</i>	Veronika Stangl
<i>Projektplanung:</i>	Veronika Stangl
<i>Projektrealisierung:</i>	Veronika Stangl
<i>Kamerasystem:</i>	Veronika Stangl Auswahl des Kamerasystems, Kamerasystem mit Raspberry Pi und Webcam
<i>AGLink – SPS-Verbindung:</i>	Veronika Stangl
<i>Benutzerprogramm:</i>	Jennifer Romirer-Maierhofer Hauptprogramm, Videoaufnahme
<i>Bedienungsanleitung:</i>	Veronika Stangl
<i>Realaufbau:</i>	Jennifer Romirer-Maierhofer
<i>Zusammenfassung:</i>	Jennifer Romirer-Maierhofer

1 Aufgabenstellung

1.1 Ausgangssituation und Relevanz

Im Zuge der Industrie 4.0 soll die Effizienz von Maschinen und ihren Komponenten gesteigert werden. Idealerweise sollten Fehler vor ihrem Eintreten als solche erkannt werden, um Standzeiten zu verhindern. Kommt es dennoch zu einem Fehlerfall, sollte sichergestellt werden, dass die Ursache schnell gefunden wird, denn ein langes Stillstehen der Maschinen kann zu erheblichen finanziellen Verlusten führen.

Fehler bleiben oft längere Zeit unerkannt, weil Bediener/innen diese nicht gleich wahrnehmen. Dadurch wird die Stehzeit der Maschine unnötig verlängert. Um eine weitere Verzögerung zu verhindern, werden auftretende Fehler oft quittiert und so schnell wie möglich beseitigt, damit die Produktion wieder aufgenommen werden kann. Dadurch sind die Situation zum Fehlerzeitpunkt und die Fehlerursache später nicht mehr nachvollziehbar.

Diese Information ist jedoch für den Auftraggeber und andere Hersteller und Warter von Maschinen von großem Interesse, da ein vollständiges Rekonstruieren der Situation zum Fehlerzeitpunkt und der Störungsursache kaum bis gar nicht möglich ist. Ohne diese Möglichkeit der Rekonstruktion können diese Faktoren jedoch nur schwer identifiziert und behoben werden, wodurch dieselben Fehler immer wieder auftreten. Dieses Problem stellt sich auch der Auftragsfirma, DAM – Dynamic Assembly Machines Anlagenbau GmbH. Es führt in vielen Fällen zu aufwendigen Fehlersuchen, die durch ein automatisch aufgenommenes Video der Maschine während der Störung umgangen werden können. Um umfassende Informationen der Störursache zu erhalten, soll dieses Video nicht nur ab dem Auftreten der Störung, sondern bereits vor dem Störfall aufgezeichnet werden.

Durch das automatische Aufnehmen und Speichern eines Videos zum und vor dem Fehlerzeitpunkt sollen die auftretenden Fehler identifizierbar und die Störungsursachen erkennbar werden, wodurch eine schnelle Reparatur und Fehlervermeidung ermöglicht wird. Diese Fehleraufzeichnung bringt somit finanzielle Vorteile für Wartungs- und Produktionsfirmen.

1.2 Ziele

Im Zuge dieser Diplomarbeit soll eine Möglichkeit geschaffen werden, bei Störmeldungen der Maschinensteuerung automatisch eine Videosequenz zu speichern, um die Fehlersituation später visuell darstellen zu können. Hierbei wird das, von den Kameras permanent aufgezeichnete, Videosignal zu einem Leitrechner übertragen. Die Kameras sollen flexible Vor- und Nachlaufzeiten haben, die die Fehlersituation vor und nach der auftretenden Störung aufnehmen sollen. Diese Zeiten sollen unabhängig voneinander einstellbar sein. Darüber hinaus soll die zu entwickelnde Software die Möglichkeit bieten, bei unterschiedlichen Fehlermeldungen der Maschine das Videosignal spezifischer Kameras zu speichern.

Die Gesamtzeit des Videos sollte höchstens eine Minute betragen und am Leitrechner in einen Ringpuffer gespeichert werden. Dieses Speichern erfolgt mit einer systematischen Bezeichnung, die das Datum, die Uhrzeit und die Kamera-ID enthält. Die Verbindungen zwischen Leitrechner und Kamera, sowie die Verbindung zwischen Leitrechner und Steuerung sollen über Ethernet erfolgen, siehe Abbildung 1.

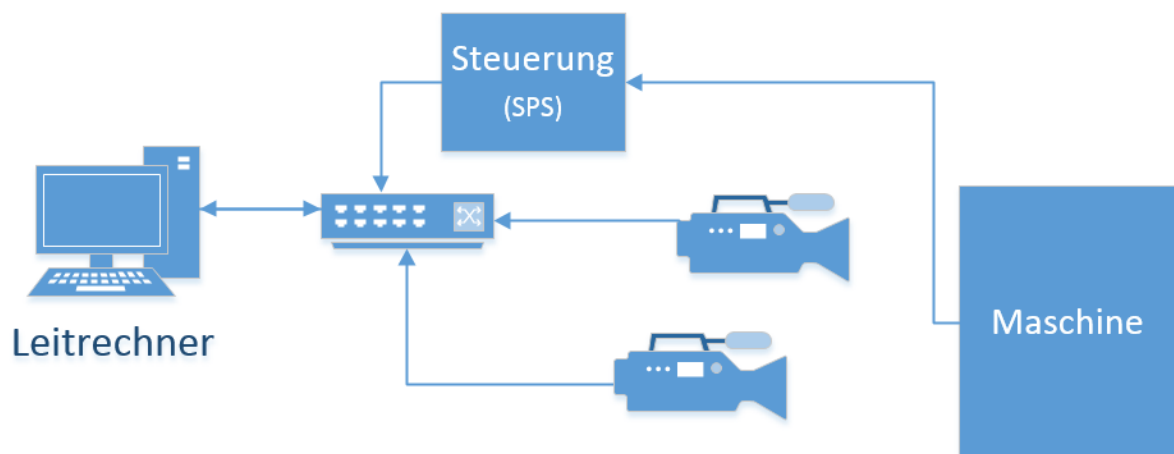


Abbildung 1: Blockschaltbild Aufgabenstellung

1.3 Begriffssammlung

Im Folgenden werden Fachausdrücke verwendet.

Der PC, auf dem das Benutzerprogramm läuft und der als Schnittstelle für die Verbindung der Komponenten dient, wird als Leitrechner bezeichnet.

Kursiv geschriebene Begriffe weisen auf Eigennamen und Sonderfälle hin. Dies kann von Methodennamen bis zu Datentypen reichen.

Die SPS (Speicherprogrammierbare Steuerung) ist ein programmierbarer Computer der für die Steuerung von Maschinen genutzt wird.

Ein Ringpuffer ist ein Speicher mit einer vorgegebenen fixen Länge, der sequentiell beschrieben wird. Wenn der Speicher voll ist, werden die ältesten Daten von den neuen überschrieben.

Als *intern* werden Programmteile oder Programmvariablen bezeichnet. Als *extern* werden die am PC gespeicherten Dateien bezeichnet. Diese befinden sich im Projektordner.

2 Projektplanung

Das Projekt wurde vorab mit der Software SAP geplant. Hierfür wurde ein Projekt erstellt, dem die entsprechenden Teilschritte und Meilensteine hinzugefügt wurden, siehe Abbildung 2. Den Teilschritten und Meilensteinen wurden Termine zugewiesen und der Stundenaufwand geschätzt, siehe Abbildung 4. Dies wurde in einem, in Abbildung 3 ersichtlichen, Balkendiagramm dargestellt.

Projektstruktur: Bezeichnung	Identifikation
▼ Alarm Prerecording Romirer_Stangl	ALARM_PRERECORDING
▼ Konfiguration Projektauftrag 18	ALARM_PREREC-AUFTRAG
▼ Projektauftrag	4000004
• Projektauftrag formulieren	4000004 0010
▼ Projektauftrag unterschreiben	4000004 0020
• Projektauftrag unterschrieben	82
▼ Projektplanung	ALARM_PREREC-PLAN
• Teilschritte planen	4000004 0030
• Stundenaufwand schätzen	4000004 0040
• Arbeitspakete verteilen	4000004 0050
• Balkenplan erstellen	4000004 0060
▼ Meilensteine festlegen	4000004 0070
• Projektplanung abgeschlossen	83
▼ Projektrealisierung	ALARM_PREREC-REALIS
▼ Realisierung Hardware	ALARM_PREREC-HARDW
▼ Kamerasystem	4000004 0080
• Kamerasystem funktionsfähig	182
▼ Verbindung mit Kamerasystem	4000004 0089
• Kamerasystem funktionsfähig	407
▼ Material beschaffen	4000004 0090
• Hardware fertig	264
• Bausatz Kamerasystem	0030 SET-KAMERA
▼ Realisierung Software	ALARM_PREREC-SOFTW
• Einlesen in Thematik	4000004 0140
• Lizenzen beschaffen	4000004 0150
▼ Verbindung SPS	4000004 0160
• SPS Ansteuerung möglich	180
• Video verwalten	4000004 0179
▼ Datenbaustein auslesen	4000004 0180
• Auslesen der Datenbausteine	181
▼ Benutzerprogramm schreiben	4000004 0181
• Benutzerprogramm fertig	183
▼ Inbetriebnahme	ALARM_PREREC-INBETR
• SPS-Verbindung testen	4000004 0199
• Kameraverbindung testen	4000004 0201
• Hard- u. Software zusammenführen	4000004 0202
• Endprodukt testen	4000004 0230
▼ Prüfprotokoll ausstellen	4000004 0250
• Inbetriebnahme abgeschlossen	86
▼ Übergabe Kundens Schulung	ALARM_PREREC-ÜBERG.
• Hardware schulen	4000004 0260
• Software schulen	4000004 0270
▼ Projekt formell übergeben	4000004 0280
• Projekt übergeben	87

▼ Dokumentation	ALARM_PREREC-DOKU
• Kamera dokumentieren	4000004 0290
• Benutzerprogramm dokumentieren	4000004 0300
▼ Dokumentation übergeben	4000004 0310
• ♦ Projekt abgeschlossen	88

Abbildung 2: Teilschritte und Meilensteine

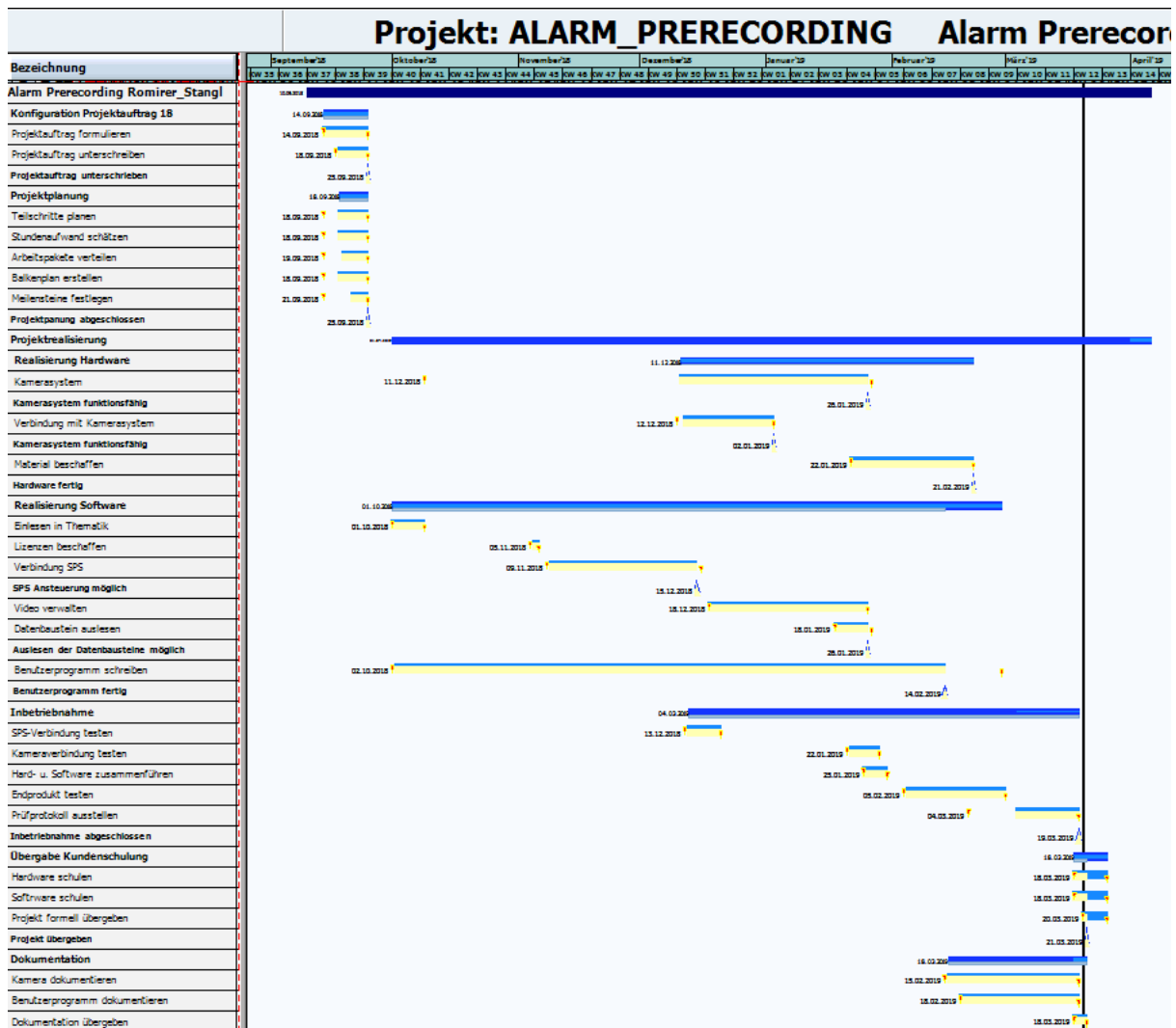


Abbildung 3: Balkenplan

S	U	S	Bezeichnung	Arbeit	Text 1
<input type="checkbox"/>			Alarm Prerecording Romirer_Stangl		
<input type="checkbox"/>		1	Konfiguration Projektauftrag 18		
<input type="checkbox"/>			Projektauftrag formulieren	1,0 H Stangl	
<input type="checkbox"/>			Projektauftrag unterschreiben	1,0 H Stangl & Romirer	
<input type="checkbox"/>			Projektauftrag unterschrieben		
<input type="checkbox"/>		1	Projektplanung		
<input type="checkbox"/>			Teilschritte planen	3,0 H Stangl	
<input type="checkbox"/>			Stundenaufwand schätzen	2,0 H Stangl	
<input type="checkbox"/>			Arbeitspakete verteilen	3,0 H Stangl	
<input type="checkbox"/>			Balkenplan erstellen	4,0 H Stangl	
<input type="checkbox"/>			Meilensteine festlegen	3,0 H Stangl	
<input type="checkbox"/>			Projektplanung abgeschlossen		
<input type="checkbox"/>		1	Projektrealisierung		
<input type="checkbox"/>		2	Realisierung Hardware		
<input type="checkbox"/>			Kamerasystem	80,0 H Stangl	
<input type="checkbox"/>			Kamerasystem funktionsfähig		
<input type="checkbox"/>			Verbindung mit Kamerasystem	27,0 H Stangl	
<input type="checkbox"/>			Kamerasystem funktionsfähig		
<input type="checkbox"/>			Material beschaffen	3,0 H Stangl	
<input type="checkbox"/>			Hardware fertig		
<input type="checkbox"/>		2	Realisierung Software		
<input type="checkbox"/>			Einlesen in Thematik	35,0 H Romirer und Stangl	
<input type="checkbox"/>			Lizenzen beschaffen	1,0 H Romirer und Stangl	
<input type="checkbox"/>			Verbindung SPS	26,0 H Romirer und Stangl	
<input type="checkbox"/>			SPS Ansteuerung möglich		
<input type="checkbox"/>			Video verwalten	7,0 H Romirer	
<input type="checkbox"/>			Datenbaustein auslesen	13,0 H Romirer	
<input type="checkbox"/>			Auslesen der Datenbausteine möglich		
<input type="checkbox"/>			Benutzerprogramm schreiben	120,0 H Romirer	
<input type="checkbox"/>			Benutzerprogramm fertig		
<input type="checkbox"/>		1	Inbetriebnahme		
<input type="checkbox"/>			SPS-Verbindung testen	4,0 H Romirer und Stangl	
<input type="checkbox"/>			Kamerverbindung testen	4,0 H Romirer und Stangl	
<input type="checkbox"/>			Hard- u. Software zusammenführen	10,0 H Romirer und Stangl	
<input type="checkbox"/>			Endprodukt testen	4,0 H Romirer und Stangl	
<input type="checkbox"/>			Prüfprotokoll ausstellen	6,0 H Romirer und Stangl	
<input type="checkbox"/>			Inbetriebnahme abgeschlossen		
<input type="checkbox"/>		1	Übergabe Kundens Schulung		
<input type="checkbox"/>			Hardware schulen	8,0 H	
<input type="checkbox"/>			Software schulen	8,0 H	
<input type="checkbox"/>			Projekt formell übergeben	4,0 H	
<input type="checkbox"/>			Projekt übergeben		

Abbildung 4: Stundenschätzung und Zuweisung in SAP

Diese Vorgänge wurden rückgemeldet und ein Soll/Ist – Abgleich der Kosten, siehe Abbildung 5 und der Stundenanzahl, siehe Abbildung 6, durchgeführt. Hier wurde auch das erstellte Material verbucht.

Ist/Plan/Abweichung		Stand: 20.03.2019		Seite: 2 / 5	
				Spalte: 1 / 3	
Objekt		PRO ALARM_PRERECORDING		Alarm Prerecording R	
Verantwortl. (Name)					
Von Geschäftsjahr		1900	Bis Geschäftsjahr	2019	
Von Periode		1	Bis Periode	12	
Kostenarten	Ist	Proj. Plan	Abw. abs.	Abw. %	
510000 Rohstoffverbrauch	70,00	70,00			
616000 Lohnkosten Montage	10.782,06	11.000,84	218,78-	1,99-	
* Alle Kostenarten	10.852,06	11.070,84	218,78-	1,98-	

Abbildung 5: Soll/Ist – Abweichung der Kosten

Ist/Plan/Abweichung		Stand: 20.03.2019		Seite: 4 / 5	
				Spalte: 3 / 3	
Objekt		PRO ALARM_PRERECORDING		Alarm Prerecording R	
Verantwortl. (Name)					
Von Geschäftsjahr		1900	Bis Geschäftsjahr	2019	
Von Periode		1	Bis Periode	12	
Kostenarten	Ist Menge	Plan Menge	Abw. abs.	Abw. %	
510000 Rohstoffverbrauch					
616000 Lohnkosten Montage	426,0	426,0			
* Alle Kostenarten	426,0	426,0			

Abbildung 6: Soll/Ist - Abweichung der Stundenanzahl

3 Projektrealisierung

3.1 Gesamtaufbau

Um ein Video mit flexibel, über ein Benutzerprogramm einstellbare, Zeiten aufnehmen zu können, wurde ein Kamerasystem entworfen, das sich aus einem Raspberry Pi und einer Webcam zusammensetzt.

Der dadurch entstehende Aufbau ist in Abbildung 7 ersichtlich.

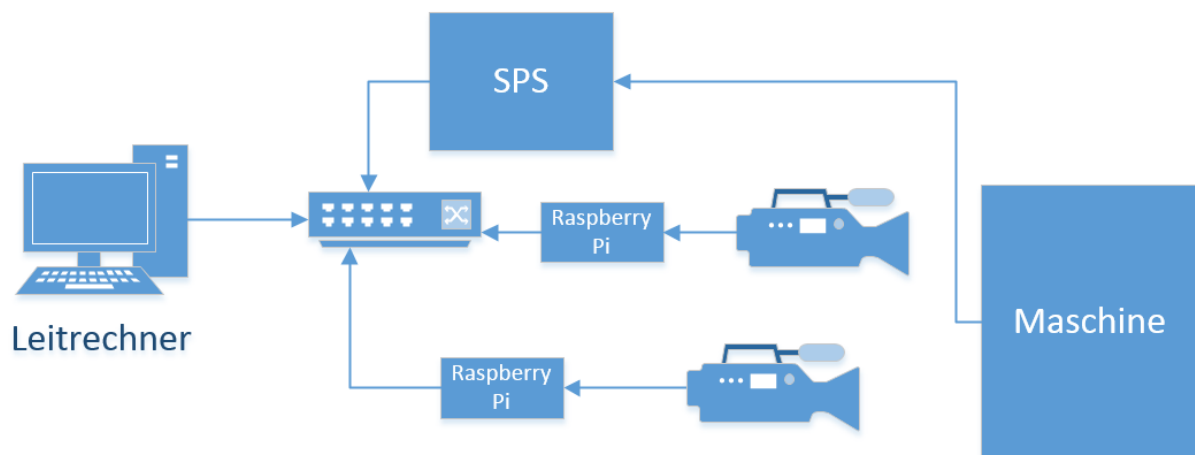


Abbildung 7: Aufbau

4 Auswahl des Kamerasystems

4.1 Alarm Pre-Recording Kamera

Bei der Auswahl des Kamerasystems wurden Ethernet-Kameras in Betracht gezogen. Die ersten Versuche wurden mit einer Alarm-Prerecording Network Kamera DS-2CD2125FWD-IS der Firma HikVision mit Alarmeingang durchgeführt. Die Kamera wird in diesem Fall über den Ethernet-Anschluss über POE mit Spannung versorgt und liefert ihre Daten über diesen Anschluss. Über den Alarmeingangspin kann eine Videoaufnahme getriggert werden.

Der Aufbau der Kamera ist in Abbildung 8 dargestellt.

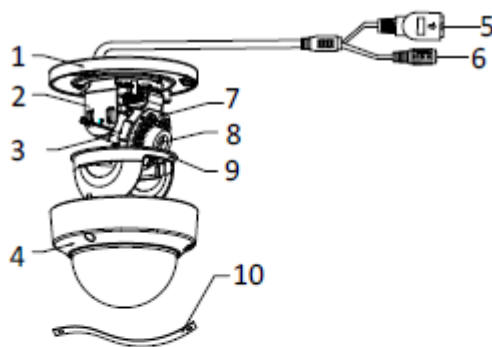


Figure 1-1 Overview of Type I Camera

Table 1-1 Description of Type I Camera

No.	Description	No.	Description
1	Mounting Base	2	Horizontal Stand
3	Vertical Stand	4	Bubble
5	Network Interface	6	Power Interface
7	IR LED	8	Lens
9	Black Liner	10	Safety Rope

Abbildung 8: Aufbau der Alarmkamera DS-2CD2125FWD-IS

Für die Testung der Kamera wurde der PC über Ethernet direkt mit der Kamera verbunden.

Über den Webbrowser konnte durch Eingabe der IP-Adresse der Kamera das Benutzerprogramm aufgerufen werden, siehe Abbildung 9.

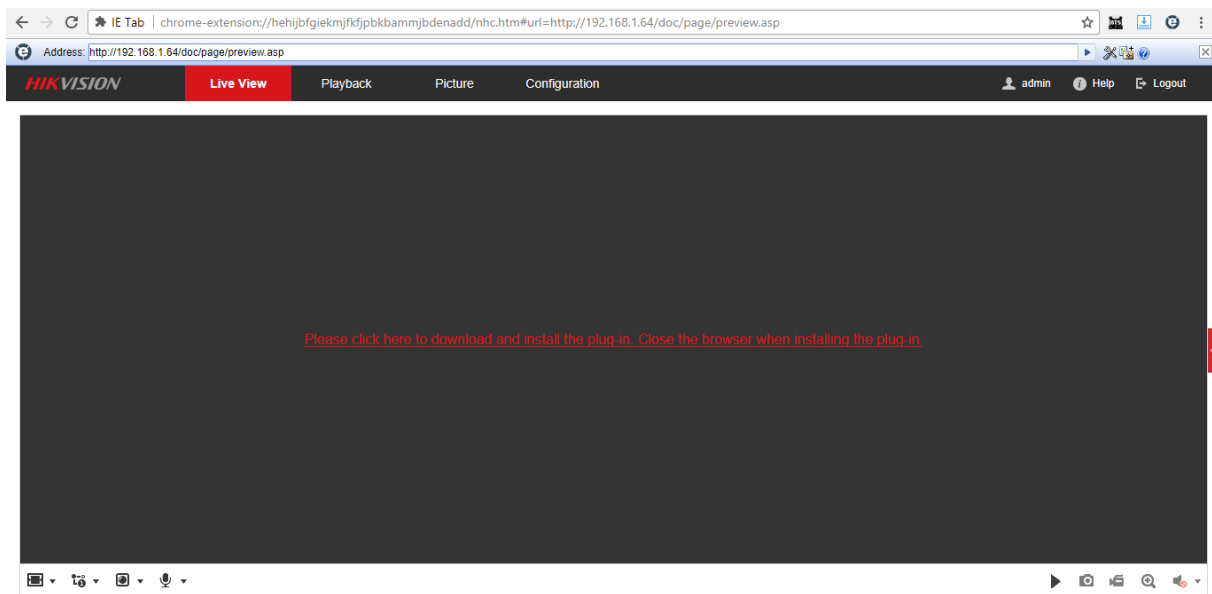


Abbildung 9: Das Benutzerprogramm kann über den Webbrowser aufgerufen werden.

Unter dem Menüpunkt „Live View“ kann das aktuelle Video der Kamera angesehen werden. Dieses Video kann gespeichert werden, zudem gibt es die Möglichkeit einzelne Bilder aufzunehmen.

Konfigurationen wie das verwendete Übertragungsprotokoll, das Bild-Format und der Pfad, in dem die Videos und Bilder gespeichert werden, können unter dem Menüpunkt „Configuration“ vorgenommen werden.

Unter dem Menüpunkt „Event Settings“ kann festgelegt werden, wann die Kamera ein Video aufnehmen soll. Der Alarmeingang kann hier noch genauer konfiguriert werden, ersichtlich in Abbildung 10.

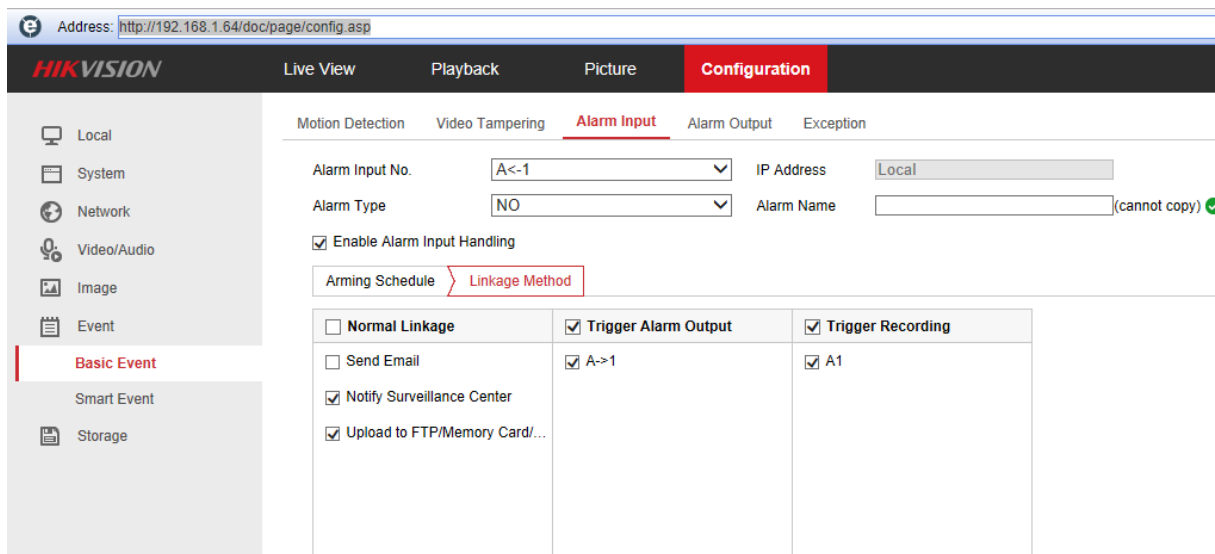


Abbildung 10: Eventsettings Webbrowser – Aufnahme Kamera1 wird bei Auslösen des Alarめingangs getriggert

Die Kameraeinstellungen können auch über eine Client-Software vorgenommen werden, siehe Abbildung 11. Diese wurde von der offiziellen Webseite der Firma HikVision heruntergeladen. (vgl. HikVision, 2018)

Auch in diesem Programm können die Events, die zum Auslösen einer Aufnahme führen sollen, konfiguriert werden.

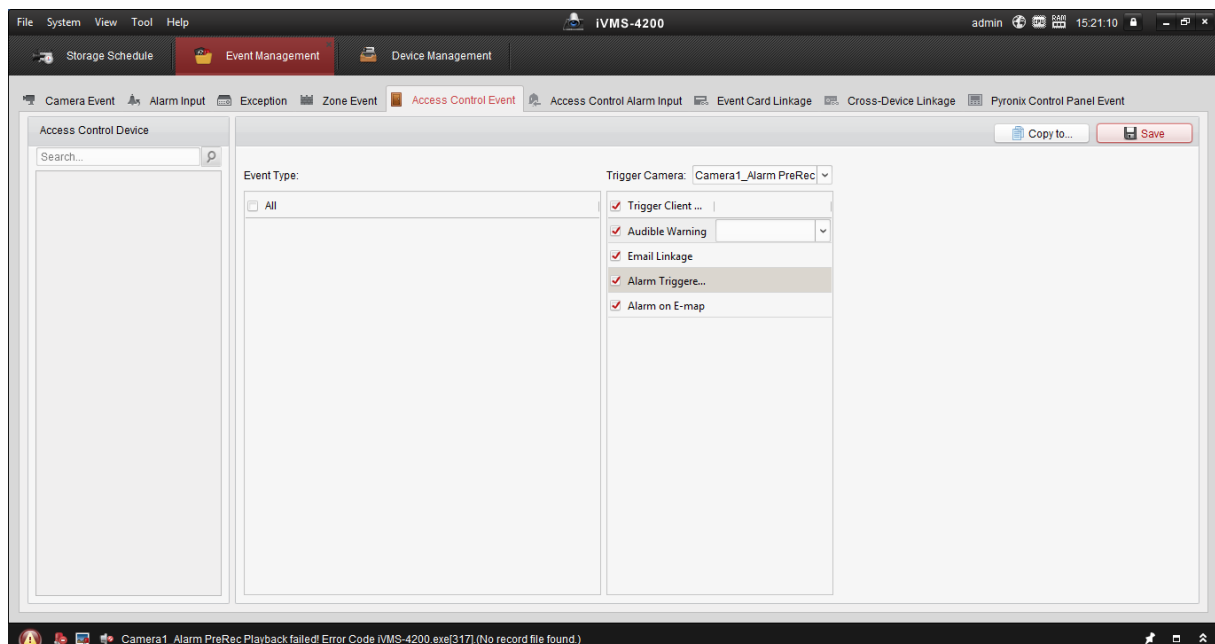


Abbildung 11: Eventsettings Benutzerprogramm

Der Alarmeingang kann zusätzlich konfiguriert werden. Hier können auch mehrere Kameras zur Aufnahme angegeben werden, siehe Abbildung 12.

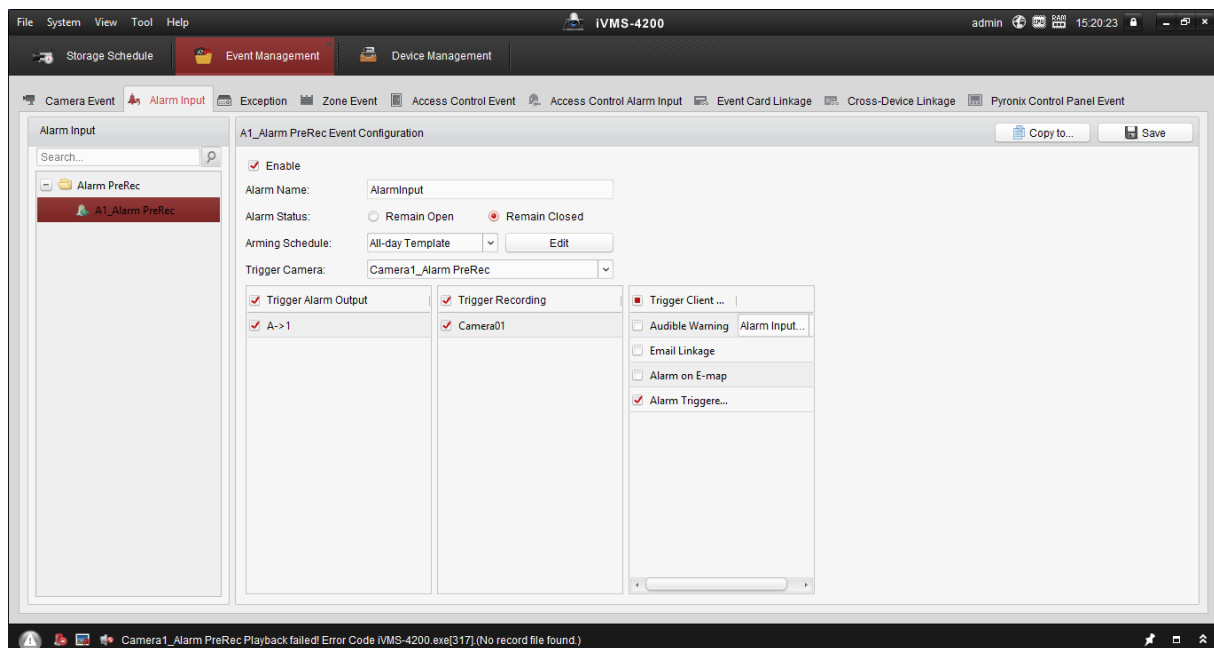


Abbildung 12: Konfigurationsmöglichkeiten

4.2 Ausschlusskriterien

Weitere Recherchen zeigten, dass die Kamera nur sekundlich Bilder an einen FTP-Server liefern kann. Diese macht die Verwendung der Kamera in dieser Aufgabenstellung unmöglich, da mit nur einem Bild pro Sekunde kein Video erzeugt werden kann.

Die Kamera bot zudem die Möglichkeit, eine Vorlaufzeit über das Benutzerprogramm einzustellen. Sie konnte für die gegebene Problemstellung jedoch nicht verwendet werden, da das Einstellen der Vor- und Nachlaufzeiten, sowie das Aktivieren der Kamera nur über das Benutzerprogramm der Kamerafirma bewerkstelligt werden kann. Dieses Problem zeigte sich auch mit anderen Ethernet-Kameras. Deshalb wurde für diese Diplomarbeit ein Kamerasystem bestehend aus einem Raspberry Pi mit USB-Kamera verwendet.

5 Kamerasystem mit Raspberry Pi und Webcam

Da Ethernet Kameras zur Umsetzung der Aufgabenstellung ungeeignet sind, wurde ein Kamerasystem, bestehend aus einem Raspberry Pi mit Webcam, verwendet.

Der Raspberry Pi ist ein Computer, siehe Abbildung 13, der in dieser Diplomarbeit dazu dient, die einzelnen Bilder der Kamera einzulesen und wird in der Programmiersprache C++ programmiert.

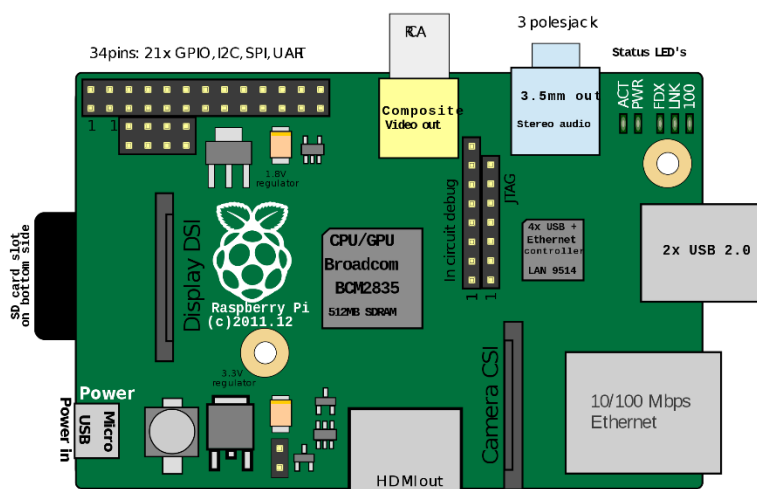


Abbildung 13: Aufbau Raspberry Pi

Für diese Bilder wurde softwaremäßig ein Ringpuffer implementiert. Bei Auftreten eines Fehlers speichert der Raspberry Pi die Daten des Puffers als Video ab, hierfür wird die Bibliothek OpenCV genutzt. Das Video wird auf dem Leitrechner gespeichert und verwaltet.

5.1 Kennwerte der WebCam

Die verwendete Kamera ist das Modell C920 HD Pro der Firma Logitech. Diese Webcam wird über USB verbunden und hat eine Frame-Rate von 30 fps. Die weiteren technischen Informationen können Abbildung 14 entnommen werden (vgl. Logitech Support, 2019).

Abbildung 15 zeigt diese Kamera.

Connection Type	USB
USB Protocol	USB 2.0
USB VID_PID	082D
UVC Support	Yes
Microphone	Yes
Microphone Type	Stereo
Lens and Sensor Type	Glass
Focus Type	Auto
Optical Resolution	True:3MP Software Enhanced:15MP
Diagonal Field of View (FOV)	78°
Horizontal Field of View (FOV)	70.42°
Vertical Field of View (FOV)	43.3°
Focal Length	3.67 mm
Image Capture (4:3 SD)	N/A
Image Capture (16:9 W)	2.0 MP, 3 MP*, 6 MP*, 15 MP*
Video Capture (4:3 SD)	N/A
Video Capture (16:9 W)	360p, 480p, 720p, 1080p
Frame Rate (max)	1080p@30fps

Abbildung 14: Technische Spezifikationen Webcam



Abbildung 15: C920 HD Pro Webcam

5.2 Verbindungseinstellungen

5.2.1 RaspberryPi-Verbindung zum Leitrechner

Um eine Verbindung zwischen dem Raspberry Pi und dem Leitrechner mit Ethernet zu ermöglichen, muss eine statische IP-Adresse vergeben werden. Mit dem Befehl „sudo nano /etc/dhcpd.conf“ wird eine Datei aufgerufen, in die die statische IP-Adresse eingetragen wird.

Damit diese Änderungen in Kraft treten, muss der Service neu gestartet werden. Die gewählte IP-Adresse wurde an den Adressraum der SPS angepasst. Wenn mehrere Kameras verbunden werden, müssen diese eine andere, noch nicht vergebene IP-Adresse im Adressraum 10.102.0.1 bis 10.102.255.254 besitzen.

5.2.2 Leitrechner-Verbindung zum Raspberry Pi

Zum Erstellen einer Verbindung zwischen Leitrechner und Raspberry Pi wurde die Bibliothek WinSCP (vgl. WinSCP, 2019) verwendet.

WinSCP bietet neben einer grafischen Oberfläche zum Austausch von Daten eine Bibliothek, die einfache Methoden zum Austausch von Daten mit dem RaspberryPi ermöglicht und eine gute Dokumentation bietet. Nach einigen Versuchen der Verbindungsherstellung und des Datenaustauschs mit verschiedenen Möglichkeiten wurde die Bibliothek WinSCP verwendet.

5.2.3 Putty Theorie und Verwendung

Putty stellt eine Verbindung mit dem Raspberry Pi her und wurde in dieser Diplomarbeit in der Testphase verwendet.

Um eine Verbindung mit dem Raspberry Pi herstellen zu können, muss die IP-Adresse angegeben werden. Mit der dann startenden Kommandozeile, siehe Abbildung 16, kann der Benutzername und das Passwort eingegeben werden. Nach erfolgreicher Anmeldung kann der Ordner des Kameraprogramms ausgewählt und das Kameraprogramm so manuell gestartet werden.

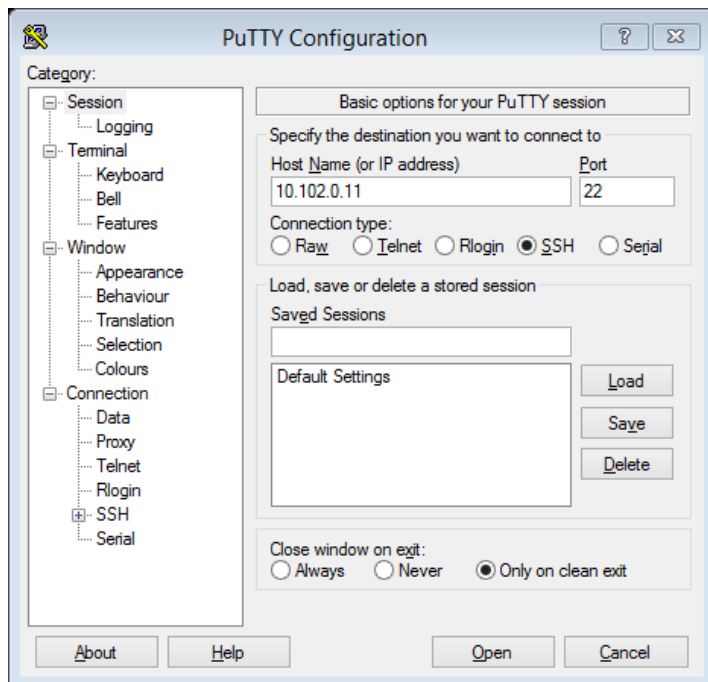


Abbildung 16: Putty – Anmeldung grafische Oberfläche

5.2.4 WinSCP Theorie

WinSCP bietet eine grafische Oberfläche und eine Bibliothek, die eine Verbindung zwischen zwei PCs ermöglicht.

Die grafische Oberfläche bietet die Möglichkeit, die Ordner des Raspberry Pis einzusehen und Dateien zu übertragen und herunterzuladen. Hierfür müssen zuerst die IP-Adresse des Raspberry Pis, der Benutzername und das Passwort angegeben werden, siehe Abbildung 17.

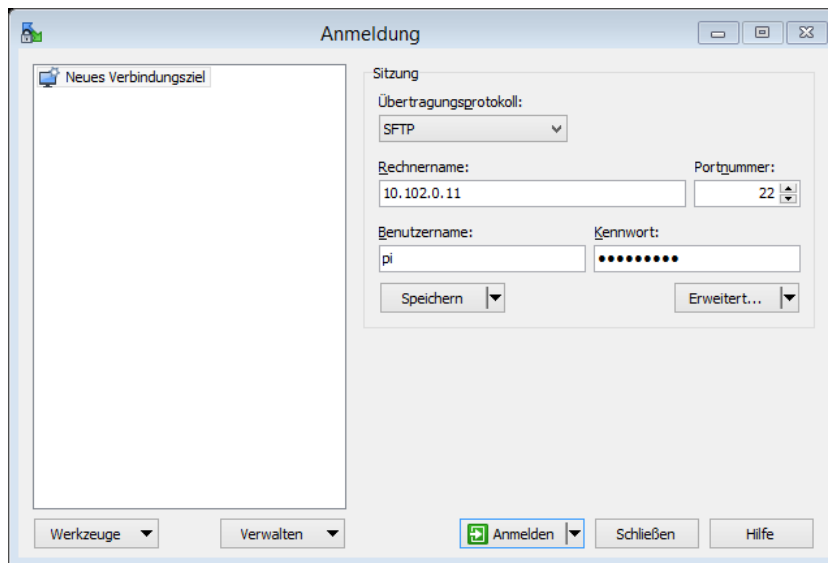


Abbildung 17: Anmeldung WinSCP

Wenn eine Verbindung hergestellt werden konnte, können die Ordner eingesehen und nach Belieben verändert werden, siehe Abbildung 18.

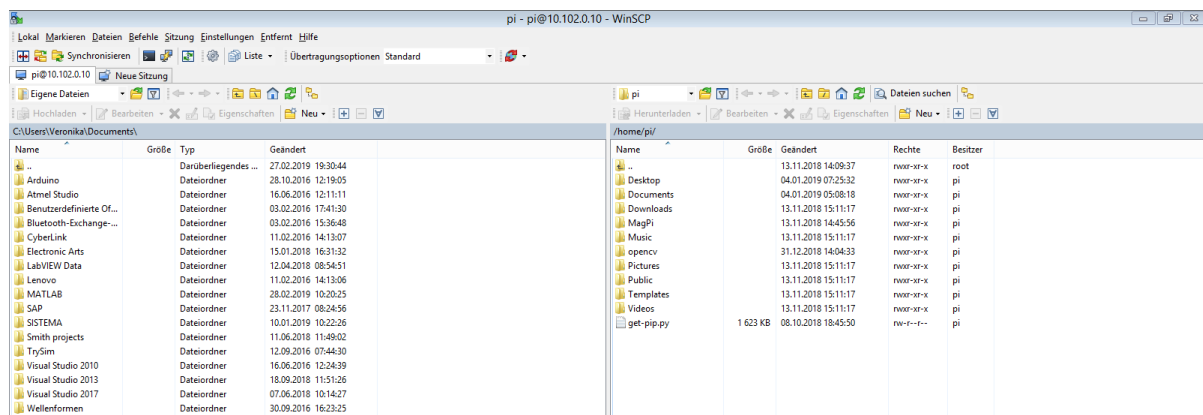


Abbildung 18: Ordneransicht WinSCP

WinSCP stellt eine C#-Bibliothek zur Verfügung, die es ermöglicht, eine Verbindung mit einem WinSCP-Client herzustellen. Sie erlaubt dem Nutzer/ der Nutzerin zudem, Daten zwischen diesen PCs auszutauschen. Um die Bibliothek im Programm nutzen zu können, muss die Bibliothek WinSCPnet als Verweis hinzugefügt werden. (vgl. WinSCP, 2019)

Die Klasse *SessionOptions* der Bibliothek enthält die notwendigen Parameter für eine Verbindung. Dieser Klasse müssen beim Erstellen eines Objektes das Protokoll, die

IP-Adresse des RaspberryPi, der Benutzername und das Passwort für den Zugang und der Zugangsschlüssel übergeben werden, siehe Abbildung 19.

```
public static SessionOptions options = new SessionOptions
{
    Protocol = Protocol.Scp,           //Protokoll für Kommunikation mit RaspberryPi ist SCP
    HostName = "10.102.0.9",          //IP-Adresse des Pis
    Password = "Raspberry",           //Passwort
    UserName = "pi",                  //Username
    GiveUpSecurityAndAcceptAnySshHostKey = true
};
```

Abbildung 19: SessionOptions

Die Klasse *Session* repräsentiert die Verbindung und enthält Methoden für den Austausch der Daten.

Um also eine funktionsfähige Verbindung zu schaffen, musste zuerst ein Objekt der Klasse *SessionOptions* erzeugt werden, dem die notwendigen Parameter übergeben werden. Danach wurde eine neue *Session* erzeugt. Bei Aufruf der Methode *ConnectCamera()* wird mit der Methode *Open()* der *Session*-Klasse eine Verbindung erstellt. Ob diese Verbindung richtig erstellt wurde, kann mit der Variable *Opened* überprüft werden.

Um eine Übertragung der Daten zu ermöglichen muss ein Objekt der Klasse *TransferOptions* erzeugt werden. Der benötigte *TransferMode* ist *Binary*. Um zu überprüfen ob die Übertragung funktioniert, wird ein Objekt der Klasse *TransferOperationResult* benötigt.

Mit der Methode *PutFiles* der *Session*-Klasse können Daten vom Leitrechner auf den Raspberry übergeben werden. Die Methode *GetFiles* ermöglicht den Download von Raspberry-Daten.

PutFiles benötigt den lokalen Datenpfad, den Pfad für den RaspberryPi, den Bool-Wert *false*, um die Daten zu kopieren, nicht auszuschneiden und die *TransferOptions* für eine Übertragung. Der Rückgabewert ist *TransferOptionsResult*.

Ob die Übertragung funktioniert, kann mit der Methode *TransferOptionsResult.Check()* überprüft werden. Bei einem Fehler wird eine *Exception* geworfen. Abbildung 20 zeigt die Codesequenz zum Übertragen von Daten.

```
TransferOptions transferOptionsUpload = new TransferOptions();
transferOptionsUpload.TransferMode = TransferMode.Binary;           //Binärübertragung

TransferOperationResult transferResultUpload;
transferResultUpload = session1.PutFiles(@"C:\Users\Veronika\Desktop\SendRequestToStoreVideo.csv",
    "/home/pi/Documents/FileEditing/AufnehmenMitPolling/", false, transferOptionsUpload);
transferResultUpload.Check();                                       //wenn nicht übertragen - Error
```

Abbildung 20: Codesequenz zum Übertragen von Daten

Die Methode *Session.GetFiles()* benötigt als Parameter zuerst den Datenpfad des RaspberryPi, dann den gewünschten Pfad am Leitrechner, anschließend wieder den Wert *false*, um die Daten zu kopieren, und ein Objekt der Klasse *TransferOptions*. Auch hier ist der Rückgabewert der Datentyp *TransferOperationResult*. Das Überprüfen ob die Übertragung erfolgreich war, erfolgt wie bei der Methode *PutFiles*.

Abbildung 21 zeigt die Codesequenz zum Herunterladen von Daten.

```
TransferOptions transferOptionsDownload = new TransferOptions();
transferOptionsDownload.TransferMode = TransferMode.Binary;

TransferOperationResult transferResultDownload;
transferResultDownload = session1.GetFiles("/home/pi/Documents/FileEditing/AufnehmenMitPolling/out.avi",
    @"C:\Users\Veronika\Desktop\", true, transferOptionsDownload);

transferResultDownload.Check();                                     //wenn nicht übertragen - Error
```

Abbildung 21: Herunterladen von Daten

Um die Verbindung mit dem RaspberryPi zu schließen wird die Methode *Close()* genutzt.

5.3 OpenCV Theorie

Die OpenCV-Bibliothek (vgl. OpenCV, 2019) ist eine freie Bibliothek, die in den Programmiersprachen C/C++, Java und Python genutzt werden kann. Sie bietet Algorithmen und Datentypen für die Bildverarbeitung. In diesem Fall wurde sie mit der Programmiersprache C++ und für die Interaktion mit der Kamera und das Speichern des Videos verwendet. Die Anwendung im Kameraprogramm und die dort verwendeten Klassen und Methoden werden im Kapitel 5.4 erläutert.

5.3.1 Benötigte Installationen

Benötigte Bibliotheken mussten installiert werden, hierfür wurde der folgende Befehl benutzt:

```
„sudo apt-get install build-essential git cmake pkg-config libjpeg-dev libtiff5-dev  
libjasper-dev libpng-dev libavcodec-dev libavformat-dev libswscale-dev libv4l-dev  
libgtk2.0-dev libatlas-base-dev gfortran libxvidcore-dev libx264-dev“
```

Um OpenCV installieren und kompilieren zu können, muss Python genutzt und zuvor installiert werden.

```
„sudo apt-get install python2.7-dev“
```

Ein weiteres Paket, das zur Installation benötigt wird, ist *pip*, ein Paket-Management-Tool und *numpy*, eine Bibliothek für Array-Operationen mit Python-Bindung. Die Befehle hierfür sind:

```
“cd ~ && wget https://bootstrap.pypa.io/get-pip.py && sudo python get-pip.py” und  
„pip install numpy“
```

5.3.2 Konfigurationen

OpenCV wurde mit von *git*, einer Software zur Versionsverwaltung von Dateien, heruntergeladen:

```
“git clone https://github.com/Itseez/opencv.git && cd opencv && git checkout 3.0.0”
```

Zum Kompilieren der OpenCV-Bibliothek wird ein Ordner erstellt:

```
“cd ~/opencv && mkdir build && cd build”
```

Bevor kompiliert werden kann, muss der Vorgang konfiguriert werden:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D INSTALL_C_EXAMPLES=ON \  
-D ENABLE_PRECOMPILED_HEADERS=OFF \  
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \  
-D BUILD_EXAMPLES=ON ..
```

Mit den Befehlen „make -j4“ und „sudo apt-get install build-essential“ konnte OpenCV nun kompiliert und installiert werden.

Um OpenCV in ein Programm einbinden zu können muss folgendes Linker-Flag gesetzt werden: ``pkg-config --cflags --libs opencv``

Eine weitere Möglichkeit des Kompilierens ist die Nutzung der Kommandozeile. Hier muss der Ordner ausgewählt werden, in dem sich das zu kompilierende Programm befindet:

„cd home/pi/Documents/FileEditing/AufnehmenMitPolling/“

Anschließendes Kompilieren der Datei AufnPoll.cpp mit dem Befehl:

„g++ AufnPoll.cpp -o main `pkg-config opencv --cflags --libs`“

Das daraus resultierende *Executable* kann mit „./main“ ausgeführt werden.

5.4 Kameraprogramm

Der Ablauf des Programms ist im Flussdiagramm in Abbildung 22 ersichtlich.

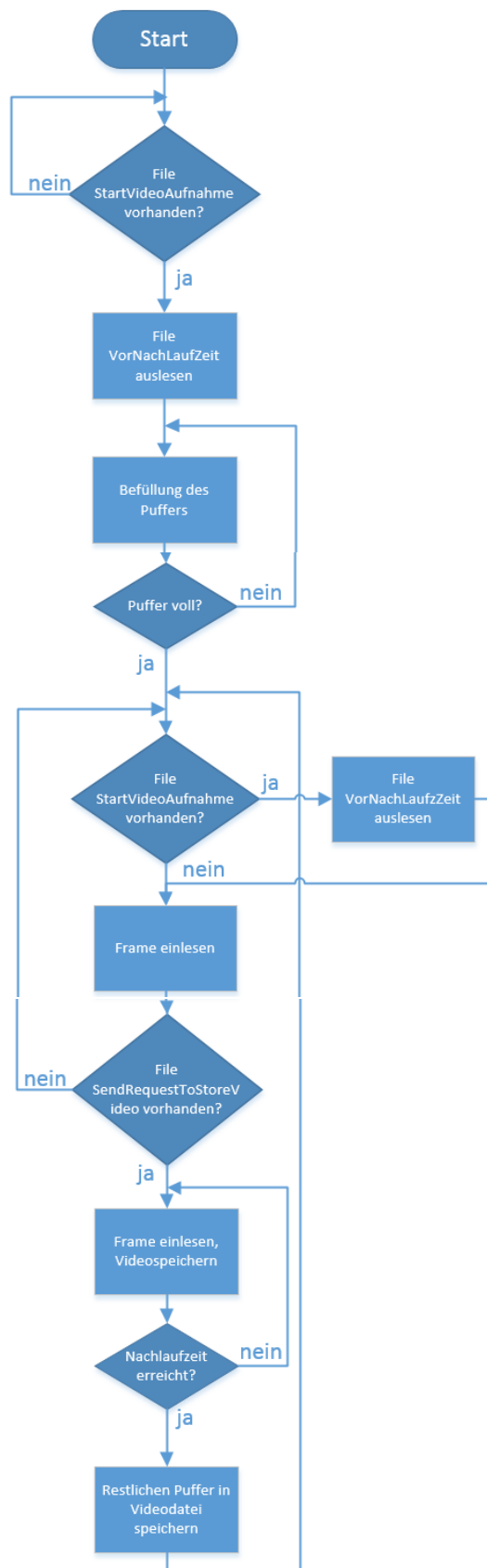


Abbildung 22: Flussdiagramm Kameraprogramm

5.4.1 CSV-File auslesen

Um die Daten der Vor- und Nachlaufzeit auszulesen, musste ein CSV-File-Reader implementiert werden. Dieser wurde mit einem variabel großen, zweidimensionalen Integer-Vektor als Rückgabeparameter realisiert. Der Funktion wird der Name des auszulesenden Files übergeben.

Mit zweier *while-Schleifen* werden die Daten ausgelesen und ans Ende des Datenfeldes angefügt. (vgl. Water Programming, 2019)

```
ifstream files(filename);

while(files)
{
    string s;
    if(!getline(files, s)) break;

    istream ss(s);
    vector<int> record;

    while(ss)
    {
        string line;
        if(!getline(ss, line, ',')) break;
        try
        {
            record.push_back(stof(line));
        }
        catch(const invalid_argument e)
        {
        }
    }
    data.push_back(record);
}
```

Abbildung 23: CSV-File auslesen

5.4.2 File-Polling

Für das File-Polling, dem Kontrollieren ob eine Datei vorhanden ist, wird der Datentyp *struct stat* genutzt. Dieser soll nach dem Systemaufruf *stat()* Fileattribute zurückliefern. Der Rückgabewert der Funktion beträgt 0, wenn sie erfolgreich war. Dem Aufruf wird der Name und Pfad der Datei übergeben, sowie die struct stat Variable als Referenz, siehe Abbildung 24. (vgl. Raspberry Projects, 2019)

```
do
{
    resultVorNach = stat("/home/pi/Documents/FileEditing/AufnehmenMitPolling/StartVideoAufnahme.txt",
        &sbVorNachLauf);           //Abfrage File vorhanden
}while((resultVorNach!=0)|| (sbVorNachLauf.st_mode & S_IFDIR)); //wenn File nicht hier - wiederholen
```

Abbildung 24: FilePolling

Mit der Methode *unlink()* des Namespace Standard wird ein File gelöscht, ihr wird der Dateipfad und Name übergeben, siehe Abbildung 25.

```
unlink("/home/pi/Documents/FileEditing/AufnehmenMitPolling/FirstRead.txt");
```

Abbildung 25: Löschen eines Files

5.4.3 File erzeugen

Mit der Klasse *ofstream* (Output file stream) können Files erzeugt und bearbeitet werden. Mit der Methode *open()* kann ein bereits bestehendes File geöffnet, beziehungsweise ein neues File erzeugt, werden. Die Methode *close()* schließt das File wieder. Abbildung 26 zeigt die resultierende Codesequenz.

```
ofstream firstReadFile;
firstReadFile.open("/home/pi/Documents/FileEditing/AufnehmenMitPolling/FirstRead.txt");
firstReadFile.close();
```

Abbildung 26: File erzeugen und schließen

Dies wird im Programm genutzt, um dem Leitrechner mitzuteilen, in welchem Bearbeitungsschritt das Kamerasystem sich befindet. Hierfür gibt es das File „*ReadingData*“, das existiert, während das CSV-File mit der Vor- und Nachlauzeit ausgelesen wird. Bei Verlassen der Funktion wird das File wieder gelöscht. Das File „*started*“ wird erzeugt, wenn das Programm startet. „*FirstRead*“ gibt es solange der Puffer der Kamera das erste Mal befüllt wird. In dieser Zeit kann die Kamera noch kein Video speichern, da noch nicht genügend Bilder aufgenommen wurden. „*WritingVideo*“ wird erzeugt, wenn das Programm beginnt, das aufgenommene Video zu speichern. Nach Beendigung dieses Vorgangs wird dieses File gelöscht und „*VideoFinished*“ erzeugt. Sollte es zu einem Fehler im Programm kommen, wird das File „*Error*“ erzeugt.

5.4.4 Kamera und Video

Um das Bild einer Kamera einzulesen, wird ein Objekt der Klasse *VideoCapture* erzeugt.

Diese Klasse bietet Methoden, um Bilder einer Kamera oder bestehende Videos auszugeben und einzulesen. Mit der Methode *get()* können Eigenschaften des Objektes in Erfahrung gebracht werden. Dies wird über Übergabeparameter bewerkstelligt. Mit dem Schlüsselwort *CV_CAP_PROP_FRAME_WIDTH* wird die Breite des von der Kamera aufgenommenen Bildes zurückgegeben, *CV_CAP_PROP_FRAME_HEIGHT* gibt die Höhe zurück. *CV_CAP_PROP_FPS* gibt die Anzahl der Frames (Bilder) pro Sekunde zurück. Diese Werte werden als Integer-Wert zurückgegeben.

```
short int frame_width = cap.get(CV_CAP_PROP_FRAME_WIDTH);    //Breite des Fensters
short int frame_height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);  //Höhe des Fensters
short int FPS = cap.get(CV_CAP_PROP_FPS);                    //FPS-Rate
```

Abbildung 27: Kamera-spezifische Werte auslesen

Ein Objekt der Klasse *Mat* erzeugt ein Array, das diverse Datentypen annehmen kann. Es wird genutzt, um ein Bild einzulesen und auch für das Puffer-Array, welches als Ringpuffer verwendet wird. Dieser Puffer bietet Platz für ein 60 Sekunden langes Video. Diese Größe wird durch die Multiplikation von der 60 Sekunden mit der Anzahl der Bilder pro Sekunde berechnet, siehe Abbildung 28.

```
Mat buffer[FPS*60];    //Buffer-Array für Gesamtzeit
```

Abbildung 28: Puffer-Array

copyTo() kopiert den Inhalt des *Mat*-Arrays in ein anderes Array. Dieses neue Array wird als Übergabeparameter übergeben.

```
cameraFrame.copyTo(buffer[frameNumber]);    //Speichern des Frames in Buffer
```

Abbildung 29: Methode *copyTo()*

Die Klasse *VideoWriter* ermöglicht es, ein Video zu speichern. Dem Konstruktor eines Objekts dieser Klasse wird der Speicherpfad und Filename des Videos, der zu verwendende Codec, die Anzahl der Frames pro Sekunde, sowie die Größe des

Videos übergeben. Zu Beginn wird der *VideoWriter* initialisiert, mit der Methode *open()* wird er geöffnet und so ein *out.avi*-File erzeugt.

Mit der Methode *write()* oder dem Operator *<<* wird der jeweilige Frame gespeichert. Die Methode *release()* schließt den *VideoWriter*.

Der Puffer wurde für die Länge der Vorlaufzeit ausgelegt. Hierfür wurde der *VideoWriter* bereits am Beginn des Programms definiert. Wenn ein Video gespeichert werden soll, wird der *VideoWriter* mit der Methode *open()* geöffnet. Wenn ein Fehler auftritt, wird ein Frame gespeichert und im nächsten Schritt mit dem nächsten Frame überschrieben. Dies wird so lange durchgeführt, bis das Ende der Nachlaufzeit erreicht wurde. Dann wird der Rest der im Puffer gespeicherten Frames in die Videodatei geschrieben. Diese Implementierung ist in Abbildung 30 ersichtlich.

```
out.open("/home/pi/Documents/FileEditing/AufnehmenMitPolling/out.avi",
        CV_FOURCC('M', 'J', 'P', 'G'), FPS, Size(frame_width*0.75, frame_height*0.75));
case ST_FILEEXISTS:
    framesNachLauf++;
    out << buffer[frameNumber];
    if(framesNachLauf == nachLaufZeit)
    {
        for(int i=0; i<vorLaufZeit; i++)
        {
            frameNumber++;
            out << buffer[frameNumber];
            if(frameNumber == vorLaufZeit)
            {
                frameNumber = 0;
            }
        }
        out.release();
    }
```

Abbildung 30: Video erstellen

Diese Sequenz wurde erweitert. Die Methode *resize()* der OpenCV-Bibliothek verkleinert das eingelesene Bild. Hierfür werden dieser Methode das zu verkleinernde Bild, die neue Variable und der Verkleinerungsfaktor übergeben, siehe Abbildung 31.

```
cap >> cameraFrame;
resize(cameraFrame, cameraFrame, Size(), 0.75, 0.75);
```

Abbildung 31: Methode *resize()*

In dieser Codesequenz wird das Bild auf jeder Seite um ein Viertel verkleinert. Durch diese Verkleinerung kann ein Video mit einer Vorlaufzeit von bis zu 60 Sekunden aufgenommen werden.

5.5 Autostart

Um ein Programm auf dem Raspberry Pi automatisch starten zu lassen, kommen mehrere Varianten in Frage. (vgl. Raspberry Tips, 2019)

Die erste Methode, das Erstellen einer Servicedatei, stützt sich auf *systemd*. Es ist ein Hintergrundprogramm, das Prozesse verwaltet. Hierfür wurde die Datei „*kameraservice.service*“ erstellt. Sie wurde in das Verzeichnis */etc/systemd/system/* gespeichert.

In diese Datei werden Informationen über den Service und das auszuführende Programm eingetragen. Die Struktur ist in Abbildung 32 ersichtlich.

```
#!/bin/bash
[Unit]
Description=My service
After=network.target

[Service]
Type=simple
ExecStart=/usr/local/bin/AufnPoll
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10
User=pi

[Install]
WantedBy=multi-user.target
```

Abbildung 32: Servicedatei

Im Unit-Sektor werden Informationen über den Service spezifiziert. „*Description*“ ist der Name des Services. „*After*“ gibt an, wann der Service gestartet wird.

Im Service-Sektor werden Informationen zum Programm angegeben. „*Type*“ spezifiziert die Art des Prozesses. In diesem Fall „*simple*“, ein einfacher, linearer Prozess, der keine speziellen Ressourcen benötigt. „*ExecStart*“ gibt den auszuführenden Befehl, den Aufruf der kompilierten Datei, an. Wenn das Programm beendet wird, soll es neu gestartet werden. Dies wird durch „*Restart=always*“ angegeben. Bevor es jedoch neu gestartet wird, sollen zehn Sekunden vergehen, angegeben durch „*RestartSec=10*“. Dies hat den Vorteil, dass dem Raspberry Pi Zeit gegeben wird den vorigen Prozess zu beenden. Der

Benutzername wird in „*User=pi*“ spezifiziert. Im Install-Sektor werden weitere Informationen zum Service angegeben.

Einfacher war es, das Programm in der Datei */etc/rc.local* einzutragen. Hier mussten der Pfad und der Dateiname des *Executables* (kompilierte C++-Datei) eingetragen werden.

Bei dieser Variante konnte jedoch eine, für OpenCV benötigte, Bibliothek nicht gefunden werden. Mit dem Befehl „*sudo nano /etc/ld.so.conf.d/opencv.conf*“ wurde eine Datei erstellt, in der der Pfad der Bibliothek eingetragen wurde. Mit dem Befehl „*sudo ldconfig -v*“ wird diese Datei bekanntgemacht. (vgl. Github, 2019)

6 AGLINK4 – SPS Verbindung

AGLink ist eine Kommunikationsbibliothek, mit der eine Verbindung mit einer SPS hergestellt werden kann. Es bietet zudem einige Unterprogramme, die die Nutzung der Bibliothek vereinfachen sollen und eine Verbindung mit der SPS herstellen können. Die Lizenz, die für die Verwendung dieser Bibliothek benötigt wird, wurde vom Auftraggeber als USB-Dongle zur Verfügung gestellt.

Der prinzipielle Kommunikationsablauf mit einer SPS ist in Abbildung 33 dargestellt.

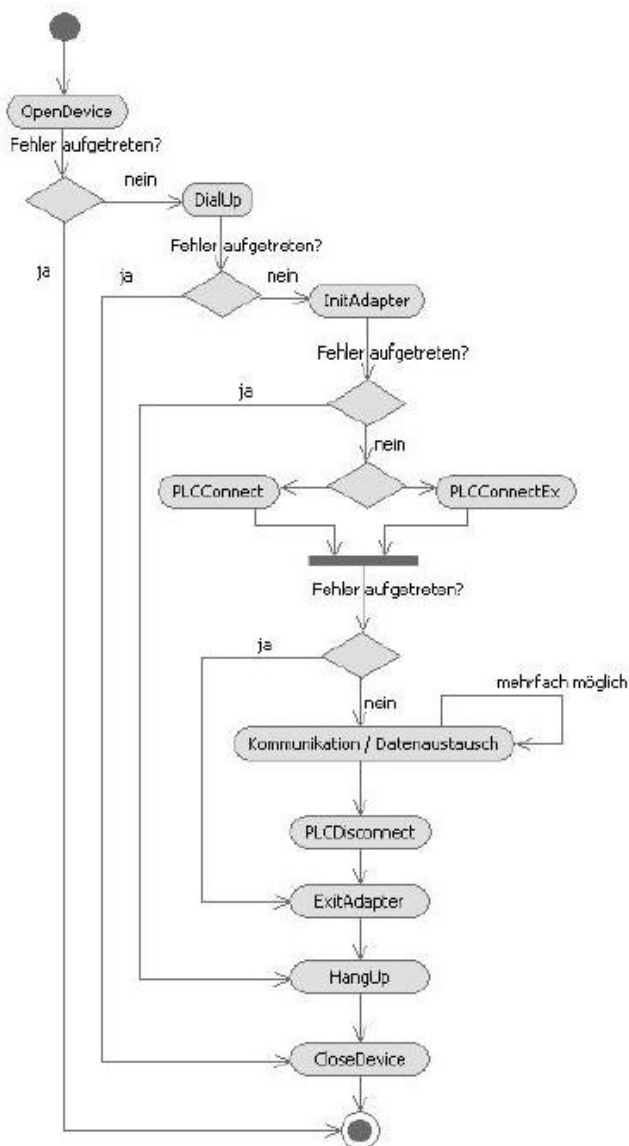


Abbildung 33: Flussdiagramm der Kommunikation mit der SPS

Der ACCON AGLink API-Guide ist ein Unterprogramm, das verschiedene Funktionen der AGLink-Bibliothek anzeigt und den dazugehörigen Code ausgibt.

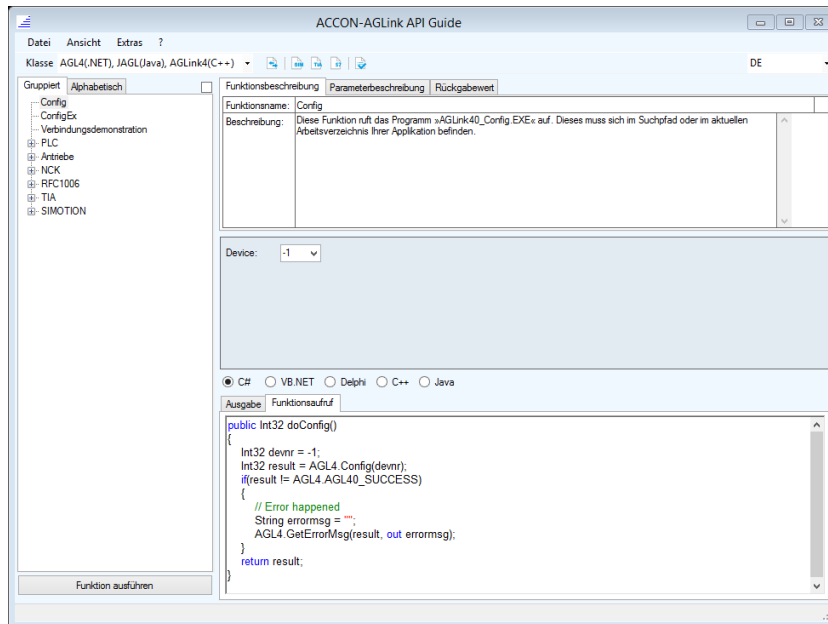


Abbildung 34: AGLink API-Guide

Das Unterprogramm AGLink40_Config.exe wird zur Konfiguration der SPS-Verbindung verwendet. Hier kann das Kommunikationsprotokoll und ein Verbindungs-Timeout angegeben werden, siehe Abbildung 35. Das Verbindungs-Timeout legt fest, wie lange auf die Antwort der SPS gewartet wird.

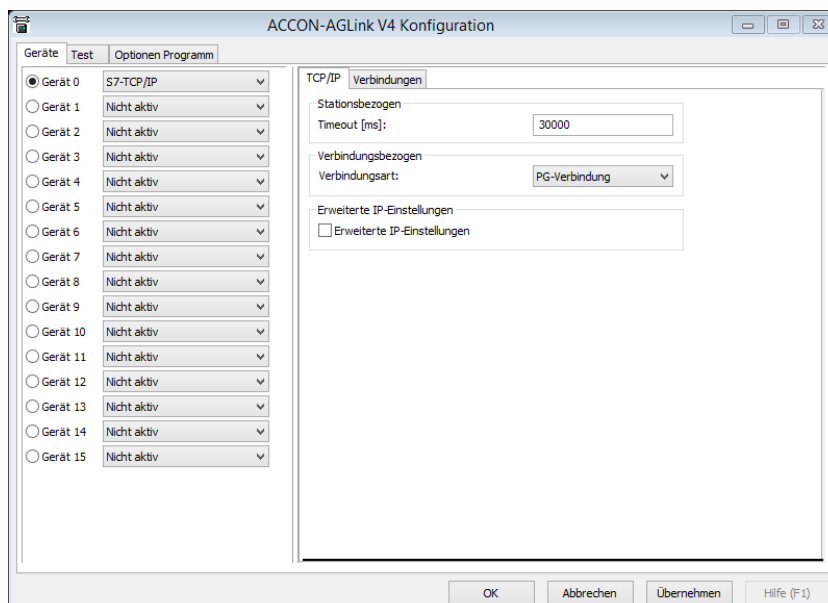


Abbildung 35: SPS-Verbindungskonfigurator

Das Programm kann zudem zur Testung der Verbindung verwendet werden, siehe Abbildung 36.

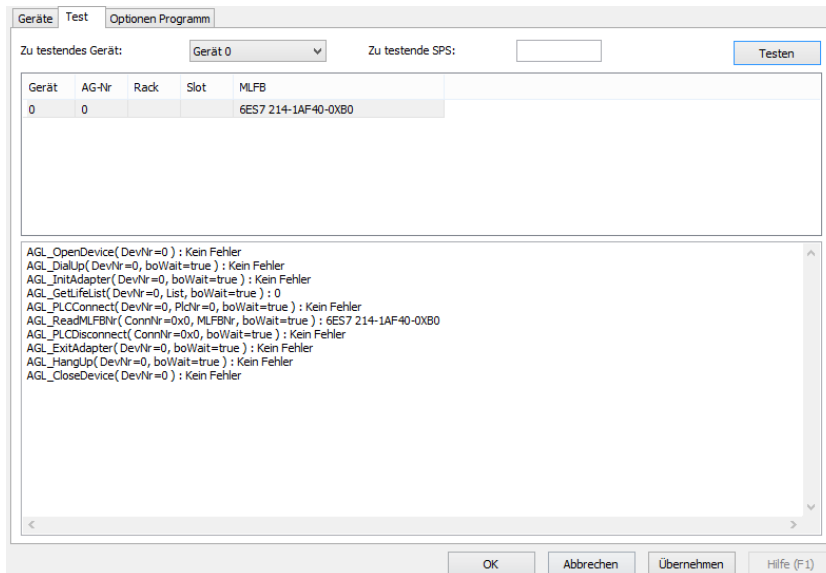


Abbildung 36: Verbindungstest Verbindungskonfigurator

6.1 AGLINK Bibliothek

Die Kommunikationsbibliothek AGLink sorgt für eine Verbindung zwischen Leitrechner und SPS. Die Bibliothek bietet Funktionen zum Datenaustausch zwischen diesen Komponenten, sowie Möglichkeiten zur Konvertierung von Datentypen.

6.2 SPS Verbindung

Um eine neue SPS-Verbindung im Programm herstellen zu können, muss zuerst eine Instanz erstellt werden. Die Methode *CreateTiaInstance* erstellt eine Instanz, die die Grundlage für die Verbindung darstellt. Die hierfür benötigte Variable der Klasse *IAGLink4Tia* wird zuvor mit null initialisiert.

IAGLink4Tia Verbindungsaufbau zum symbolischen Zugriff auf S7-1200/S7-1500, die mit TIA-Portal projektiert wurden

Funktion:

IAGLink4Tia AGL4ConnectionFactory.CreateTiaInstance(Int32 devNr, Int32 plcNr, int timeout)

Parameter:

devNr	Nummer des Devices
plcNr	AG-Nummer
timeout	Timeout

Rückgabewert:

AGLink4-Schnittstelle, mit angegebenen Daten initialisiert

Anschließend wird mit der Methode *ConfigEx* der Verbindungskonfigurator aufgerufen. Der Pfad zum Speichern der resultierenden Datei die die Konfiguration enthält wird zuvor mit der Methode *SetParaPath* festgelegt.

Funktion:

Int32 AGL4.ConfigEx(Int32 devNr, path)

Parameter:

devNr	Nummer des Devices -1 für alle Devices
path	Projektpfad

Rückgabewert:

>= 0	bei erfolgreicher Verbindung AGL40_SUCCESS
< 0	Errornummer

Funktion:

AGL4.SetParaPath(path)

Parameter:

Path	Projektpfad
------	-------------

Der Rückgabeparameter der Funktion *ConfigEx* ist vom Typ Int32 und liefert bei erfolgreicher Herstellung der Verbindung den Rückgabewert AGL40_SUCCESS.

Wenn diese Schritte erfolgreich durchgeführt wurden, kann eine Verbindung hergestellt werden. Dies wird mit der Funktion *Connect* bewerkstelligt. Ihr Rückgabewert ist ein Bool-Wert, der true ist, wenn die Verbindung erfolgreich aufgebaut wurde.

6.3 SPS Datenbausteine auslesen

Zum Auslesen der Datenbausteine wurde die Funktion *ReadMix* verwendet. Diese Funktion erlaubt es, einzelne Bits auszulesen.

Funktion:

Int32 AGL4.ReadMix(int ConnNr, AGL4.DATA_RW40 rwfield, int num, int timeout)

Parameter:

ConnNr	Variable für den Verbindungshandler
rwfield	Indikator für die Lesestructur
num	Anzahl der Strukturen
timeout	Timeout

Rückgabewert:

≥ 0	bei erfolgreicher synchroner Verbindung AGL40_SUCCESS
< 0	Errornummer

Der Funktion wird eine Variable für die Verbindung übergeben, sowie ein zuvor initialisiertes Pufferfeld, die Länge dieses Feldes sowie ein Timeout.

7 Benutzerprogramm – Alarm Pre- Recording - Einstellungen

7.1 Einführung

In Rahmen dieser Diplomarbeit sollte ein Windowsprogramm erstellt werden, das die Abarbeitung der Videoaufnahme beinhaltet. Um die vom Auftraggeber festgelegten flexiblen Einstellungen treffen zu können, müssen diese Einstellungen auf einer grafischen Oberfläche angezeigt werden. Ein leichtes Abarbeiten der Einstellungen durch den einfachen Benutzer / die Benutzerin soll ermöglicht und somit ein serielles Abarbeiten implementiert werden.

Die verwendete grafische Oberfläche basiert auf Windows Formen, folglich als Formen bezeichnet und wird in der Programmiersprache C# implementiert.

Laut Microsoft beinhalte der Namespace *System.Windows.Forms* Klassen zur Erstellung Windowsbasierter Anwendungen, (vgl. Microsoft - Windows Forms, 2019)

7.1.1 Flussdiagramm – Visio

Ziel dieser grafischen Oberfläche ist es, Einstellungen des Kamerasystems festzulegen und diese an das jeweilige System weiterzuleiten. Das gesamte Programm unterteilt sich in einen Konfigurationsmodus und einen automatisierten Modus. Der Konfigurationsmodus stellt die Kamerawerte ein und der automatisierte Modus verteilt die Arbeitsschritte selbst an das Kamerasystem. Das folgende Flussdiagramm zeigt diesen Ablauf, Abbildung 37. Dieses Flussdiagramm beinhaltet eine einfache Form des automatisierten Ablaufes, dieser wird näher in Abbildung 38 erklärt.

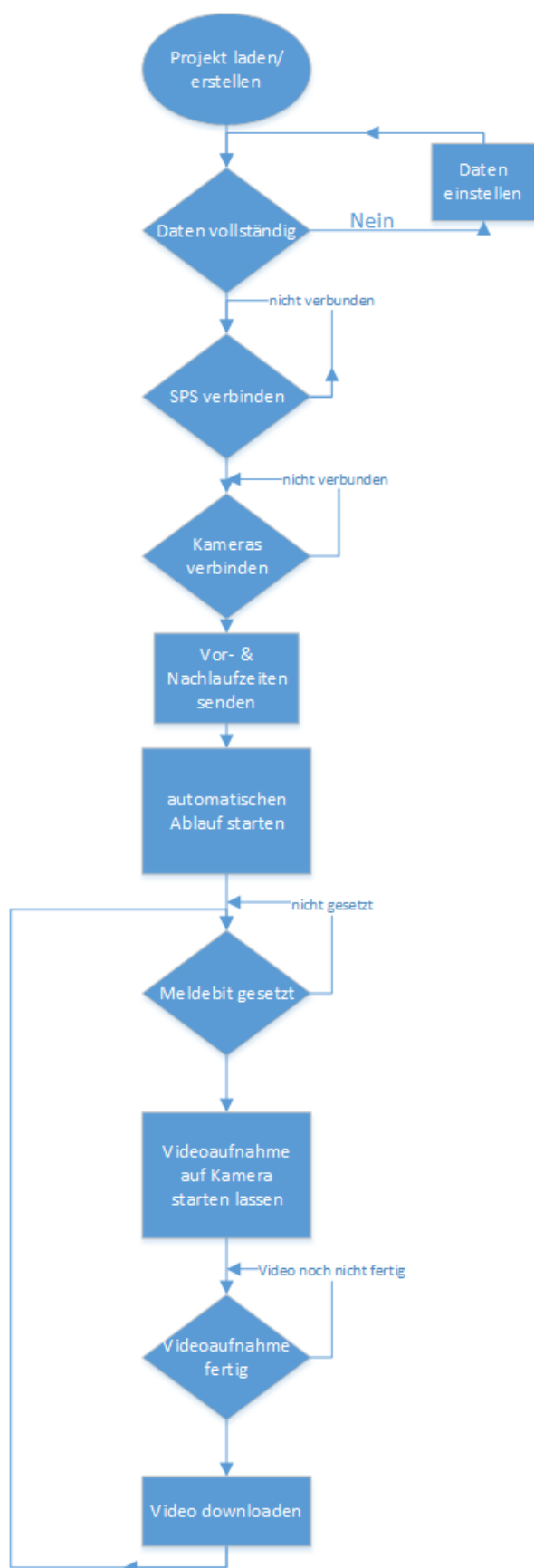


Abbildung 37: Flussdiagramm Ablauf – Hauptprogramm

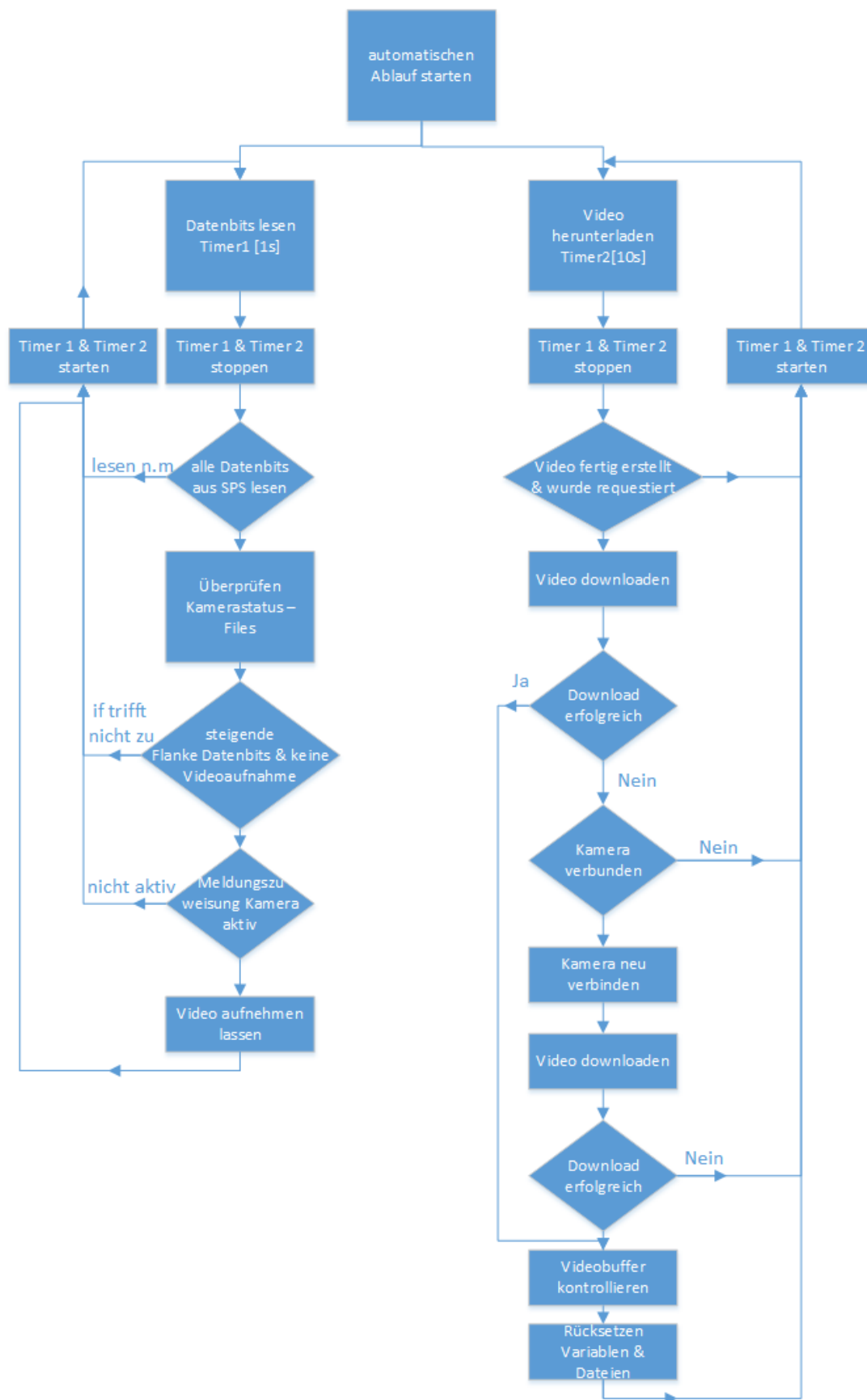


Abbildung 38: Flussdiagramm - automatisierter Ablauf

7.2 Grafische Oberfläche

Das Hauptprogramm ist eine grafische Oberfläche, das durch Menüleisten und *Panels* strukturiert ist. Panels würden, so Microsoft (vgl. Microsoft - Panels, 2019), zum Gruppieren von Auflistungen von Steuerelementen verwendet werden. Diese ermöglichen demnach auch das Anzeigen von Windows Forms innerhalb eines gewissen Bereiches. Der grau-strichlierte Rahmen in Abbildung 39 zeigt den Einschubbereich eines Panels. Unter diesem Panel-Rahmen befindet sich die Meldezeile, die die Meldungen anzeigt. Die obere Zeile symbolisiert die Menüleiste. Diese hat Unterpunkte, wie in Abbildung 40 ersichtlich ist.

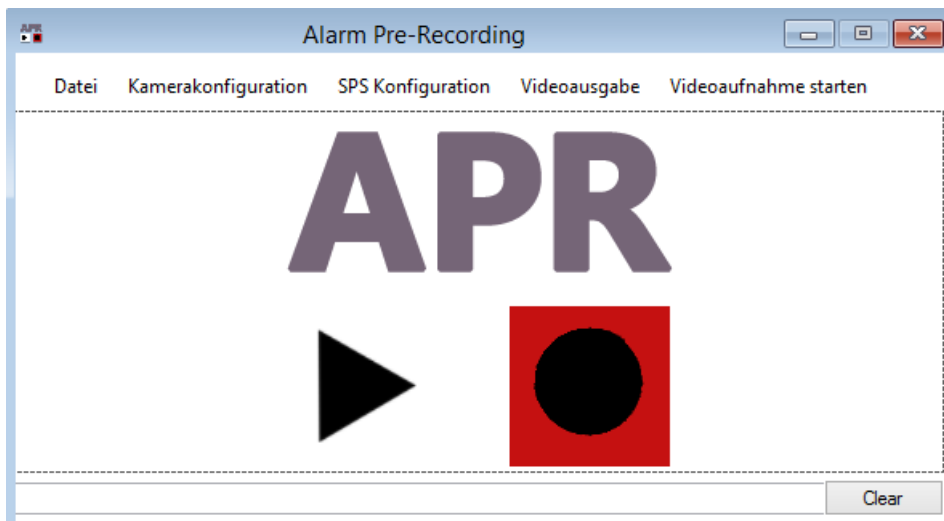


Abbildung 39: Hauptprogramm – Panelansicht

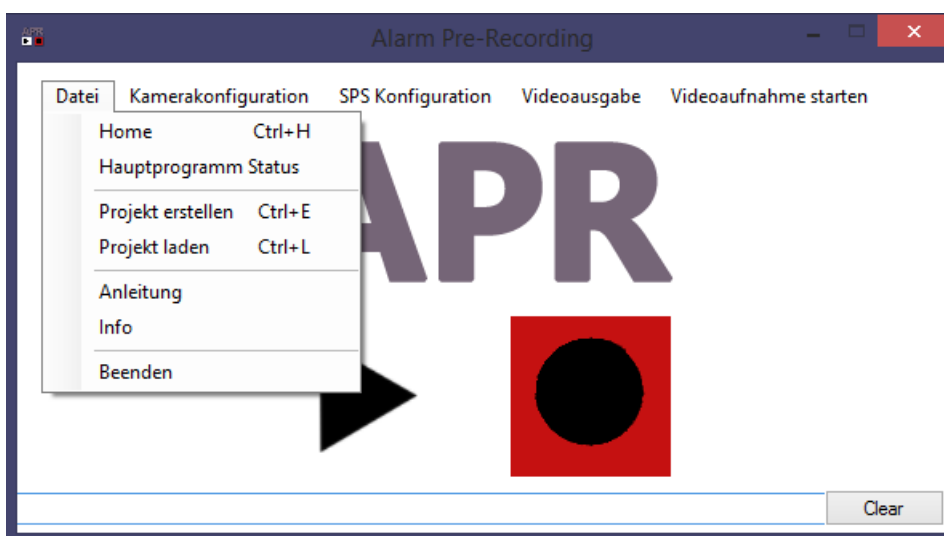


Abbildung 40: Menüunterpunkte des Hauptprogramms

7.2.1 Laden der Oberfläche

Beim ersten Laden der grafischen Oberfläche werden alle Menüunterpunkte, die nicht auswählbar sein sollen, deaktiviert. Wenn die Daten einer Einstellung hinzugefügt wurden, werden die entsprechenden *Buttons* aktiviert und der Zugriff auf einen Menüunterpunkt erlaubt. Des Weiteren wird ein Timer für das zyklische Leeren der Meldezeile aktiviert. Der Timer stammt aus der Bibliothek *System* und generiert alle zehn Sekunden ein Ereignis, das die angegebene Methode aufruft, siehe Abbildung 41 und Abbildung 42. Die angegebene Methode *T_changeMeldungszeile* ruft das Leeren der Meldezeile auf.

```
public static System.Timers.Timer TimerClearMeldungszeile = new System.Timers.Timer();
```

Abbildung 41: Erstellen eines Timers – Hauptprogramm

```
//Wenn Timer durch andere Form gestoppt wurde, ist der Timer nicht mehr enabled - daher ...  
if (!TimerClearMeldungszeile.Enabled)  
{  
    TimerClearMeldungszeile.Elapsed += new ElapsedEventHandler(T_changeMeldungszeile);  
    TimerClearMeldungszeile.Interval = 10000; //alle 10s die Meldungszeile zu löschen  
    TimerClearMeldungszeile.Start(); //wird auch gestoppt und deaktiviert  
}
```

Abbildung 42: Starten des Timers – Hauptprogramm

7.2.2 Windows Panels öffnen

Ein Panel ist ein geöffnetes Windows Form–Objekt, welches innerhalb eines Rahmens angezeigt wird. Der entsprechende Speicher des Panels muss freigegeben werden, wenn die Panelform nicht mehr benötigt wird. Da die Menüpunkte unabhängig voneinander vom Benutzer/der Benutzerin aufgerufen werden können, muss festgehalten werden, welche Windows Form geöffnet ist. Dieses Festhalten der letzten Windows Form ermöglicht das Freigeben des Speichers der Windows Form und wird mittels *References* festgelegt (vgl. Stackoverflow - Zugriff nichtstatischer Methoden, 2019).

Die maximale Größe eines Panels muss festgelegt werden und alle Formen, die innerhalb dieses Panels angezeigt werden sollen, müssen dieser Größe entsprechen. Hierfür wird zunächst der Rahmen der Windows Form entfernt, siehe Abbildung 43, und eine maximale Größe eingestellt. Dies hat den Hintergrund, dass das Schließen

von Forms innerhalb eines Panels verhindert werden und die Formgröße der Panelgröße entsprechen soll.

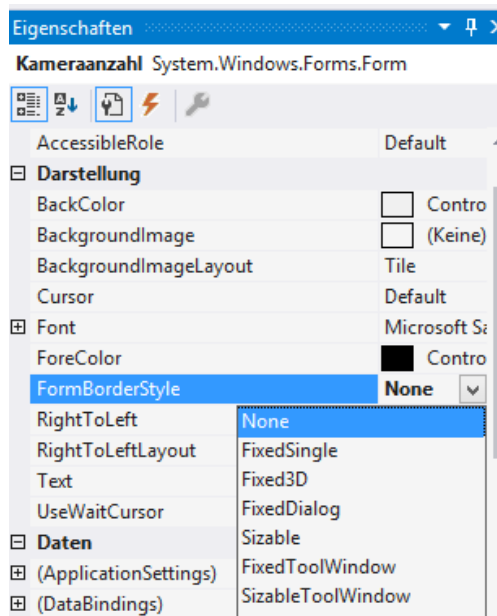


Abbildung 43: FormBorderStyle der Form entfernen

Das Öffnen von Panels wird am Beispiel des Kamerazuweisungspanels gezeigt, (vgl. Youtube - Panels erstellen, 2019).

Zunächst wird überprüft, ob die *Reference* auf das letzte Panel nicht *null* ist. Wenn dies der Fall ist und das vorige Panel der Formklasse „HauptprogrammStatus“ entspricht, so wird der entsprechende Timer des Hauptprogramm-Statusfensters gestoppt. Das Aktualisieren der Form, wenn diese nicht geöffnet ist, wird somit verhindert. In Abbildung 44 ist ersichtlich, wie die letzte Form aus dem Panel-Switch entfernt und der Speicher freigegeben wird. Panel-Switch entspricht dem Namen des Panels und stellt eine erkennbare Gruppierung für Steuerelemente, hier Windows Forms, bereit. Es wird die letztgeöffnete Form aus dem Panel entfernt und der zugehörige Speicher freigegeben.

```
//letztes Panel aus dem RAM und dem Panel-Switch entfernen
if (CurrentPanel != null)
{
    if (CurrentPanel.GetType().Name == "HauptprogrammStatus")
        //Check, ob das letzte Panel ein Kamerastatus Panel war - dann müssen die Timer gelöscht werden
        {
            HauptprogrammStatus.TimerRefreshStatus.Stop();
        }
    Panel_Switch.Controls.Remove(CurrentPanel);
    CurrentPanel.Dispose();
}
```

Abbildung 44: Hauptprogrammstatus-Timer stoppen

Nachdem das Panel entfernt wurde, wird eine neue Instanz der entsprechenden Windows Form eröffnet und dem Panel *Panel-Switch* hinzugefügt. Das Panel zeigt nun auf das gerade erzeugte Windows Form-Objekt und es wird angezeigt. Der Überschriftentext des Hauptprogrammes wird auf den Projektpfad und das entsprechende Windows Form-Objekt geändert, dies kann Abbildung 45 entnommen werden. Der *Toplevel* entspricht der Eigenschaft der Form, ob die Form über allen anderen Formen angezeigt werden soll oder nicht. In diesem Fall ist dies nicht erwünscht, (vgl. Microsoft - *TopLevel*, 2019).

```
//Neue Instanz der Form erstellen
KameraWarnung KameraMeldungPanelForm = new KameraWarnung();
KameraMeldungPanelForm.TopLevel = false;
//hier wird die WindowsForm KameraMeldung als Control in das Panel_Switch
//Panel eingespeist und angezeigt
Panel_Switch.Controls.Add(KameraMeldungPanelForm);
CurrentPanel = KameraMeldungPanelForm;
KameraMeldungPanelForm.Show();
this.Text = AlarmPreRecMain.projektFolderPath + " || Kamerazuweisung Meldung";
```

Abbildung 45: Dem Panel wird eine Windows Form hinzugefügt

Jedes andere Panel hat denselben Aufbau. Es können leichte Variationen vorkommen.

7.2.3 Videoaufnahme starten

Beim Öffnen der *Videoaufnahme starten*-Form muss überprüft werden, ob der Projektpfad inklusive Projektname angegeben, alle Einstellungen eingestellt und die SPS verbunden ist. Erst wenn dies der Fall ist, darf die Form geöffnet werden, siehe Abbildung 46.

```
//Um sich mit der Kamera verbinden zu dürfen, müssen diese Einstellungen vorhanden sein
if (Einstellungen.Kameraanzahl == 0 || Einstellungen.SPSVorNachlaufArray == (null) ||
    Einstellungen.SPSWarnungsEinstellungArray == (null))
{
    changeCurrentWorkStatus_Meldungsfeld("Nicht alle benötigten Daten wurden angegeben.", Color.Red);
    return;
}
try
{
    if (!SPSConnection.Ag_connection.Connected)
    {
        changeCurrentWorkStatus_Meldungsfeld("SPS nicht verbunden", Color.Red);
        return;
    }
}
catch (NullReferenceException)
{
    changeCurrentWorkStatus_Meldungsfeld("SPS nicht verbunden", Color.Red);
    return;
}
```

Abbildung 46: Überprüfen der Einstellungen zum Öffnen der *Videaufnahme starten* Form

Diese Form wird nicht in ein Panel geladen, sondern vollständig angezeigt. Das Einstellungs-, bzw. Hauptprogramm-Form wird versteckt, ist aber noch im Hintergrund aktiv. Somit ist der Zugriff auf Daten der Hauptprogramm-Form weiterhin möglich.

7.2.4 Ändern der Meldetextzeile

Die Meldezeile zeigt die aktuellen Meldungen, Erfolgs- oder Fehlermeldungen, in einer Textzeile an. Diese Meldungen werden alle zehn Sekunden, mit eines Timers der *System*-Bibliothek, geleert. Die Meldezeile befindet sich nicht in einem Panel, sondern im Grundaufbau der Benutzeroberfläche, dem Hauptprogramm *Alarm Pre-Recording*, und ist somit formübergreifend erreichbar.

Um den Inhalt der Meldezeile zu ändern, wird die Methode *changeCurrentWorkStatus_Meldungsfeld* aufgerufen. Sie ändert die Farbe und den Text dieses Textelementes des Hauptprogrammes.

Die Veränderung des Inhalts einer Meldezeile wird durch die oben genannte Methode in Abbildung 47 durchgeführt. Die Methode besitzt die zwei Übergabeparameter *currStatus* und *colourforTxt*. Sie ändern den aktuellen Meldestatus und die aktuelle Hintergrundfarbe der Meldezeile. Diese Einstellungen werden auf das eigentliche Meldungsfeld übertragen.

```
public void changeCurrentWorkStatus_Meldungsfeld(string currStatus, Color colourforTxt)
{
    CurrentWorkStatus.Text = currStatus;
    CurrentWorkStatus.BackColor = colourforTxt;
}
```

Abbildung 47: Funktionsbeschreibung *changeCurrentWorkStatus_Meldungsfeld*

7.2.5 Leeren der Meldezeile

Die Methode *ClearMeldungsfenster_Click()* leert die Meldezeile und färbt den Hintergrund der Meldezeile weiß. Da diese Methode sowohl vom Hauptprogramm, als auch vom Timer-Event, siehe Kapitel Laden der Oberfläche, aufgerufen wird, entsteht ein Problem. Die beiden Aufrufe geschehen von zwei verschiedenen Threads, dies ist jedoch nicht erlaubt. Die Erklärung der Threads ist in Abbildung 48 ersichtlich.

Ein *Prozess* ist ein ausführendes Programm. Ein Betriebssystem verwendet Prozesse, um Anwendungen voneinander zu trennen, die ausgeführt werden. Ein *Thread* ist die grundlegende Einheit, in der ein Betriebssystem die Prozessorzeit belegt.

Abbildung 48: Erklärung Threads (vgl. Microsoft - Threads, 2019)

Um den Zugriff auf einen anderen Thread gewähren zu können, muss dieser Zugriff erzwungen werden, siehe Kapitel 8.1.4 Fehler: Cross-Threads.

Benötigt ein Kontrollelement, wie in diesem Fall das Textelement, ein erzwungenes Eingreifen durch mehrere Threads, muss der Zugriff verändert geschehen. Es wird ein *Delegate* hergestellt, der die *Clear*-Methode erneut aufruft. Dieser Aufruf durch den *Delegate* ist ein „erzwungener“ Aufruf und umgeht das Problem der multiplen Threads, somit wird das Ändern von Kontrollelementen zur Laufzeit des Programmes ermöglicht. *Delegates* werden in Abbildung 49 erklärt.

Ein *Delegattyp* stellt Verweise auf Methoden mit einer bestimmten Parameterliste und dem Rückgabetyp dar. Delegate ermöglichen die Behandlung von Methoden als Entitäten, die Variablen zugewiesen und als Parameter übergeben werden können. Delegate ähneln dem Konzept von Funktionszeigern, die Sie in einigen anderen Sprachen finden. Im Gegensatz zu Funktionszeigern sind Delegate allerdings objektorientiert und typsicher.

Abbildung 49: Delegateerklärung (vgl. Microsoft - Delegate, 2019)

Nachdem das Kontrollelement erfolgreich aufgerufen wurde, werden die Inhalte des Textelements geleert, siehe Abbildung 50.

```
//Abfrage, ob Controll Element aktiviert werden muss
if (this.CurrentWorkStatus.InvokeRequired)
{
    //neuer Delegat erstellt und die Methode mit diesem Delegat aufgerufen
    StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(ClearMeldungsfenster_Click);
    this.Invoke(d, new object[] { sender, e });
}
else
{
    CurrentWorkStatus.Text = "";
    CurrentWorkStatus.BackColor = Color.White;
}
```

Abbildung 50: Leeren der Meldezeile mittels Delegate

7.2.6 Beenden

Soll das Programm geschlossen werden, muss die Verbindung zu den Kameras und der SPS getrennt sein. Bei der Verbindung mit den Kameras wird überprüft, ob diese verbunden waren und wenn ja, müssen sie getrennt werden. Des Weiteren werden alle Timerressourcen freigegeben und die SPS Verbindung getrennt. Die Implementierung ist in Abbildung 51 ersichtlich.

```
//Alle Verbindungen zu externen Geräten trennen
SPSConnection.DisconnectSPS();
if (Videoaufnahme.KameraXIsConnected != null)
{
    if (Videoaufnahme.KameraXIsConnected.GetLength(0) > 0)
    {
        for (int i = 0; i < Einstellungen.Kameraanzahl; i++)
        {
            if (Videoaufnahme.KameraXIsConnected[i] == true)
                KameraVerbindung.Disconnect(i);
        }
    }
}
TimerClearMeldungszeile.Dispose();
HauptprogrammStatus.TimerRefreshStatus.Dispose();
Datenbaustein.TimerCheckDB.Dispose();
Datenbaustein.TimerCheckIFVideoWasProduced.Dispose();
```

Abbildung 51: Verbindungen zu externen Geräten trennen

7.3 Einstellungen

7.3.1 Variablen

Im Programmteil *Einstellungen* befinden sich mehrere *public static* Variablen, die das Verhalten des Benutzerprogramms beeinflussen.

Die einzelnen Einstellungen für die SPS und Kamera, wie die Kameraanzahl und die Vor- und Nachlaufzeit, werden in einer für sie vorgesehenen statischen Variable

gespeichert. Formen, die grafische Oberfläche des Programms, sind nicht statisch und von der Programminstanz abhängig. Auch die Bedienelemente einer Form, wie ein Button, sind nicht statisch. Deswegen muss der Zugriff außerhalb der Forminstanz auf ein Bedienelement, über eine *Reference* stattfinden. Diese *Reference* ermöglicht den Zugriff über eine gespeicherte Instanz der Form und wird in Abbildung 53 dargestellt. Ihr Datentyp entspricht dem Klassendatentyp der Form, für die sie eine *Reference* herstellen soll. Die Funktionsweise einer *Reference* mit *get;set;* wird in Abbildung 52 erklärt, das *set;* Schlüsselwort weist einen Eigenschaftswert zu.

Das Schlüsselwort `get` definiert eine *Accessormethode* in einer Eigenschaft oder einem Indexer, die den Eigenschaftswert oder das Indexer-Element zurückgibt. Weitere Informationen finden Sie unter

Abbildung 52: Erklärung *References* - *get;* (vgl. Microsoft - *References*, 2019)

```
//Referenz um zur Haupt-Form zurückkehren zu können. In diesem Fall ist das AlarmPreRecMain
public static AlarmPreRecMain RefToMainForm { get; set; }
//Verweis zur aktiven Form Videoaufnahme - für den Aufruf nicht statischer Methoden
public static Videoaufnahme ViewVideoAufnahme { get; set; }
```

Abbildung 53: Reference nichtstatischer Formen

7.3.2 Speichern in einer CSV Datei

Die Daten der Einstellungen werden in einer Comma-Separated-Value Datei, kurz CSV-Datei, gespeichert. Das CSV-File trennt spaltenweise seine Zellen mit „ , “ und zeilenweise mit „\r“ und „\n“. Um eine Spalte auslesen zu können, wird eine Zeile eingelesen und der Inhalt einer Spalte sind alle Zeichen bis zum Zeichen „ , “. Das File kann als Ganzes oder zeilenweise eingelesen werden.

Wenn das File ganz eingelesen wird, muss ein eindimensionales Feld erstellt werden, das die gesamte Zeile in einem Feldelement speichert. Die Unterteilung der Zeilen wird mit einer *Split*-Funktion durchgeführt.

7.3.3 Speichersystem

In Abbildung 54 ist das Schema der Datenspeicherung in einem CSV-File dargestellt. Die blau gekennzeichneten Zellen sind Bezeichnungen der Zellenwerte, die nicht im tatsächlichen CSV-File enthalten sind. Sie wurden eingesetzt, um den Verlauf des Speicherschemas vereinfacht zu erklären. Die grün gekennzeichneten Zellen zeigen den Reihenindex der Datei, dieser ist auch nicht im CSV-File enthalten, sondern dient

der Erklärung des Schemas. Beim Festlegen einer Kameraanzahl wird automatisch ein Array erzeugt, das die Zeilenanzahl 4 und die Spaltenanzahl gleich der Variable *Kameraanzahl* beträgt. Dies hat zur Folge, dass das Einstellungsfile eine minimale Größe von vier Zeilen und einer Spalte hat. Die Zellen, die noch nicht manuell festgelegt wurden, wie die IP-Adresse einer Kamera, werden mit Standardwerten initialisiert. So lautet der Wert der Zelle der Zeile 1 und der Spalte 0 zum Beispiel „“. Wenn die Meldungen vom Benutzer/ von der Benutzerin festgelegt wurden, erstellt das System das Array erneut, wobei die Reihenlänge auf *4+Anzahl der Meldungen* festgelegt wird und die Spaltenlänge auf *4+Kameraanzahl*. Ein Array ohne Meldungen hat mindestens die Größe von vier Zeilen und eine Spaltengröße die der Kameraanzahl entspricht.

Zum Erstellen eines Einstellungsfiles wird mindestens die Kameraanzahl benötigt. Die Verbindungsparameter der Kamera werden beim Speichern mit ihren Standardwerten initialisiert. Die Vervollständigung der Konfigurationen erfolgt systematisch, so werden als Nächstes die Vor- und Nachlaufzeiten der Kameras festgelegt und die neuen Konfigurationen wiederum in das Einstellungsfile gespeichert. Danach werden die Datenbits der Meldungen festgelegt und die neuen Konfigurationen in das Einstellungsfile gespeichert. Dann werden die Zuweisungen der Datenbits festgelegt und gespeichert. Schlussendlich werden die Verbindungsdaten zur Kamera festgehalten und in das Einstellungsfile gespeichert. Mit der zunehmenden Befüllung der Konfigurationen gewinnt das Einstellungsfile an Größe. Sobald Konfigurationen eingestellt wurden, können diese einzeln aufgerufen und verändert werden. Das Einstellungsfile wird als Ganzes gespeichert.

Kamera anzahl									
Index									
Kameraanzahl	Kamera- anzahl (n)								0
	Kamera 1	Kamera 2	Kamera 3	Kamera n		Kamera 1	Kamera 2	Kamera n ...	
IP Adresse	zb 10.102.0. 10	„			Vorlauf- zeit (in s)	7	...		1
Passwort	**	„			Nachlau- f-zeit (in s)	8	...		2
Benutzer	pi	„							3
	Meldungs- -text	DB Nummer	Byte Nummer	Bit nummer	Kamera 1	Kamera 2	Kamera 3	Kamera n	
Index der Meldung	Meldung 1	10	0	n	True	False	True	False	4
Index der Meldung	Meldung 2	10	0	n+1	False	False	False	True	5
	Meldung k				k

Abbildung 54: Schema der Datenspeicherung in einem CSV-File

Eine fertige Einstellungsdatei kann wie Abbildung 55 aussehen. Hierbei ist zu beachten, dass die Einstellungen der Kameraverbindung, in Bezug auf deren IP-Adressen, entweder leer sind, oder einen vom Programm anerkannten Standardwert enthalten. Zum Beispiel „“ oder 10.102.0.10 für die IP-Adresse der Kamera.

```
3,,,,,
10.102.0.9,10.102.0.10,10.102.0.11,3,3,1,
**,**,**,2,5,1,
pi,pi,pi,,,
Merkerbit1,4,0,5,True,False,False
Merkerbit2,4,1,0,True,True,True
```

Abbildung 55: Beispiel einer fertigen Einstellungsdatei

7.3.4 Einstellungen speichern – Hintergründe

Das Speichern schreibt alle bisher eingestellten Konfigurationen in das Einstellungsfile, die Reihenfolge und der Aufbau entsprechen Abbildung 54. Es wird dabei unterschieden, ob eine Kameraanzahl zum ersten Mal gespeichert wird oder bereits eine andere Anzahl existierte. In der ersten Zelle wird die Kameraanzahl gespeichert, wie in Abbildung 56 ersichtlich ist.


```
List<string> FilesaveList = new List<string>();

if (FilesaveList.Count == 0)
    FilesaveList.Add(Kameraanzahl.ToString("00.0") + ",");
else
    FilesaveList[0] = Kameraanzahl.ToString("00.0") + ",";
```

Abbildung 56: Speichern der Kameraanzahl

Beim Speichern der Kamera-IP-Adressen werden die IP-Adressen mit zwei verschachtelten Schleifen implementiert. Eine Schleife steht für die Zeilen und die andere Schleife für die Spalten der Tabelle, siehe Abbildung 57.

```
//Zeilen sind Ip Adressen und etc, Spalten sind die einzelnen Kameras
for (int x = 0; x < KameraVerbindung.KameraIPAdressen.GetLength(0); x++)
{
    //speichert die Werte zeilenweise
    string content = "";
    for (int y = 0; y < KameraVerbindung.KameraIPAdressen.GetLength(1); y++)
    {
        content += KameraVerbindung.KameraIPAdressen[x, y] + ","; //siehe Tabellenschema
    }
}
```

Abbildung 57: Speichern der Kamera IP-Adressen

Falls bereits Vor- und Nachlaufzeiten existieren, werden diese in den Zeilen der Kamera IP-Adressen gespeichert, dies wurde bereits im Kapitel Speichersystem behandelt, siehe Abbildung 54. Es muss beachtet werden, dass die Vor- und Nachlaufzeiten nur in zwei Zeilen gespeichert werden.

```
//Wenn die Kameraipadressen gespeichert wurden, werden die Vor- Nachlaufzeiten ergänzt
//es gibt nur 2 Reihen im vornachlaufarray - Vor- und Nachlaufreihe
if (x < 2 && SPSVorNachLaufArray != null)
{
    for (int y = 0; y < SPSVorNachLaufArray.GetLength(0); y++)
    {
        content += SPSVorNachLaufArray[y, x] + ","; //siehe Tabellenschema
    }
}
```

Abbildung 58: Speichern der Vor- und Nachlaufzeiten

Nachdem die IP-Adressen und Vor- und Nachlaufzeiten zwischengespeichert wurden, werden sie der Liste hinzugefügt. Hier ist zu beachten, dass der Reihenindex der IP-Adressen als Index für die Liste dient. Demnach muss dieser Index inkrementiert werden, um die 0-te Reihe, die Kameraanzahl, zu überspringen, wie in der folgenden Abbildung zu sehen ist.

```
//Kameraeinstellungen in das Filearray speichern
if (FilesaveList.Count < 4)
    FilesaveList.Add(content);
else
    FilesaveList[x + 1] = content;
```

Abbildung 59: Speichern der Kamerakonfiguration in Hinsicht auf die IP-Adressen

Die Meldungen, bestehend aus der Datenbausteinnummer, dem Datenbausteinbyte und dem Datenbausteinbit, werden nach dem Datenspeicherungsschema in Abbildung 54 gespeichert. Eine mögliche Meldung kann folgendermaßen aussehen: „20.0.7“, hier entspricht 20 der Datenbausteinnummer, 0 entspricht dem Datenbausteinbyte und 7 entspricht dem Bit des Datenbausteinsbytes. Die Bits selbst werden in den ersten vier Spalten gespeichert, die Zuweisungen der Kameras in den weiteren Spalten. Die Zuweisungen besitzen eine Spaltenanzahl von *Kameraanzahl + maximale Spaltenanzahl der Meldebits*, siehe Abbildung 60. Sie starten beim Index vier, wie aus dem Datenspeicherungsschema des Kapitels Speichersystem, Abbildung 54, entnommen werden kann.

```
for (int x = 0; x < DBBitzeichenfolge.GetLength(0); x++)
{
    string content_db = "";
    //Wenn die DBBitzeichenfolge existiert, existiert auch die Meldung/Kameraeinstellungen,
    //auch wenn diese nicht fertig eingestellt sind
    for (int y = 0; y < 4; y++)
    {
        content_db += DBBitzeichenfolge[x, y] + ",";
    }
    for (int y = 4; y < (Kameraanzahl + 4); y++)
    {
        content_db += SPSWarnungsEinstellungArray[x, (y - 4)] + ",";
    }
}
```

Abbildung 60: Speichern der Meldebits

Die Daten werden der Liste nach dem Datenspeicherungsschema hinzugefügt, siehe Abbildung 61.

```
//Daten in das Array speichern
if (FilesaveList.Count < (DBBitzeichenfolge.GetLength(0) + 4))
    FilesaveList.Add(content_db);
else
    FilesaveList[x + 4] = content_db;
```

Abbildung 61: Der Speicherungsliste werden die Datenbits hinzugefügt

Nachdem die Einstellungen temporär der Liste hinzugefügt wurden, wird die Liste in ein externes CSV-File ausgeschrieben, siehe Abbildung 62. Die Daten werden zeilenweise ausgeschrieben, dafür wird ein *Streamwriter* verwendet. Dieser *Streamwriter* ist eine Unterstützung, um die *string*-Zeichen in das CSV-File speichern zu können, (vgl. Stackoverflow - CSV Files speichern, 2019).

```
//Ausschreiben in das lokale File
using (StreamWriter outfile = new StreamWriter(AlarmPreRecMain.projektFolderPath + "\\Einstellungen.csv"))
{
    //File ausschreiben
    for (int x = 0; x < FilesaveList.Count; x++)
    {
        outfile.WriteLine(FilesaveList[x]);
    }
    outfile.Close();
    return true;
}
```

Abbildung 62: Speichern der Konfigurationsdaten in die lokale Einstellungen Datei

Mit der Methode *WriteLine* (vgl. Microsoft - StreamWriter, 2019) wird ein String zeilenweise ausgeschrieben, siehe Abbildung 63.

<code>WriteLine(Single)</code>	Writes the text representation of a 4-byte floating-point value followed by a line terminator to the text string or stream. (Inherited from <code>TextWriter</code>)
--------------------------------	--

Abbildung 63: Methode *WriteLine* (vgl. Microsoft - StreamWriter, 2019)

7.3.5 Laden aller Konfigurationen aus dem Einstellungsfile

Wenn ein Projekt geladen werden soll, wird die Funktion *LoadCompleteEinstellungsfile()* aufgerufen. Hier wird das gesamte *Einstellungsfile* in eine *string* Variable geladen. Die Zeilenumbrüche *\n* werden durch *\r* ersetzt, um einen einheitlichen Zeilenumbruch innerhalb des Einstellungsfiles zu ermöglichen. Grundsätzlich können beide Zeilenumbruch-Indikatoren verwendet werden. Danach wird das gesamte geladene File in Spalten unterteilt. Nach jedem *\r* wird in einem *stringarray* eine neue Zeile begonnen und das File unterteilt, dies kann Abbildung 64 entnommen werden. Zum Schluss werden die leeren Zeilen und Spalten aus dem File entfernt.

```
string whole_file = System.IO.File.ReadAllText(AlarmPreRecMain.projektFolderPath +  
    "\\Einstellungen.csv");  
  
//In Zeilen aufspalten  
whole_file = whole_file.Replace('\n', '\r');  
string[] linesa = whole_file.Split(new char[] { '\r' },  
    StringSplitOptions.RemoveEmptyEntries);
```

Abbildung 64: Einlesen des Einstellungsfiles

Um einen Einstellungswert auslesen zu können, wird eine Zeile so lange eingelesen, bis ein „ , “ vorkommt. Der Wert bis zu „ , “ ist somit der Spaltenwert jener Zeile. Das Einlesen der Kameraanzahl ist in Abbildung 65 zu sehen.

```
string[] c_temp = linesa[0].Split(',');  
Kameraanzahl = Convert.ToDecimal(c_temp[0]);
```

Abbildung 65: Festlegen der Spaltenwerte des Einstellungsfiles

Um die Kamera-IP-Adressen einlesen zu können, siehe Abbildung 66, wird die Datengröße des Arrays festgelegt und die Zeilen und Spalten schleifenartig abgearbeitet. Zeilenweise wird die entsprechende Zeile in Spalten unterteilt. Diese Unterteilung wird temporär in *line_row* gespeichert. Die Daten werden nach dem Datenspeicherungsschema in Abbildung 54 abgearbeitet und geladen. Nach dem erfolgreichen Laden der Daten werden die entsprechenden Menüunterpunkte aktiviert und weitere Variablen, die von dem Fortschritt der Konfiguration abhängig sind, erstellt. Beispielsweise ist die Variable des Vor- und Nachlaufarrays von der Kameraanzahl abhängig und kann erst nach dem Einstellen der Kameraanzahl erstellt werden.

```
KameraVerbindung.KameraIPAdressen = new string[3, (int)Kameraanzahl];  
for (int row = 1; row <= 3; row++)  
{  
    string[] line_row = linesa[row].Split(',');  
    for (int c = 0; c < (int)Kameraanzahl; c++)  
    {  
        //Value array soll immer bei 0;0 starten  
        KameraVerbindung.KameraIPAdressen[row - 1, c] = line_row[c];  
    }  
}  
RefToMainForm.enableDB(true);  
RefToMainForm.enableVorNachLauf(true);
```

Abbildung 66: Einlesen der Kamera IP-Adressen

Das eben erklärte Einlesen der Daten wird für alle weiteren Daten nach dem Datenspeicherungsschema in Abbildung 54 angewendet. Es existieren leichte Abweichungen in der Implementierung, der Grundgedanke bleibt derselbe.

Wenn Daten nicht erfolgreich eingelesen werden können, wird eine Warnung ausgegeben und darauf hingewiesen, dass diese Daten ergänzt werden müssen.

7.3.6 Die Konfigurationsdaten auf den Standardwert zurücksetzen

Die Methode *ResetDataSet()* setzt alle Variablen auf ihren Standardwert zurück. Dieser Standardwert entspricht *null*, mit diesem Standardwert wird auf kein Objekt verwiesen und somit die Arraygrößen der Arrayvariablen entfernt.

Des Weiteren werden die Menüpunkte so deaktiviert, dass kein Zugriff auf noch nicht einstellbare Konfigurationen möglich ist. Das Verhindern des unerlaubten Zugriffes auf die Menüpunkte ist in Abbildung 67 ersichtlich.

```
//Menüpunkte deaktivieren, wenn man sie noch nicht erreichen darf
RefToMainForm.enableKameraCNT(false);
RefToMainForm.enableVorNachlauf(false);
RefToMainForm.enableWarnung(false);
RefToMainForm.enableKameraCNT(true);
RefToMainForm.enableDB(true);
```

Abbildung 67: Zugriffe auf die Menüpunkte festlegen

Es werden auch die Verbindungen zu den Kameras und der SPS getrennt und alle Timer gestoppt.

7.3.7 Ändern der Größe eines Arrays

Soll die Größe eines mehrdimensionalen *stringarrays* verändert werden, wird die Funktion *ResizeArray()* aufgerufen. Die Größe des *stringarrays* wird an die angegebenen Größen der Reihen und Spalten angepasst. Muss ein Array vergrößert werden, werden die leeren Zellen mit dem neuen Wert aufgefüllt. Der Rückgabeparameter entspricht dem neuen Array, (vgl. Stackoverflow - Arraygrößen verändern, 2019). Die Implementierung ist in Abbildung 68 ersichtlich, bei der Vergrößerung eines Arrays müssen die neuen Indizes mit dem neuen Wert initialisiert werden.

```
var newArray = new string[rows, cols];  
//gibt die kleinere Zahl aus - Restreihen / Spalten sind null  
int minRows = Math.Min(rows, original.GetLength(0));  
int minCols = Math.Min(cols, original.GetLength(1));  
for (int i = 0; i < minRows; i++)  
    for (int j = 0; j < minCols; j++)  
        newArray[i, j] = original[i, j];
```

Abbildung 68: Verändern der Größe eines Arrays

7.3.8 Neuladen des Datensatzes bei einer Kameraanzahländerung

Beim Ändern der Kameraanzahl müssen mit der Funktion *ReloadDataSetUponKameraChange()* die abhängigen Daten verändert werden. Wenn die Anzahl der Kameras der Kamera IP-Adressen verändert wurde, wird jede Kameraverbindung getrennt und der IP-Adressen-Array verändert. Hier muss noch abgefragt werden, ob die Kameras wirklich verbunden waren, siehe Abbildung 69.

```
//wenn die Kameraanzahl geändert wurde, müssen alle Kameras neu verbunden und gespeichert werden  
for (int i = 0; i < KameraVerbindung.KameraIPAdressen.GetLength(1); i++)  
{  
    if (Videoaufnahme.KameraXIsConnected != null)  
        if (Videoaufnahme.KameraXIsConnected.GetLength(0) > 0)  
            if (Videoaufnahme.KameraXIsConnected[i] == true)  
                KameraVerbindung.Disconnect(i); //dadurch, dass alle Kameraverbindungen getrennt wurden,  
                                                    //müssen die Kameras neu verbunden werden  
                                                    //und deren IP Adressen Einstellungen überprüft werden  
}  
KameraVerbindung.KameraIPAdressen = ResizeArray(KameraVerbindung.KameraIPAdressen, 3,  
    (int)Kameraanzahl, "");
```

Abbildung 69: Die Größe der Kamera IP-Adressen verändern

Bei der Änderung der Kameraanzahl muss auch das Array der Vor- und Nachlaufzeiten und das Array der Kamerazuweisungen verändert werden.

7.3.9 Neuladen des Datensatzes bei einer Datenbitänderung

In der Funktion *ReloadDataSetUponDBBitsChange()* wird, wie auch in *Neuladen des Datensatzes bei einer Kameraanzahländerung*, die Größe abhängiger Variablen verändert. Diese abhängige Variable ist das Kamerazuweisungsarray und muss beim Hinzufügen von Meldebits auf die Größe der Kamerazuweisungen verändert werden. Dies wird durch die Funktion *ResizeArray* bewerkstelligt.

7.4 Projekte laden oder erstellen

Um die vom Benutzer/von der Benutzerin eingegebenen Daten zu speichern, wird dem Projekt ein Ordner zugewiesen. Es kann ein neues Projekt angelegt oder ein bereits vorhandenes Projekt geladen werden.

Der Projektpfad wird in der *public static* Variable *Projektpath* gespeichert. Der Datentyp der Variable ist ein *string*. Dieser beinhaltet den Projektpfad und den Projektnamen des Projektordners als *string*.

7.4.1 Projekt erstellen

Wenn ein neues Projekt erstellt wird, muss der Menüunterpunkt *Projekt erstellen* ausgewählt werden. Hier muss der neue Projektname angegeben werden. Ebenso soll der zu verwendende Projektpfad angegeben werden, in diesem wird der neue Ordner erstellt, (vgl. Stackoverflow - Ordner auswählen, 2019). Das Erstellen eines neuen Ordners erfolgt durch das Klicken des Buttons *Projekt erstellen*. Sobald ein Projektname oder ein Projektpfad festgelegt wurde, wird dies in dem entsprechenden Textfeld angezeigt, dies kann der Abbildung 70 entnommen werden.

```
//ausgewählten Pfad in der Textbox anzeigen lassen  
prjktPath_txtBox.Clear();  
prjktPath_txtBox.AppendText(browseProjektFolderPath.SelectedPath);
```

Abbildung 70: Projektpfad in der Textbox anzeigen lassen

Wenn während der momentanen Programminstanz bereits einmal ein *AlarmPreRecording.Projektpath* festgelegt wurde, wird dieser als Standardwert ausgewählt, es wird somit im Ordnerverzeichnis zum Projektpfad gesprungen. Diese Implementierung ist in Abbildung 71 zu sehen.

```
//setzt beim Öffnen der Suchfunktion den initialisierten Pfad auf den vorherigen Pfad  
browseProjektFolderPath.RootFolder = Environment.SpecialFolder.MyComputer;  
if (AlarmPreRecMain.projektFolderPath != null)  
    browseProjektFolderPath.SelectedPath = AlarmPreRecMain.projektFolderPath;
```

Abbildung 71: Sprung zum Projektpfad im Ordnerverzeichnis

Der angewählte Projektpfad wird in der Textanzeige angezeigt. Der Code hierfür ist in Abbildung 72 ersichtlich.

```
// Man lässt einen Pfad suchen, der der neue Projektpfad werden soll
DialogResult result = browseProjektFolderPath.ShowDialog();
if (result == DialogResult.OK)
{
    //Pfad wurde in der globalen privaten Variable zwischengespeichert - wird überschrieben
    //ausgewählten Pfad in der Textbox anzeigen lassen
    prjktPath_txtBox.Clear();
    prjktPath_txtBox.AppendText(browseProjektFolderPath.SelectedPath);
}
```

Abbildung 72: Anzeigenlassen des momentanen Projektpfades

Die Oberfläche des Menüunterpunktes *Projekt erstellen* ist in Abbildung 73 ersichtlich.

Beim Erstellen eines neuem Projekts, müssen alle Einstellungen neu eingegeben und gespeichert werden!

Projektname angeben

Projektpfad angeben und Projekt erstellen

Abbildung 73: Grafische Oberfläche - Projekt erstellen

Beim Erstellen eines neuen Projektes werden alle statischen Variablen auf den Standardwert *null* zurückgesetzt und verweisen somit auf kein Objekt. So wird verhindert, dass ein zuvor geöffnetes Projekt Einwirkung auf ein neues Projekt hat. Hier wird nochmal überprüft, ob ein Projektname und ein Projektpfad angegeben wurden, dies ist in Abbildung 74 dargestellt.

```
if (browseProjektFolderPath.SelectedPath == null)
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kein Projektpfad angegeben!" +
        " Erstellen des Projektes wird abgebrochen",
        System.Drawing.Color.Red);
    return;
}

// neuen Projektordner erstellen
if (projektName == null || projektName.Equals(""))
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kein Projektname angegeben! " +
        "Erstellen des Projektes wird abgebrochen",
        System.Drawing.Color.Red);
    return;
}
```

Abbildung 74: Projektname und -Pfad überprüfen

Danach wird der neue Projektpfad festgelegt, dieser beinhaltet den Projektordner. Weiters wird das Ordnerverzeichnis erstellt. Dieses erstellt nur dann die jeweiligen Ordner, wenn sie nicht schon bereits existieren, siehe Abbildung 75. Die Funktion hierfür wird in Abbildung 76 beschrieben, Quelle: (vgl. Microsoft - Create Directory, 2019)

```
AlarmPreRecMain.projektFolderPath = browseProjektFolderPath.SelectedPath + "\\\" + projektName;  
//Erstellt keinen Ordner, wenn das Verzeichnis bereits vorhanden ist  
Directory.CreateDirectory(AlarmPreRecMain.projektFolderPath);
```

Abbildung 75: Projektverzeichnis erstellen

Creates all directories and subdirectories in the specified path unless they already exist.


C#	 Copy
<pre>public static System.IO.DirectoryInfo CreateDirectory (string path);</pre>	

Abbildung 76: Erstellen eines Ordners (vgl. Microsoft - Create Directory, 2019)

Nach dem Erstellen des Projektes werden die Daten in der Datei *Einstellungen* zurückgesetzt. Diese Funktion wird im Kapitel 7.3 Einstellungen erklärt.

7.4.2 Projekt laden

Wenn ein bestehendes Projekt geladen werden soll, wird der entsprechende Menüunterpunkt geöffnet. Hier wird der Projektpfad festgelegt und das Laden der Daten aus dem Projektpfad gestartet.

Das Festlegen des Projektpfades, welcher im Programm verwendet wird, entspricht der Implementierung in *Projekt erstellen*. Wenn der Projektpfad festgelegt wurde, wird die Methode aufgerufen, die die Einstellungen aus dem Projektpfad lädt. Diese Methode wird im Kapitel 7.3 Einstellungen implementiert und erklärt. Wenn beim Laden der Einstellungen ein Fehler aufgetreten ist oder nicht alle Daten aus der Datei gelesen werden konnten, wird darauf hingewiesen, dass diese neu einzutragen sind. Die grafische Oberfläche der Form *Projekt laden* ist in Abbildung 77 dargestellt.



Abbildung 77: Projekt laden - grafische Oberfläche

7.5 Hauptprogramm-Statusanzeige

Nach dem Laden oder Erstellen eines Projektes wird die Hauptprogramm-Statusanzeige angezeigt. Diese veranschaulicht in Abbildung 78, welche Einstellungen noch getroffen werden müssen und welche bereits getroffen wurden. Die Zustände der Einstellungen werden alle 20 Sekunden aktualisiert.

	Status	Bearbeiten
► Kameraanzahl	2.0	Kameraanzahl bearbeiten
Vor- und Nachlaufzeiten	eingestellt	Vor- & Nachlaufzeiten bearbeiten
Datenbits für Meldungen	Aktiv	Meldungsbits bearbeiten
Meldungszuweisungen	zugewiesen	Meldungszuweisungen bearbeiten
SPS - Verbindung	nicht verbunden	SPS Verbindung bearbeiten
Kamera - Verbindung	nicht verbunden	Kamera Verbindung bearbeiten

Peripherie verbinden und System starten

Abbildung 78: Hauptprogrammstatus

Da die Aktualisierung der Form nur durchgeführt werden kann, wenn die Hauptprogrammstatus- Form auch aktiv ist, muss dies überprüft werden. Wird diese Statusform geschlossen, so muss auch die Aktualisierung der Form, also der entsprechende Timer, gestoppt werden. Diese Aktualisierungsmethode wird alle 20 Sekunden vom Aktualisierungstimer aufgerufen. Diese Methode überprüft die Werte einzelner Konfigurationen und den Verbindungsstatus der Kameras und der SPS und zeigt diese Änderungen in der Form an. Um Verbindungen feststellen zu können, wird für die SPS Verbindung versucht ein Bit auszulesen, und für die Überprüfung der Kamerverbindungen kontrolliert, ob ein File existiert, das immer auf dem

Kamerasystem existieren sollte. Kann eine Überprüfung nicht durchgeführt werden, so wird die Verbindung als *unterbrochen* angesehen.

Um eine Einstellung verändern zu können, können die Buttons des *DataGridView*-Elements, siehe Abbildung 79, gedrückt oder der entsprechende Menüunterpunkt ausgewählt werden. Ein *DataGridView*-Element ist eine Tabelle der Windows Form. Es ist möglich dieser Tabelle Daten oder Schaltflächen zuzuschreiben. Wird ein Button-Klick registriert, muss zunächst überprüft werden, für welchen Button der Buttonspalte der Klick gilt. Daraufhin wird der richtige Menüunterpunkt geöffnet, siehe Abbildung 80.

Das `DataGridView`-Steuerelement ermöglicht die flexible Anzeige von Daten in tabellarischer Form. Sie können das `DataGridView`-Steuerelement verwenden, um schreibgeschützte Ansichten mit kleinen Datenmengen anzuzeigen, oder Sie können es skalieren, um bearbeitbare Ansichten von sehr umfangreichen Datasets anzuzeigen.

Abbildung 79: DataGridView-Elemente (vgl. Microsoft - DataGridView-Elemente, 2019)

```
if (e.ColumnIndex == HauptprogrammStatusDataGrid.Columns[2].Index)
{
    switch (e.RowIndex)
    {
        case 0:
            Einstellungen.RefToMainForm.anzahlToolStripMenuItem_Click();
            break;
        case 1:
            Einstellungen.RefToMainForm.vorlaufzeitUndNachlaufzeitDerKameraToolStripMenuItem_Click();
            break;
    }
}
```

Abbildung 80: Öffnen des Menüunterpunktes - Hauptprogramm-Status

7.5.1 Start der Videoaufnahme-Form

Wurden alle Einstellungen erfolgreich geladen, können mit einem Klick die SPS und alle Kameras verbunden werden. Hier wird nochmals überprüft, ob alle Einstellungen und der Projektpfad angegeben wurden. Danach werden die SPS und alle Kameras verbunden und das Fenster Videoaufnahme geöffnet.

7.6 Kameraanzahl

Im Menüunterpunkt Kameraanzahl wird die Anzahl der zu verwendenden Kameras eingestellt. Das Einlesen der Anzahl erfolgt über eine numerische *Up&Down Box*, siehe Abbildung 81. Eine numerische *Up&Down Box* inkrementiert oder dekrementiert eine Zahl aufgrund des Drückens einer Schaltfläche. Es wird ein Wertebereich

vorgegeben, in dem sich die Kameraanzahl befinden kann. Hier wurde ein Minimum von 0 und ein Maximum von 100 Kameras angesetzt, siehe Abbildung 82.

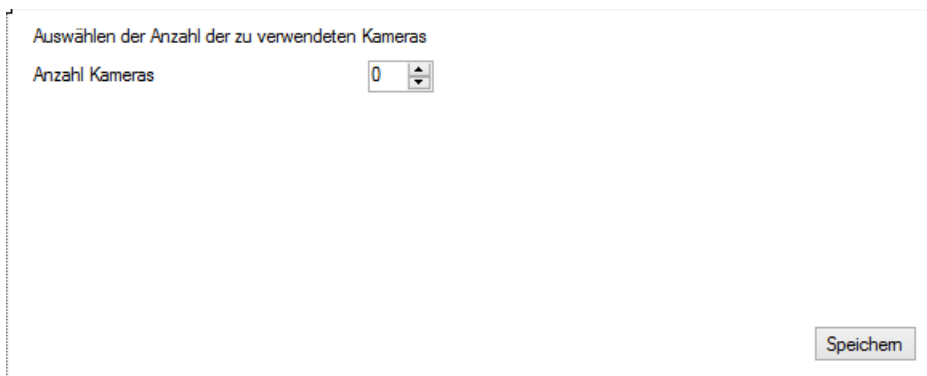


Abbildung 81: Grafische Oberfläche - Einstellungen der Kameraanzahl

Daten	
(ApplicationSettings)	
(DataBindings)	
DecimalPlaces	0
Increment	1
Maximum	100
Minimum	0

Abbildung 82: Festlegung des Minimums und Maximums der Kameraanzahl

Wurde bereits ein Projekt geladen, so wird die Kameraanzahl in die *Up&Down Box* geladen und angezeigt.

7.6.1 Speichern der Kameraanzahl

Beim Speichern der Kameraanzahl wird wieder überprüft, ob ein Projektpfad angegeben wurde und ob die Kameraanzahl nicht 0 ist, siehe Abbildung 83.

```
if (anzKameras.Value == 0)
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Geben Sie eine Kameraanzahl ein",
        System.Drawing.Color.Red);
    return;
}
//Um sicher zu gehen
if (AlarmPreRecMain.projektFolderPath == null)
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kein Projekt angegeben!" +
        " Erstellen Sie ein neues oder laden Sie ein bereits vorhandenes Projekt",
        System.Drawing.Color.Red);
    return;
}
```

Abbildung 83: Überprüfung des Projektpfades bei der Kameraanzahl

Wenn diese Anforderungen erfüllt sind, wird die Kameraanzahl *intern* gespeichert und in die externe Datei der Einstellungen gespeichert. Weiters wird aus dem Programmteil *Einstellungen* die Methode aufgerufen, die signalisiert, dass die Kameraanzahl geändert wurde und alle davon abhängigen Größen auch geändert werden müssen, *ReloadDataSetUponKameraChange()*.

Die Implementierung der Methode des Speicherns wird im Kapitel 7.3 Einstellungen erklärt, einzelne Schritte sind hingegen in Abbildung 84 ersichtlich.

```
//Kameraanzahl in das File speichern und die Einstellungen lokal
Einstellungen.Kameraanzahl = anzKameras.Value;
//Kameraanzahl könnte geändert worden sein, wenn dies der Fall ist werden die
//abhängigen Einstellungen angepasst
Einstellungen.ReloadDataSetUponKameraAnzahlChange();
//Daten in das Einstellungsfile speichern
Einstellungen.SaveAllSettingsInFile();
Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kameraanzahl wurde erfolgreich" +
    " gespeichert", System.Drawing.Color.LightGreen);
```

Abbildung 84: Kameraanzahl speichern

Danach werden die Menüunterpunkte *Vor- und Nachlaufzeiten* und *Datenbausteine* aktiviert, siehe Abbildung 85. Die genaue Implementierung der Methoden erfolgt im Kapitel 7.3 Einstellungen, hier werden lediglich die Werte der Menüunterpunktschaltflächenfreigabe übergeben. Im Beispiel der Abbildung 85 entsprechen die Zustände der Menüunterpunkte dem Wert *true* und können somit gedrückt werden.

```
//Aktivieren der Menüreiter
Einstellungen.RefToMainForm.enableVorNachLauf(true);
Einstellungen.RefToMainForm.enableDB(true);
```

Abbildung 85: Aktivierung der Menüunterpunkte

Wird bei der Kameraanzahlspeicherung eine *exception*, also eine Ausnahme im Codeblock, geworfen, wird diese von einem *catch* aufgefangen und eine Fehlermeldung ausgegeben. *Try-catches* sind *exception handlings*, sogenannte Ausnahmebehandlungen, die versuchen einen Codeblock auszuführen. Wenn in diesem Codeblock eine *Exception* geworfen wird, wird der *catch* Codeblock ausgeführt. Dieser kann etwaige Fehlermeldungen enthalten, (vgl. Stackify - Exception Handlings, 2019). Das Speichern der Kameraanzahl wird ebenfalls abgebrochen wie in Abbildung

86 dargestellt ist. Diese Abbildung zeigt den Inhalt des catch-Blocks, der try-Block beinhaltet die Codesegmente aus Abbildung 84.

```
Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kameraanzahl konnte nicht" +  
" gespeichert werden", System.Drawing.Color.Red);  
return;
```

Abbildung 86: Kameraanzahl konnte nicht gespeichert werden

7.7 Datenbaustein – Meldebits

Um die einzelnen Meldungen festlegen zu können, wird ein *Datatable* angelegt. In dieser Tabelle werden die einzelnen Spalten der Tabelle hinzugefügt.

Die grafische Darstellung der Meldebits ist in Abbildung 87 dargestellt. Diese zeigt auch die benötigten Zeilen und Spalten des *Datatables*.

Es wurden die Überschriften aller Spalten in dem *Datatable* festgelegt. Die Standardeinstellungen der Datenbits sind in Abbildung 88 ersichtlich. Dieser Default wird dem Benutzer/der Benutzerin angezeigt, wenn Meldungen erstmals hinzugefügt werden.

	Merkerbitname	Datenbausteinnummer	Bytenummer	Bitnummer
	Merkerbit1	20	0	0
	Merkerbit2	20	0	1
	Merkerbit3	20	0	2
▶	Merkerbit4	20	0	3
*				

Merkerbits wurden erfolgreich gespeichert

Abbildung 87: Grafische Darstellung – Meldebits

	Merkerbitname	Datenbausteinnummer	Bytenummer	Bitnummer
	Merkerbit1			
▶*				

Abbildung 88: Standardeinstellungen der grafischen Oberfläche der Datenbits

Die Spalten *Datenbausteinnummer*, *Bytenummer* und *Bitsnummer* sind Spalten des Datentyps *string* und somit Textfelder, siehe Abbildung 89.

```
DBBitsTable.Columns.Add("Datenbausteinnummer", typeof(string));  
DBBitsTable.Columns.Add("Bytenummer", typeof(string));  
DBBitsTable.Columns.Add("Bitnummer", typeof(string));
```

Abbildung 89: Hinzufügen der Spalten der Datenbits

Existieren bereits Meldebit-Informationen in den Einstellungen, werden diese in das grafische Element geladen.

Beim Speichern der Meldungen werden mittels zweier verschachtelter Schleifen die Zeilen und Spalten abgearbeitet. Hier wird überprüft, ob keine Zelle der Meldungen leer ist. Die letzte Zeile wird ignoriert, da sie das Hinzufügen einer neuen Zeile ermöglicht. Ebenso wird überprüft, ob die Bitnummer kleiner als acht ist. Der Grund dafür ist die maximale Anzahl an Bits. Ein Byte entspricht acht Bits, die von 0 bis 7 verlaufen.

Weiters werden die Zellenwerte auf ihre Richtigkeit überprüft. Wenn ein Zellenwert, abgesehen von dem Meldebitnamen, nicht ganzzahlig ist, wird ein Fehler ausgegeben. Das Überprüfen erfolgt mit eines Integer Parsers, siehe Abbildung 90.

```
//check if value is all numbers  
//isNumeric = True - wenn erfolgreich konvertiert wurde  
//isNumeric = False - wenn in 'value' ungültige Zeichen (Buchstaben, Sonderzeichen) vorhanden  
//sind oder dieser leer ist  
  
//nur positive Zahlen  
if (col != 0)  
{  
    bool isNumeric = Int32.TryParse(value, NumberStyles.None, CultureInfo.InvariantCulture,  
        out int resultplacebo);  
  
    if (!isNumeric)  
    {  
        Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Ungültige Eingabe bei" +  
            " Kamera " + (rows+1), System.Drawing.Color.Red);  
        DBBitsArray = null;  
        return;  
    }  
}
```

Abbildung 90: Parsen der Zahlenwerte

War die gesamte Eingabe in Ordnung, werden die Meldebits *extern* gespeichert und die internen Variablen verändert, die durch die Eingabe der Meldebits verändert werden müssen, siehe Abbildung 91. Die Methode *DBBits_Saver* überprüft die Existenz des Projektpfades und ruft das Speichern der Einstellungswerte auf.

```
//DBits lokal speichern
Einstellungen.DBBitzeichenfolge = DBBitsArray;
Einstellungen.ReloadDataSetUponDBBitsChange();
Einstellungen.DBZustandbeiBit = new int[Einstellungen.DBBitzeichenfolge.GetLength(0)];
DBBits_Saver();
Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Merkerbits wurden erfolgreich" +
    " gespeichert", System.Drawing.Color.White);
Einstellungen.RefToMainForm.OpenhauptPrgrmStatus();
```

Abbildung 91: Speichern der Meldebits

7.8 SPS Verbindung

Beim Start der SPS-Verbindung wird überprüft, ob bereits eine Verbindungseinstellung für den AGLINK, siehe Kapitel 6 AGLINK4 – SPS Verbindung, existiert. Wenn dies der Fall ist, kann sofort eine Verbindung mit der SPS hergestellt werden, ohne vorher die Einstellungen erneut anzugeben. Dies wird in Abbildung 92 angezeigt.

```
//überprüft, ob ConfigFile schon existiert, Nachsehen in Projektpfad
bool configExistsAlready = File.Exists(AlarmPreRecMain.projektFolderPath + "\\AGLink40CfgDev0000.xml");
//wenn obiges File existiert, wird der Verbinden Button freigeschalten, sonst muss man manuelle
//Verbindung hergestellt werden
buttonVerb.Enabled = configExistsAlready;
```

Abbildung 92: Überprüfen ob die AGLINK Einstellungen bereits existieren

Die grafische Oberfläche der SPS-Verbindung wird in Abbildung 93 dargestellt.

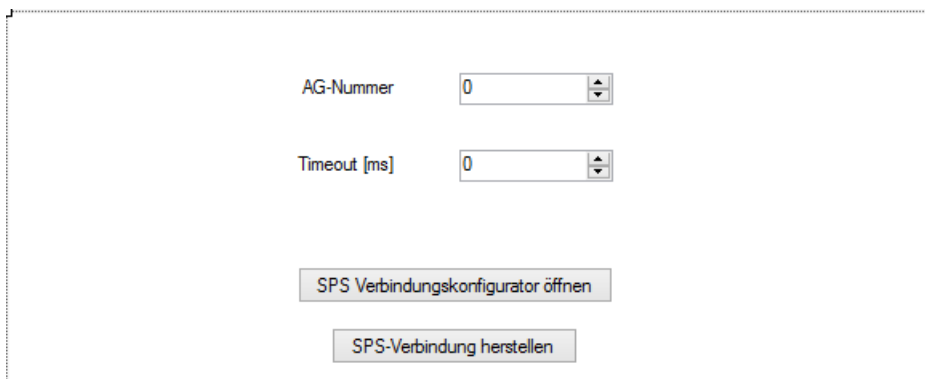


Abbildung 93: Grafische Oberfläche der SPS-Verbindung

7.8.1 SPS-Verbindungskonfigurator öffnen

Hier wird nochmals der Projektpfad auf seine Existenz überprüft. Wenn der Pfad nicht angegeben wurde, wird dazu aufgefordert einen Pfad anzugeben und das Öffnen des Konfigurators wird abgebrochen.

Weiters wird überprüft, ob bereits eine Einstellungsdatei für den AGLINK existiert. Wenn dies der Fall ist, kann sofort eine Verbindung mit der SPS hergestellt werden, ohne die Einstellungen neu angeben zu müssen. Dies wird in Abbildung 92 dargestellt. Hier, in Abbildung 94, wird der Verbindungskonfigurator geöffnet, es werden die Devicenummer und der Projektordnerpfad angegeben. Zudem wird der Speicherpfad der automatisch erstellten Einstellungen des AGLINKs auf den Projektordner angesetzt.

```
//Öffnen des AGLink Verbindungskonfigurator
resConf = AGL4.ConfigEx(devNr, "-f -p=\"\" + AlarmPreRecMain.projektFolderPath + "\"");
//Pfad für Speicherung der Verbindungsdaten
AGL4.SetParaPath(AlarmPreRecMain.projektFolderPath);
```

Abbildung 94: Öffnen des Verbindungskonfigurators

Der Verbindungskonfigurator, ersichtlich in Abbildung 95, erstellt die Verbindungseinstellungen zur SPS. Die Verbindung zu einer SPS muss mindestens einmal getestet werden um die Konfigurationsdatei erfolgreich herstellen zu können, siehe Abbildung 96.

AG-Nr	IP-Adresse	Rack	Slot	Typ
0	10.102.0.101			S7-1200/1500

Abbildung 95: Verbindungskonfigurator

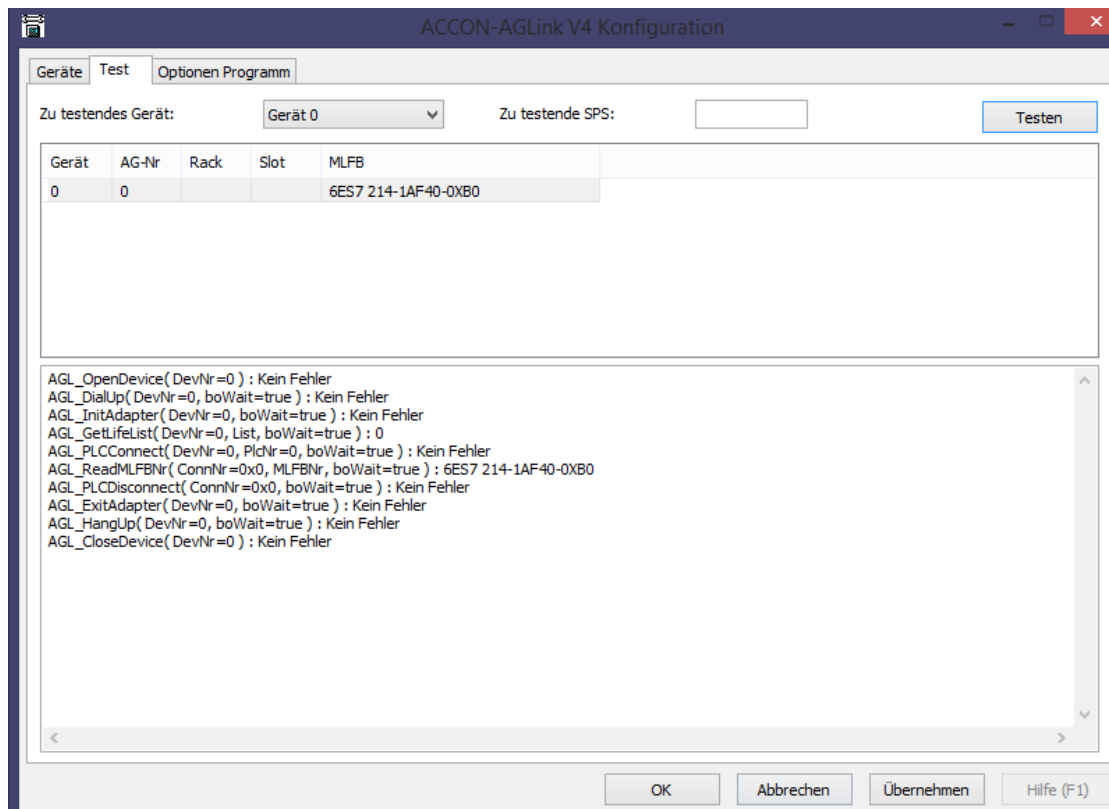


Abbildung 96: Verbindung zur SPS testen

Wenn das Erstellen der Einstellungsdaten des AGLINKs erfolgreich durchgeführt wurde, wird der Button zur *Verbindung der SPS* aktiviert, Abbildung 97. Sollten jedoch keine gültigen AGLINK Einstellungen festgelegt worden sein, wird eine Fehlermeldung in der Meldezeile ausgegeben, Abbildung 98.

```

//AGLink konnte erfolgreich eine Probeverbindung herstellen. Die Einstellungen passen für den AGLIN
if (resConf == AGL4.AGL40_SUCCESS)
{
    buttonVerb.Enabled = true;
}
    
```

Abbildung 97: AGLINK Einstellungen erfolgreich gespeichert

```

else
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Konfigurator konnte nicht " +
        "ausgeführt werden", System.Drawing.Color.Red);
}
    
```

Abbildung 98: Fehlermeldung bei keine gültigen AGLINK Einstellungen

Beim erfolgreichen Verbinden der SPS wird die Nachricht ausgegeben, die in Abbildung 99 ersichtlich ist.

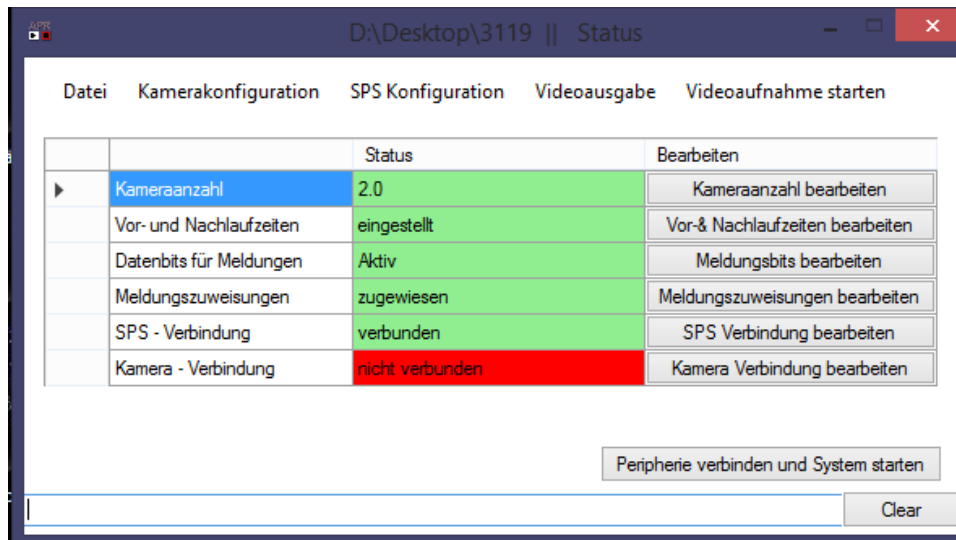


Abbildung 99: Erfolgreiches Verbinden mit der SPS

7.8.2 Verbindung mit der SPS

Wenn die Einstellungsdatei der SPS bereits im Projekt angelegt wurde oder eine neue im Konfigurator erstellt wurde, kann eine Verbindung mit der SPS erstellt werden. Zunächst werden der Projektpfad und die Einstellungen einer Instanz der AGLINK-Verbindung übergeben. Diese speichert die Einstellungen und ermöglicht eine Verbindung zur SPS, Abbildung 101. Die Variable hierfür ist eine statische Variable, da die Verbindung auch aufrecht sein soll, wenn die Form nicht geladen ist. Diese Variable ist vom Datentyp *IAGLink4Tia* und stammt aus der Bibliothek *Accon.AGLink*, welche die Einstellungen des Verbindungskonfigurators speichern kann, wie in Abbildung 100 ersichtlich ist. Diese besitzt die Properties *get*; *private set*; und erlaubt den Lesezugriff und den privaten Schreibzugriff der Variable.

```
public static IAGLink4Tia Ag_connection { get; private set; } = null;
```

Abbildung 100: Definierung der Verbindungsinstanz

```
//Pfad für Speicherung der Verbindungsdaten
AGL4.SetParaPath(AlarmPreRecMain.projektFolderPath);
//Instanz wird hergestellt um eine Verbindung mit diese Daten möglich zu machen
Ag_connection = AGL4ConnectionFactory.CreateTiaInstance(devNr, plcNr, timeout);
```

Abbildung 101: Instanz zur Verbindung der SPS anlegen

Zunächst wird versucht, eine Verbindung zur SPS aufzubauen und dieses Ergebnis wird in einer boolschen Writevalue gespeichert. Wenn die SPS verbunden ist, so entspricht das Resultat dem boolschen Wert *true*. In Abbildung 102 wird diese Variable abgefragt und je nach ihrem Zustand eine Meldung ausgegeben. Wenn die Verbindung nicht erfolgreich war, so wird die Methode abgebrochen und das Verbindungsfenster nicht geschlossen. Wenn sie erfolgreich war, wird das Verbindungsfenster geschlossen und das Hauptmenüfenster wieder angezeigt.

```
try
{
    bool result = false;
    //Verbindungsaufbau
    result = Ag_connection.Connect();

    if (!result)
    {
        Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("SPS-Verbindung konnte nicht " +
            "hergestellt werden", System.Drawing.Color.Red);

        return false;
    }
    else
    {
        Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("SPS-Verbindung wurde " +
            "erfolgreich hergestellt", System.Drawing.Color.White);
        return true;
    }
}
```

Abbildung 102: Verbindungsaufbau mit der SPS

Falls bei der Verbindung keine SPS erreicht werden konnte, wird eine *exception* geworfen. Die *exception* wird mit einem *catch* aufgefangen. Diese Ausnahme wird als Fehlverbindung interpretiert und der Verbindungsaufbau ist nicht erfolgreich.

7.8.3 Verbindung mit der SPS trennen

Auf die Methode *DisconnectSPS()* kann von jeder Form aus zugegriffen werden. Sie trennt die Verbindung zwischen PC und SPS. Kann diese Verbindung nicht getrennt werden geschieht nichts, Abbildung 103. Wenn noch nie eine Verbindung hergestellt wurde, kann auch die dazugehörige Variable *Ag_connection* nicht vollständig gelesen und nicht auf eine bereits bestehende Verbindung überprüft werden. Dies führt zu einer *exception*, die von einem *catch* Block aufgefangen wird, dieser Block beinhaltet keine Implementierung, da bei keiner bestehenden Verbindung mit einer SPS, diese auch

nicht getrennt werden muss. Demnach kann eine Fehlermeldung im catch-Block vernachlässigt werden, da der Fehler erwartet wird.

```
//Wenn die AGLINK Connection nie aufgerufen wurde können die dazugehörigen Variablen nicht gelesen  
//und somit auch nicht verarbeitet werden  
try  
{  
    if (Ag_connection.Connected)  
        Ag_connection.Disconnect();  
}  
catch  
{  
}
```

Abbildung 103: Trennung der SPS Verbindung

7.9 Videoanzeige mittels Windows Media Player

Um ein Video innerhalb des Benutzerprogrammes abspielen zu können, muss das Framework des *Windows Media Players* in den Form-Designer eingefügt werden. Die Verwendung setzt das Inkudieren des Windows Media Players in Visual Studio voraus. Dies erfolgt, indem der Toolbox weitere Toolboxelemente hinzugefügt werden. Im Menüunterpunkt COM-Komponenten kann der Player hinzugefügt werden, (vgl. Youtube - Leitvideo zum Hinzufügen eines Mediaplayers, 2019), siehe Abbildung 104.

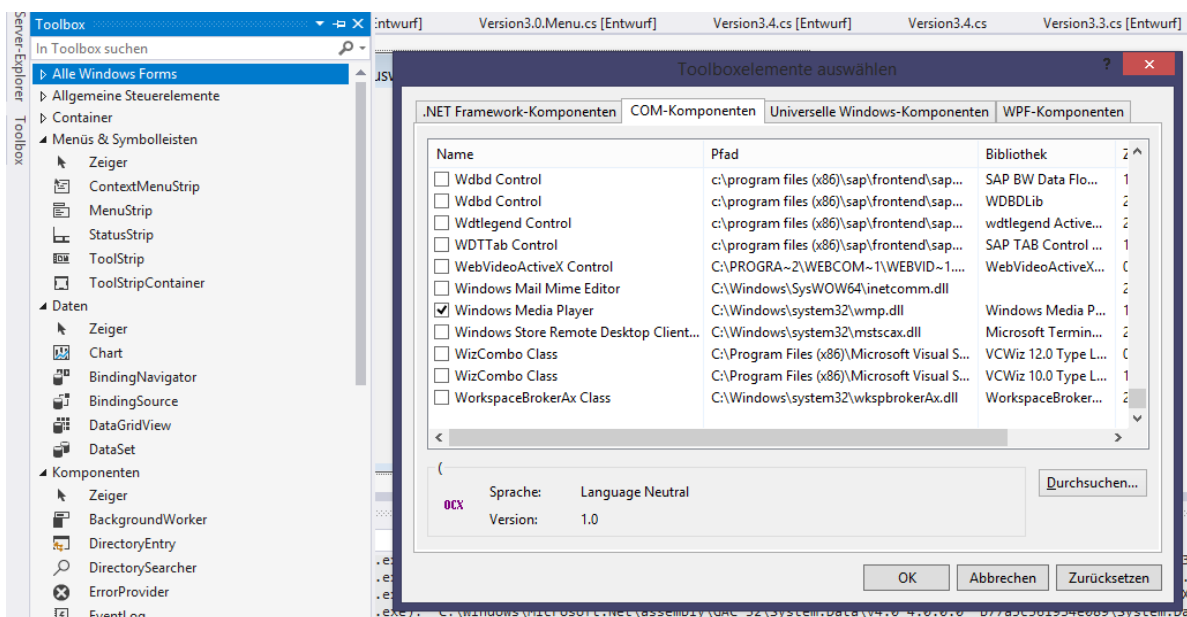


Abbildung 104: Aktivieren des Windows Media Players in Visual Studio

Zunächst werden mit dem „Open“ – Button ein oder mehrere Videofiles ausgewählt, die vom Windows Media Player abgespielt werden können. Hier sucht der *FileOpenDialog* nach einem File des Datentyps *.AVI*. Der *FileOpenDialog* öffnet ein Dialogfenster, das nur die Auswahl eines Verzeichnispfades zulässt. Als Standard-Suchpfad wird hier der Projektordnerpfad verwendet. Falls dieser noch nicht existiert, wird aus dem Verzeichnis „C:/" ausgewählt, siehe Abbildung 105.

Wurde ein Video-File erfolgreich ausgewählt, werden dessen Filename und Filepfad in Attributen einer Klasse gespeichert und in einer *Windows Form Listbox*, im Folgenden als Anzeigeelement bezeichnet, angezeigt. Da mehrere Files auswählbar sind, müssen mittels einer *foreach* Schleife alle Video-Files einzeln eingelesen werden. Daraufhin werden die einzelnen Files in das Anzeigeelement geladen. Aus diesem Element können die einzelnen Files ausgewählt und abgespielt werden, siehe Abbildung 106. Des Weiteren zeigt dieses Anzeigeelement alle Videofiles auf der Form an, die ausgewählt wurden.

```
using (OpenFileDialog ofd = new OpenFileDialog() { Multiselect = true, ValidateNames = true,
    Filter = "AVI|*.avi" })
{
    if (AlarmPreRecMain.projektFolderPath != null)
    {
        //Suchpfad im Projektpfad öffnen
        if (Directory.Exists(AlarmPreRecMain.projektFolderPath + "\\"))
        {
            ofd.InitialDirectory = AlarmPreRecMain.projektFolderPath + "\\";
        }
    }
    else
    {
        ofd.InitialDirectory = "C:\\";
    }
}
```

Abbildung 105: Festlegung des Suchpfades für die Videos

```
if (ofd.ShowDialog() == DialogResult.OK)
{
    List<MediaPlayer> files = new List<MediaPlayer>();
    foreach (string fileName in ofd.FileNames)
    {
        //Auch mehrere Files in der Listbox anzeigbar
        FileInfo fi = new FileInfo(fileName);
        files.Add(new MediaPlayer() { Filename = Path.GetFileNameWithoutExtension(fi.FullName),
            Path = fi.FullName });
    }
    DisplayFile.DataSource = files;
    return;
}
```

Abbildung 106: Einzelne Videofiles in das Auswahlelement einspeisen

7.9.1 Das Suchen eines Videos im Projektordnerverzeichnis

Diese Funktion wurde in der Methode *BrowseVideo_Click()* implementiert. Das Laden der Videos wurde mit einer *foreach*–Schleife realisiert. Der Parameter *fileName* ist ein temporärer Speicher für das abzuarbeitende Video und beinhaltet den Namen des Videos. Es wird jede Variable des Typs *fileName* in der Variable *ofd.FileNames* durchgegangen. *ofd.FileNames* hat alle zuvor ausgewählten Videofiles in einer Liste gespeichert, wie in Abbildung 107 ersichtlich ist.

```
List<MediaPlayer> files = new List<MediaPlayer>();  
foreach (string fileName in ofd.FileNames)
```

Abbildung 107: Abarbeitung der Videoliste

Die Klasse *FileInfo* teilt die Eigenschaften der Variable *fileName* in eigene Untervariablen, siehe Abbildung 108. Diese können mit dem Ausdruck „*fi.*“ + z.B: „*FullName*“ verwendet werden. Somit können einzelne Parameter der einzelnen Videos ausgelesen werden.

```
//Man kann auch mehrere Files in der Listbox anzeigen lassen  
FileInfo fi = new FileInfo(fileName);
```

Abbildung 108: Videoanzeige - Videofiles Informationen speichern

Daraufhin wird der Liste *files* jedes einzelne Videofile hinzugefügt. *files* ist eine Liste des Datentyps *MediaPlayer*. Dieser Datentyp ist eine neu angelegte Klasse, die einen Filenamen und einen Filepfad beinhaltet, siehe Abbildung 109.

```
class MediaPlayer  
{  
    public string Filename { get; set; }  
    public string Path { get; set; }  
}
```

Abbildung 109: Videoanzeige - Klasse MediaPlayer

Die Funktion *add* der Liste *files* fügt ein neu erstelltes Objekt der Klasse *MediaPlayer* mit den Übergabeparametern des Filenamens und des Pfades zu. Daraufhin wird die Datenquelle des Anzeigeelements *DisplayFile* als die Liste *files* festgelegt, siehe Abbildung 110.

```
//Auch mehrere Files in der Listbox anzeigbar
FileInfo fi = new FileInfo(fileName);
files.Add(new MediaPlayer() { Filename = Path.GetFileNameWithoutExtension(fi.FullName),
    Path = fi.FullName });
}
DisplayFile.DataSource = files;
```

Abbildung 110: Videoanzeige - Anzeigeelement mit Daten speisen

7.9.2 Änderung der ausgewählten Videos

Das Anzeigeelement wird beim ersten Laden der Instanz geladen und Änderungen nach dem ersten Laden werden nicht erkannt. Somit muss eine Methode bei der Änderung der angewählten Files getriggert werden. Diese Methode, hier *DisplayFolder_SelectedIndexChanged()*, lädt alle Video-Files neu. Das Auslösen der Methode wird in den Einstellungen des Anzeigeelementes vorgenommen, siehe Abbildung 111.

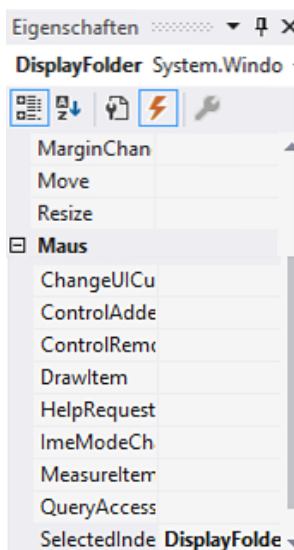


Abbildung 111: Einstellen der Triggermethode DisplayFolder_SelectedIndexChanged

Diese Änderung der angewählten Files kann durch den Benutzer/die Benutzerin geschehen. Die Files werden neu in das Anzeigeelement *DisplayFile* geladen, insofern diese nicht *null*, also gelöscht sind, wie in Abbildung 112 ersichtlich ist.

```
MediaPlayer file = DisplayFile.SelectedItem as MediaPlayer;
if (file != null)
{
    WinMediaPlayer.URL = file.Path;
    WinMediaPlayer.Ctlcontrols.play();
}
```

Abbildung 112: Laden der Videos in das Anzeigeelement

7.10 Vor- und Nachlaufzeiten

Beim Aufrufen der Kamera Vor- und Nachlaufzeiten wird ein *Datatable* angelegt. Diesem *Datatable* wird eine Anzahl an Reihen hinzugefügt, die der Anzahl der Kameras entspricht. Jede Reihe trägt den Namen *Kamera* und den Index der Kamera, zum Beispiel *Kamera 1*. Ebenfalls werden zwei Spalten hinzugefügt, eine für die Kameravorlaufzeiten und die andere für die Kameranachlaufzeiten der jeweiligen Störungsmeldung. Diese Zeiten sind in ganzen Sekunden anzugeben und werden als *string* abgespeichert.

Nach dem Festlegen der Werte des *Datatables* wird diese Tabelle als Quelle für das *DataGridView* auf der Windows Form verwendet. Einem *DataGridView* können nicht dynamisch Reihen und Spalten hinzugefügt und deren Inhalte modifiziert werden, deshalb wird das *Datatable*-System angewendet, welches dieses Verhalten ermöglicht. Nach dem Laden der Reihen und Spalten wird die erste Spalte, der Name der einzelnen Kameras, als *readonly* markiert, da die Änderung des Namens nicht ermöglicht werden dürfen, (vgl. Stackoverflow - Readonly DataGridView, 2019). Die Implementierung ist in Abbildung 113 ersichtlich.

```
DataTable VorNach LaufTable = new DataTable();
VorNach LaufTable.Columns.Add("Kamera", typeof(string));

//dynamisch die Reihen erstellen, je nachdem wie viele Kameras benötigt werden
for (int kameracnt = 1; kameracnt <= Einstellungen.Kameraanzahl; kameracnt++)
{
    VorNach LaufTable.Rows.Add("Kamera " + kameracnt);
}

VorNach LaufTable.Columns.Add("Vorlauf der Kameraaufnahme [ganze Sekunden]", typeof(string));

VorNach LaufTable.Columns.Add("Nachlauf der Kameraaufnahme [ganze Sekunden]", typeof(string));

VorNachDataGridView.DataSource = VorNach LaufTable;
//Die erste Spalte - die Namen der Merkerbits - dürfen nicht verändert werden
VorNachDataGridView.Columns[0].ReadOnly = true;
```

Abbildung 113: Erstellen eines *Datatables* für das *Datagridview*- Element der Vor- und Nachlaufzeiten

Falls beim Laden der Windows Form bereits erkannt wird, dass Vor- und Nachlaufzeiten existieren, so werden diese einzeln in jede Reihe und Spalte hinzugefügt. Hierfür wird eine verschachtelte Schleife verwendet, da dem *DataGridView* kein ganzes Array auf einmal zugewiesen werden kann. Die Vor- und Nachlaufzeiten werden *intern* als *string array* verwaltet. Des Weiteren ist zu beachten,

dass das *DataGridView*-Element in der ersten Spalte den Namen der Kameras speichert. Da dieser nicht veränderbar sein soll und die Vorlaufzeiten in Spalte 2 beginnen, wird diese Spalte übersprungen, dies ist wiederum in Abbildung 114 zu sehen.

```
//Wenn Einstellungen bereits existieren - anzeigen lassen
if (Einstellungen.SPSVorNach LaufArray != null)
{
    for (int rows = 0; rows < Einstellungen.SPSVorNach LaufArray.GetLength(0); rows++)
    {
        //Das Zählen der Spalten beginnt erst ab 1, weil die 1. Spalte eine Überschriftenspalte ist
        //und nicht überprüft werden soll.
        for (int col = 1; col <= Einstellungen.SPSVorNach LaufArray.GetLength(1); col++)
        {
            VorNachDataGridView.Rows[rows].Cells[col].Value = Einstellungen.SPSVorNach LaufArray[rows, col - 1];
        }
    }
}
```

Abbildung 114: Laden der Vor- und Nachlaufzeiten in das DataGridView-Element

7.10.1 Speichern der Vor- und Nachlaufzeiten

In dieser Methode wird jeder einzelne Zellenwert darauf überprüft, ob er eine ganze Zahl ist. Wäre ein Zellenwert keine Zahl, so wird das Speichern abgebrochen und eine Fehlermeldung ausgegeben. Sie signalisiert dem Benutzer/ der Benutzerin, dass der Zellenwert ungültig ist. Für das Überprüfen des Zellenwertes wird die Funktion *TryParse* (vgl. Microsoft - *TryParse*, 2019) verwendet, diese *parsed* den Zellenwert vom Typ *string* in den Typ *Int32* um. Weiters wird bei dieser Funktion angegeben, dass keine besonderen Zahlenkombinationen erwünscht sind. Mögliche Zahlenkombinationen sind positive und negative Zahlen. Für rein positive Zahlen wird keine Zahlenkombination vorgenommen. Würde der Zellenwert aus einem Buchstaben oder Ähnlichem bestehen, liefert die Funktion *TryParse false* als Rückgabeparameter. Die Handhabung dieser Implementierung ist in Abbildung 115 zu sehen.

```
//Zeiten lokal speichern
string[,] VorNach LaufArray = new string[(int)Einstellungen.Kameraanzahl, 2];
int col;
//Überprüfen der Eingabe auf unzulässige Zeichen (nur Zahlen sind zulässig)
for (int rows = 0; rows < VorNachDataGrid.Rows.Count; rows++)
{
    //Das Zählen der Spalten beginnt erst ab 1, weil die 1. Spalte eine Überschriftenspalte ist und
    //nicht überprüft werden soll.
    for (col = 1; col < VorNachDataGrid.Rows[rows].Cells.Count; col++)
    {
        string value = VorNachDataGrid.Rows[rows].Cells[col].Value.ToString();
        VorNach LaufArray[rows, col - 1] = value;

        //Überprüfen, ob nur Zahlen vorhanden sind
        //isNumeric = True - wenn erfolgreich konvertiert wurde
        //isNumeric = False - wenn in 'value' ungültige Zeichen (Buchstaben, Sonderzeichen) vorhanden
        //sind oder dieser leer ist
        //nur positive Zahlen
        bool isNumeric = Int32.TryParse(value, NumberStyles.None, CultureInfo.InvariantCulture,
            out int resultplacebo);

        if (!isNumeric)
        {
            Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Ungültige Eingabe",
                System.Drawing.Color.Red);
            VorNach LaufArray = null;
            return;
        }
    }
}
```

Abbildung 115: Überprüfung der Zellenwerte der Vor- und Nachlaufzeiten

Nachdem eine Zeile vollständig auf ihre korrekten Werte überprüft wurde, wird kalkuliert, ob die Vor- und Nachlaufzeiten dieser Kamera innerhalb der Zeitgrenze von einer Minute liegen. Diese Zeitgrenze wurde vom Auftraggeber vorgegeben. Hier werden die Zellenwerte der Zeile von einem *string* in einen *Int32 geparsed* und addiert. Wenn das Ergebnis größer als 60 Sekunden ist, wird das Speichern abgebrochen und die bereits gespeicherten Werte zurückgesetzt. Zudem wird eine Fehlermeldung ausgegeben, die auf den Fehler hinweist, siehe Abbildung 116.

Wenn jedoch alle Zeiten den Rahmenbedingungen entsprechen, werden die Zeiten im internen Zeitenarray gespeichert und die Einstellungen in das Einstellungen-File ausgelagert. Dieses Auslagern wird in dem Kapitel 7.3 Einstellungen erklärt. Der Weg zum Auslagern der Einstellungen ist in Abbildung 117 ersichtlich.

```
//Zeilenweise max Zeit berechnen - maximal 60 sec
if ((Int32.Parse(VorNachDataGrid.Rows[rows].Cells[col - 2].Value.ToString()) +
    Int32.Parse(VorNachDataGrid.Rows[rows].Cells[col - 1].Value.ToString())) > 60)
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Das Video der Kamera " +
        (rows + 1) + " darf maximal 60 Sekunden lang sein", System.Drawing.Color.Red);
    VorNachLaufArray = null;
    return;
}
}
//Vor und Nachlaufzeiten lokal speichern
Einstellungen.SPSVorNachLaufArray = VorNachLaufArray;
VorNachLauf_Saver();
Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Zeiten wurden erfolgreich" +
    " gespeichert", System.Drawing.Color.LightGreen);
Einstellungen.RefToMainForm.OpenhauptPrgrmStatus();
```

Abbildung 116: Überprüfung der Zeitenlänge pro Kamera

```
if (AlarmPreRecMain.projektFolderPath == null)
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kein Projekt angegeben! " +
        "Erstellen Sie ein neues oder laden Sie ein bereits vorhandenes Projekt", System.Drawing.Color.Red);
    return;
}
Einstellungen.SaveAllSettingsInFile(); //ins File auslagern
```

Abbildung 117: Aufruf der Speichermethode der Einstellungen

7.11 Zuweisung der Meldebits

Die Meldebits, wie auch die Vor- und Nachlaufzeiten, werden in dem *Datatable* zwischengespeichert. Dort werden die einzelnen Reihen und Spalten und deren Inhalte hinzugefügt. Meldebits können erst erstellt werden, wenn die dazugehörigen Datenbits festgelegt wurden.

Der Reihename der Meldungen entspricht dem Meldungsnamen der zuvor eingestellten Datenbits.

Dies ist auch der Grund, warum die Anzahl der Meldungen der Datenbits der Anzahl der Meldungen entspricht, die der Kamera zugewiesen werden müssen. Die Spaltenanzahl entspricht der Anzahl der Kameras, da jede Kamera die Möglichkeit haben soll, für jede Meldung aktiviert zu werden. Die Zuweisung, welche Kamera bei welcher Meldung aktiviert wird, geschieht mittels einer Checkbox. Diese kann bei aktiviertem Zugriff angehakt werden, wie in Abbildung 118 ersichtlich ist

Meldungen	Kamera 1	Kamera 2
Merkerbit1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Merkerbit2	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Merkerbit3	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Merkerbit4	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Abbildung 118: Zuweisung der Meldebits

Durch die Änderung des Datentyps der *Datatable*-Spalte auf *bool* wird automatisch eine Checkbox-Spalte erzeugt.

Nach Erzeugen des *DatagridView*-Elements wird die erste Spalte, der Name der Meldungen, als schreibgeschützt markiert. So wird die unerlaubte Änderung der Meldungsnamen verhindert, wie in Abbildung 119 ersichtlich ist.

```
for (int kameracnt = 1; kameracnt <= Einstellungen.Kameraanzahl; kameracnt++)
{
    Meldungtable.Columns.Add("Kamera " + kameracnt, typeof(bool));
}
WarnungenDataGridView.DataSource = Meldungtable;
//Die erste Spalte - die Namen der Merkerbits - dürfen nicht verändert werden
WarnungenDataGridView.Columns[0].ReadOnly = true;
```

Abbildung 119: Erstellen der Checkboxes und die Spalten als schreibgeschützt festlegen

Die grafischen Einstellungen der Zeilen und Spalten können erst nach der Zuweisung des *DatagridViews* geschehen. Das *DatagridView*-Element muss bereits alle seine tabellarischen Daten besitzen, bevor die Spaltenbreite oder der Zustand der einzelnen Checkboxes eingestellt werden kann. Diese Zuweisung wird als *Bindung des DatagridView – Elements* bezeichnet, siehe Abbildung 120. Das Einstellen der Spaltenbreite ist in Abbildung 121 ersichtlich.

The [DataGridView](#) control supports the standard Windows Forms data binding model, so it can bind to a variety of data sources. Usually, you bind to a [BindingSource](#) that manages the interaction with the data source. The [BindingSource](#) can be any Windows Forms data source, which gives you great flexibility when choosing or modifying your data's location. For more information about data sources the [DataGridView](#) control supports, see the [DataGridView control overview](#).

Abbildung 120: Datagridview-Elemente Binden (vgl. Microsoft - Binden der Datagridview-Elemente, 2019)

```
for (int kameracnt = 1; kameracnt <= Einstellungen.Kameraanzahl; kameracnt++)
{
    WarnungenDataGridView.Columns[kameracnt].Width = 70;
}
```

Abbildung 121: Einstellen der Spaltengröße

Wenn bereits Meldungenzuweisungen existieren, hier *Einstellungen.SPSWarnungsEinstellungArray*, können die geladenen Einstellungen in das *DatagridView*-Element geladen werden. Hier ist zu beachten, dass das *DatagridView*-Element die erste Spalte, den Namen der Meldung, inkludiert. Deswegen muss diese „erste“ / nullte Spalte übersprungen werden, siehe Abbildung 122.

```
foreach (DataGridViewRow row in WarnungenDataGridView.Rows)
{
    for (int col = 1; col < WarnungenDataGridView.ColumnCount; col++)
    {
        try //wennn das Array einen Wert null liefert, so wird ein exception handling aufgerufen
        {
            x = Boolean.Parse(Einstellungen.SPSWarnungsEinstellungArray[row.Index, col - 1]);
        }
        catch (ArgumentNullException)
        {
            x = false;
        }
        if (x == true)
        {
            (WarnungenDataGridView.Rows[row.Index].Cells[col] as DataGridViewCheckBoxCell).Value = x;
        }
    }
}
```

Abbildung 122: Zuweisungen grafisch laden

In den Einstellungen werden die Zuweisungen im Datentyp *string* als *true* oder *false* gespeichert. Diese string-Werte werden in einen boolschen Datentyp geparkt, bzw. umgewandelt. Falls bei dieser Umwandlung eine *exception* geworfen wird, so wird diese erneut mit einem *catch* aufgefangen und der Zellenwert des *DatagridView*-Elements als *false* festgelegt. Wenn der geparkte Wert *true*, also angehakt, ist, wird die Checkbox mit diesem Zustand initialisiert. Der Standardzustand der Checkbox ist *null*, dieser Zustand wird als *false*, also nicht angehakt, interpretiert, weshalb dieser Zustand nicht weiter behandelt wird, siehe Abbildung 122.

7.11.1 Meldungenzuweisungen speichern

Durch einen Button-Klick wird die Speichern-Methode aufgerufen. Diese Methode überprüft jede Zeile und Spalte auf ihren Inhalt und speichert diesen *intern*, sofern er den Rahmenbedingungen entspricht.

Bei der Abarbeitung der Zeilen und Spalten ist zu beachten, dass auch hier die erste Spalte, der Meldungsname, extra berücksichtigt wird. Daher muss diese Spalte

übersprungen werden. In Abbildung 123 ist erkennbar, dass die Zellenwerte als *string* lokal gespeichert werden. Dieser *string* kann auch dem Meldungsnamen entsprechen, weshalb der *string* auf einen boolschen Wert geprüft werden muss. Aufgrund des Zustandes der Checkbox wird einem temporären *stringarray* der *string* hinzugefügt, der den boolschen Zustand darstellt.

```
string CheckedCell = WarnungenDataGridView.Rows[x].Cells[y].Value.ToString();
if (CheckedCell != null)
{
    //nochmal abfragen, weil CheckedCell auch Reihennamen beinhaltet
    if (CheckedCell == "True")
        Warnungarray[x, y - 1] = "True";
    else
        Warnungarray[x, y - 1] = "False";
}
```

Abbildung 123: Zuweisung der Meldungen aus den gespeicherten Einstellungen

Wenn jede Checkbox erfolgreich temporär gespeichert wurde, wird dieser temporäre Array dem eigentlichen Einstellungsarray weitergegeben und die Methode des externen Speicherns aufgerufen. Wenn diese erfolgreich ist, wird eine Erfolgsmeldung ausgegeben, wie in Abbildung 124 erkennbar ist.

```
Einstellungen.SPSWarnungsEinstellungArray = Warnungarray;
//wenn die Kamerameldungen erfolgreich gespeichert werden konnten, dann
if (KameraMeldung_Saver())
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Kameraeinstellungen für" +
        " Meldungen wurden gespeichert", System.Drawing.Color.White);
    Einstellungen.RefToMainForm.OpenhauptPrgrmStatus();
}
```

Abbildung 124: Speichern der Konfigurationswerte im Einstellungsfile

In der Methode des externen Speicherns wird wieder überprüft, ob ein Projektpfad angegeben wurde und die Speichermethode des Programmteils *Einstellungen* des Kapitels 7.3 Einstellungen aufgerufen.

Wenn die Speicherung im Programmteil *Einstellungen.cs* erfolgreich war, wird der Rückgabeparameter *true* zurückgegeben, wenn sie nicht erfolgreich durchgeführt werden konnte, wird eine Fehlermeldung angezeigt und das Speichern abgebrochen, siehe Abbildung 125.

```
//Kameraeinstellungen in File speichern
if (!Einstellungen.SaveAllSettingsInFile())
{
    Einstellungen.RefToMainForm.changeCurrentWorkStatus_Meldungsfeld("Vor- und Nachlaufzeiten konnten" +
        " nicht gespeichert werden", System.Drawing.Color.Red);
    return false;
}
else
    return true;
```

Abbildung 125: Überprüfen des erfolgreichen Speicherns der Zuweisungen

8 Benutzerprogramm - Videoaufnahme starten

8.1 Videoaufnahme

Nachdem die Kameraeinstellungen festgelegt sind und die SPS verbunden ist, ist es möglich sich mit den Kameras zu verbinden. Hierzu wird eine neue Programminstanz geöffnet, die die Verbindungen der Kameras ermöglicht.

Die Videoaufnahme entstammt dem gleichen Grundgedanken wie das Hauptprogramm der *Alarm Pre-Recording* Einstellungen. Hier werden ebenso Panels für das Anzeigen einzelner Windows Forms verwendet. Das Aufrufen der Forms geschieht auch hier mittels einer Menüleiste.

8.1.1 Erstes Laden der grafischen Oberfläche

Beim ersten Laden der Windows Form werden Variablen initialisiert und deren Datenbereiche festgelegt. Des Weiteren wird überprüft, ob die Videoaufnahme bereits geöffnet war. Ist diese Instanz der Videoaufnahme die erste, so wird die Codesequenz in Abbildung 126 ausgeführt.

Zunächst wird die Datengröße des Arrays *KameraXIsConnected* festgelegt und die Anzeigevariablen der Kamera IP-Adressen werden zurückgesetzt. Danach wird die Form zur Verbindungsherstellung mit der Kamera aufgerufen.

```
//Neue Variable anlegen, Verbindungsstatus Kamera
KameraXIsConnected = new bool[(int)Einstellungen.Kameraanzahl];
if (KameraVerbindung.KameraIPAdressen == null)
{
    //IP-Adressengröße anlegen
    KameraVerbindung.KameraIPAdressen = new string[(int)Einstellungen.Kameraanzahl, 3];
    for (int row = 0; row < Einstellungen.Kameraanzahl; row++)
    {
        for (int column = 0; column < 3; column++)
        {
            KameraVerbindung.KameraIPAdressen[row, column] = "";
        }
    }
}
CallConnection();
```

Abbildung 126: Erstladen der Videoaufnahme Programminstanz

Falls die Videoaufnahme bereits geöffnet war, werden andere Funktionen abgearbeitet. Zunächst wird ein leerer *KameraXIsConnected*-Array auf seine Datengröße beschränkt. Wenn dieser bereits Daten beinhaltet, wird mittels einer *for* Schleife

überprüft, ob alle Kameras verbunden sind. Wenn jede Kamera bereits verbunden ist, können die möglicherweise geänderten Vor- und Nachlaufzeiten automatisch übertragen werden. Das Kamerasystem arbeitet dann mit diesen neuen Zeiten.

8.1.2 Vor- und Nachlaufzeiten senden

Sollen die Vor- und Nachlaufzeiten übertragen werden, wird nochmals überprüft, ob jede Kamera verbunden ist. Wenn dies der Fall ist, wird die Form *CallSendFiles* geöffnet und die Möglichkeit des Sendens der Zeiten zur Verfügung gestellt.

8.1.3 Meldezeile

Das Prinzip der Meldezeile entspricht dem Prinzip der Hauptprogramm-Meldezeile, mit dem Unterschied, dass diese nicht alle zehn Sekunden geleert wird. Diese Meldezeile behält ihren Inhalt, da dieser zyklisch vom automatischen Ablauf des Systems aktualisiert wird.

Diese Meldezeile wird ebenso durch mehrere Threads aktualisiert, weshalb auch hier ein *Delegate* mit einem erzwungenen Eingriff durchgeführt werden muss.

Die Methoden *ChangeMeldezeile* und *ClearMeldezeile* ähneln dem Grundprinzip der Hauptprogramm-Meldezeile. Hier wird ebenso das *Delegate* Pattern verwendet. Der Grund hierfür wird nochmals im folgenden Kapitel 8.1.4 Fehler: Cross-Threads erläutert.

8.1.4 Fehler: Cross-Threads

Beim Verwenden des fertigen Kamerasystems mit Meldezeilen tritt bei der Veränderung der Meldezeile ein Fehler auf. Dieser besagt, dass das Benutzerprogramm von verschiedenen Threads verwendet wird und der momentane Zugriff nicht erlaubt werden darf.

Der Grund für diesen Fehler ist, dass der verwendete Timer des Files *Datenbaustein.cs* einen eigenen Thread produziert. In diesem Timer-Thread wird die Methode *changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme* aus dem File *Videoaufnahme.cs* aufgerufen. Diese Methode ändert den angezeigten Text der Meldezeile. Da aber hier ein Thread auf einen anderen zugreifen möchte und dieses

Verhalten nicht akzeptiert wird, wird eine *Exception* geworfen, wie in Abbildung 127 ersichtlich ist.

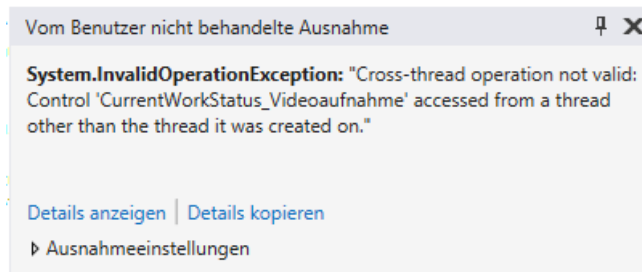


Abbildung 127: Fehlermeldung der Crossthreads

Die *Exception* kann durch eine *Invoke*-Funktion, siehe Abbildung 128, abgewendet werden. Zunächst wird das Control-Element, das Meldungsfenster, abgefragt, ob ein *Invoke* benötigt wird. Dieses *InvokeRequired* überprüft, ob ein erneutes Ausführen des Threads des Control-Elements notwendig ist.

Calls the procedure at the address given by *expression*, passing the arguments on the stack or in registers according to the standard calling conventions of the language type.

Abbildung 128: Invoke (vgl. Microsoft - Invokes, 2019)

Wird die Meldezeile vom Timer aus bearbeitet und somit ein *Invoke* benötigt, wird die betroffene Methode mit einem *Delegate*-Handler gezwungen aufgerufen.

Nachdem die Methode erneut mit dem richtigen Thread aufgerufen wurde, wird der Text der Meldezeile aktualisiert.

Ist kein *Invoke* für das Control-Element notwendig, so wird der Text aktualisiert. Dies geschieht bereits im selben Thread, (vgl. Microsoft - Multithreading, 2019), siehe Abbildung 129.

```
//Erstellen des Delegate Handlers - rekursive Methode
delegate void StringArgReturningVoidDelegate(string currStatus, Color colourforTxt);
public void changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme(string currStatus, Color colourforTxt)
{
    //Abfrage, ob Controll Element aktiviert werden muss
    if (this.CurrentWorkStatus_Videoaufnahme.InvokeRequired)
    {
        //neuer Delegat erstellt und die Methode mit diesem Delegat aufgerufen
        StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(
            changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme);
        this.Invoke(d, new object[] { currStatus, colourforTxt });
    }
    else
    {
        //Kein Invoke benötigt, oder durch Delegat bereits aufgerufen
        CurrentWorkStatus_Videoaufnahme.Text = currStatus;
        CurrentWorkStatus_Videoaufnahme.BackColor = colourforTxt;
    }
}
```

Abbildung 129: Cross Threads - Methodenaufruf - Meldezeile ändert Text

8.1.5 Beenden

Soll das Programm beendet werden, muss zunächst sichergestellt werden, dass keine Kamera momentan aufnehmen möchte oder gerade aufnimmt. Wäre dies der Fall, müsste das Fertigstellen dieser Videoaufnahmen abgewartet werden.

Nimmt momentan keine Kamera auf, so wird der Benutzer/die Benutzerin gefragt, ob das gesamte Programm geschlossen oder zu der Programminstanz der Kamerakonfigurationen zurückgekehrt werden soll. Diese Befragung wird mit einem *DialogResult* durchgeführt, (vgl. Stackoverflow - DialogResult, 2019)

Beim Schließen des Programms werden alle Ressourcen freigegeben, siehe Kapitel 7.2.6 Beenden.

Soll das Programm zu den Einstellungen zurückkehren wird festgelegt, dass die Videoaufnahme bereits offen war und sie nun versteckt wird, das heißt der Zugriff auf die Daten der Programminstanz ist möglich, sie wird aber nicht am Bildschirm angezeigt. Die Kameraaufnahme und alle Timer werden gestoppt und das Hauptprogramm geöffnet.

8.1.6 Zurück zu den Haupteinstellungen

Soll über den Menüunterpunkt *Zurück zu den APR Einstellungen* das Hauptprogramm wieder geöffnet werden, wird auch hier überprüft, dass momentan keine Kamera aufnehmen möchte oder aufnimmt. Wird gerade ein Video aufgenommen, kann das Programm nicht zu den Einstellungen im Hauptprogramm zurückkehren, sondern muss das Fertigstellen des Videos abwarten.

Wenn das Programm bereit ist zu den Einstellungen zurückzukehren, wird der Benutzer/die Benutzerin darauf aufmerksam gemacht, dass die Verbindung mit dem Kamerasystem und somit die gesamte Videoaufnahme unterbrochen wird. Jegliche Datenbitänderung der SPS wird in diesem Zustand nicht aufgenommen und ignoriert. Soll dennoch die Programminstanz der Kamerakonfiguration aufgerufen werden, so werden die Timer und die Kameraaufnahmen unterbrochen. Ebenso wird festgelegt, dass die Videoaufnahme bereits geöffnet war, die Form versteckt und das Hauptprogramm geöffnet werden soll, wie in Abbildung 130 dargestellt ist.

```
//Wenn man zurück zu den Einstellungen möchte, werden die Timer gestoppt, die Verbindung zur Kamera
//existiert noch
//und die Einstellungen werden angezeigt
KameraCntWindowsForm = 0;
//Kameraverbindung muss nicht getrennt werden, weil die Timer gestoppt sind und keine weiteren
//Dateien übertragen werden können
VideoaufnahmeWasAlreadyOpen = true;
Einstellungen.ViewVideoAufnahme.Hide();
AlarmPreRecMain.TimerClearMeldungszeile.Start();
Einstellungen.RefToMainForm.Show();
Datenbaustein.StopTimers();
```

Abbildung 130: Zurückkehren zu den Kamerakonfigurationen

8.1.7 Schließen der gesamten Form

Soll das Programm geschlossen werden, wird auch hier die Kameraaufnahme überprüft und, wenn eine Kamera ein Video erstellt, darauf hingewiesen. Wird nicht aufgenommen, so werden alle Verbindungen, Kameraverbindungen und SPS Verbindung, getrennt und die Ressourcen freigegeben, siehe Abbildung 131.

```
//alles wird geschlossen
for (int i = 0; i < Einstellungen.Kameraanzahl; i++)
{
    if (Videoaufnahme.KameraXIsConnected != null)
    {
        if (Videoaufnahme.KameraXIsConnected[i] == true)
            KameraVerbindung.Disconnect(i);
    }
    ResetData();
    Kamerastatus.TimerRefreshStatus.Dispose(); //Timer disponieren
}
Einstellungen.RefToMainForm.Close();
//Kamera und SPS Verbindung werden im Main-Programm geschlossen
```

Abbildung 131: Schließen des gesamten Programms

8.2 Kameraverbindung

Um eine Verbindung mit den Kameras herstellen zu können, wird die Bibliothek *WinSCP* verwendet. Die Verwendung dieser Bibliothek wird in Kapitel 5.2.4 WinSCP Theorie erklärt.

Die Verbindung der Kameras benötigt Verbindungsparameter, die *SessionOptions*. Hier wird unterschieden, welchen Sicherheitsgrad die Verbindung benötigt oder welches Übertragungsprotokoll verwendet werden soll. Für die Anwendung im Rahmen der Diplomarbeit wurden diese Parameter auf Abbildung 132 synchronisiert, (vgl. Stackoverflow - WinSCP Verwendung, 2019).

```
public static SessionOptions options = new SessionOptions
{
    Protocol = Protocol.Scp,           //Protokoll für Kommunikation mit RaspberryPi ist SCP
    HostName = "",                     //IP-Adresse des Pis
    Password = "",                     //Passwort
    UserName = "",                     //Username
    GiveUpSecurityAndAcceptAnySshHostKey = true //Sicherheitsstufe
};
```

Abbildung 132: Übertragungsparameter der Kameraverbindung festlegen

Das Herstellen einer Verbindung wird als das Öffnen einer Session bezeichnet. Da mehrere Verbindungen/Sessions eröffnet werden sollen, werden die einzelnen Sessions in einer Liste des Datentyps *Session* gespeichert, wie in Abbildung 133 zu sehen ist.

```
//In Liste werden alle Sessions für Kameras gespeichert. Jede Kamera besitzt ihre eigene Session
public static List<Session> sessionList = new List<Session>();
```

Abbildung 133: Speichern der Sessions in Listen

8.2.1 Verbindungsaufbau mit dem Raspberry Pi

Wurden die Einstellungen der Kamera bereits erfolgreich eingegeben und soll eine Verbindung zur Kamera hergestellt werden, wird die Methode *ConnectCamera()* aufgerufen. Hier werden die einzelnen Einstellungen den *SessionOptions* übergeben. Mit diesen Werten wird endgültig eine Verbindung hergestellt.

```
options.HostName = hostnameref;
options.Password = passwordref;
options.UserName = usernameref;
options.TimeoutInMilliseconds = 20000; //nach 20 sec Timeout der Session
```

Abbildung 134: Festlegen der SessionOptions

Danach wird überprüft, ob die *Sessionlist* bereits Sessions enthält. Wenn an dem Index, an dem eine neue Session eröffnet werden soll, bereits eine Session existiert, wird darauf hingewiesen, dass die Kamera bereits verbunden ist. Wenn die Kamera nicht verbunden, die Session somit geschlossen, ist, wird sie aus der Liste entfernt und die neue Session der Liste hinzugefügt und geöffnet, (vgl. WinSCP - Verbindungen, 2019). Die Implementierung ist in Abbildung 135 ersichtlich.

```
Session sessionlocal = new Session();
//Wenn die Liste leer ist, dann kann man sowieso ein Element hinzufügen
//Wenn sie nicht leer ist und der Index der genutzt werden soll bereits Element existiert, wird dies
//zuerst dort gelöscht und dann eingefügt
if (sessionList.Count != 0)
{
    if ((kameracnt + 1) <= sessionList.Count)
    {
        if (Videoaufnahme.KameraXIIsConnected[kameracnt] == true)
        {
            Einstellungen.ViewVideoAufnahme.changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme("Es" +
                " besteht bereits eine Verbindung mit der Kamera", System.Drawing.Color.Yellow);
            return true;
        }
        sessionList.RemoveAt(kameracnt); //Entfernt bereits vorhandene Session beim Index
        //KameracntWindowsForm - macht Platz für neue Session
    }
}
sessionList.Insert(kameracnt, sessionlocal); //fügt eine neue Session in Liste ein
sessionList[kameracnt].Open(options); //Verbinden
```

Abbildung 135: Session eröffnen

Nachdem überprüft wurde, ob die Session geöffnet wurde, wird eine Erfolgsmeldung ausgegeben.

8.2.2 Verbindung mit dem Raspberry Pi trennen

Die Verbindung mit dem Raspberry Pi wird mit der Methode *DisconnectCamera()* getrennt. Soll eine Verbindung getrennt werden, wird diese nach dem Prinzip aus Abbildung 136 erledigt. Konnte die Verbindung nicht getrennt werden oder hat die Verbindung nie bestanden, wird der Fehler mit einem *catch* aufgefangen und ausgegeben.

```
sessionList[KameraCnt].Close(); //Schließen der Verbindung
Einstellungen.ViewVideoAufnahme.changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme("Die Verbindung " +
    "mit der Kamera " + (KameraCnt + 1) + " wurde getrennt", System.Drawing.Color.White);
```

Abbildung 136: Trennen der Verbindung

8.2.3 Verbindung mit dem Raspberry Pi erneut aufbauen

Die Methode *reconnectCamera()* wird aufgerufen, wenn eine Kamera neu verbunden werden soll. Dies muss geschehen, wenn ein File von der Kamera gedownloadet werden soll, diese aber nicht verbunden ist. Bevor eine Fehlermeldung ausgegeben wird, wird nochmals versucht die Kameraverbindung herzustellen.

Die Methode ruft die Methode *ConnectKamera* mit den vorigen Kameraparametern auf und versucht die Verbindung erneut herzustellen.

8.2.4 Dateien auf den Raspberry Pi laden

Diese Methode *UploadFile()* überträgt das zu übertragende File an den Raspberry Pi. Hierzu muss der Stammpfad des Files und der Zielpfad des Files angegeben werden. Es wird hierfür eine binäre Übertragung verwendet, die über die entsprechende Session das File überträgt, dies ist in Abbildung 137 zu sehen.

```
TransferOptions transferOptionsUpload = new TransferOptions();  
transferOptionsUpload.TransferMode = TransferMode.Binary;           //Binärübertragung  
  
TransferOperationResult transferResultUpload;  
transferResultUpload = sessionList[KameraCnt].PutFiles((AlarmPreRecMain.projektFolderPath + File),  
    targetPathRPI, false, transferOptionsUpload);  
  
transferResultUpload.Check();           //wenn nicht übertragen - Error  
return true;
```

Abbildung 137: Datenübertragung auf den Raspberry Pi

Wenn bei der Übertragung etwas schiefgegangen ist, liefert das *transferResultUpload.Check()* aus der Bibliothek *WinSCP* einen Fehler, der mittels *catch* aufgefangen wird. Die Fehlermeldung wird anschließend in der Meldezeile angezeigt.

8.2.5 Dateien vom Raspberry Pi laden

Wie auch beim Upload von Files, wird durch die Methode *DownloadFile()* eine binäre Übertragung gestartet. Mit des Stamm- und Zielpfads kann das zu downloadende File heruntergeladen werden. Hierfür wird die Methode *GetFiles* aus der Bibliothek *WinSCP* verwendet.

8.2.6 Verbindung mit dem Raspberry Pi herstellen und alle Dateien senden

Soll die Verbindung aller Kameras und die Übertragung der Vor- und Nachlaufzeiten der Kameras automatisch erfolgen, wird die Methode *ConnectCameraAllinOne()* aufgerufen. In einer Schleife wird jede Kamera verbunden und nach dem erfolgreichen Verbinden jeder Kamera werden die Vor- und Nachlaufzeiten mit der Methode *alleVorNachZeitensenden* der Form *CameraSendFiles*, übertragen.

8.2.7 Dateien vom Raspberry Pi entfernen

Soll ein File von der Kamera, bzw. dem Raspberry Pi, gelöscht werden, kann dies durch das Aufrufen der Methode *RemoveFile()* ermöglicht werden.

Hierzu wird die Methode *RemoveFiles* aus der Bibliothek *WinSCP* aufgerufen, die das angegebene File vom Raspberry Pi löscht.

8.2.8 Überprüfung, ob der Raspberry Pi verbunden ist

Um die erfolgreiche Verbindung einer Kamera feststellen zu können, wird in der Methode *KameralConnected()* die Existenz einer Datei am Raspberry Pi überprüft. Hierfür wird die Methode *FileExists* aufgerufen und der Pfad des zu prüfenden Files übergeben. Die Existenz des Files wird mittels eines Rückgabeparameters bekannt gemacht. Existiert das File, wird *true* zurückgegeben.

8.3 Verbindungsaufbau mit den Kameras

Um eine Verbindung mit einer Kamera, beziehungsweise einem Raspberry Pi, herzustellen, müssen die IP-Adresse, der Benutzername und das Passwort angegeben werden. Danach wird die Verbindung mit der Kamera erstellt.

Die IP-Adresse wird auf ihre Richtigkeit überprüft. Hierzu wird wieder ein *TryParse* des Datentyps *IPAdress* verwendet. Diese Funktion liefert über einen Rückgabewert Informationen darüber, ob die IP-Adresse der korrekten Form einer IP-Adresse entspricht. Des Weiteren dürfen der Benutzername und das Passwort nicht leer sein. Danach wird die Verbindung zur Kamera hergestellt, siehe Kapitel 8.2 Kameraverbindung.

Wurde die Verbindung erfolgreich hergestellt, werden die Einstellungen der Kamera gespeichert. Die Form zur Verbindung mit der Kamera wird so lange neu geöffnet, bis alle Kameras erfolgreich verbunden wurden. Ebenso werden dann die Einstellungen *extern* gespeichert und die Form der Übertragung der Vor- und Nachlaufzeiten angezeigt.

8.4 Senden der Dateien an den Raspberry Pi

Mögliche Dateien die an den Raspberry Pi gesendet werden können, werden automatisch vom Benutzerprogramm erstellt und auch wieder gelöscht. Die Dateien befinden sich im Projektordner und lauten:

- VorNachLaufZeit.csv
- StartVideoAufnahme.txt
- SendRequestToStoreVideo.csv

8.4.1 Senden der Kameraaufnahmezeiten

Nachdem die Kameras erfolgreich verbunden wurden, müssen die einzelnen Vor- und Nachlaufzeiten an die jeweilige Kamera übermittelt werden, dies geschieht mit der Methode *AlleVorNachZeitensenden()*.

Hierzu wird für jede Kamera eine temporäre *VorNachLaufZeit*-CSV-Datei erstellt Diese beinhaltet eine Zeile und zwei Spalten, die Vorlaufzeit und die Nachlaufzeit, siehe Abbildung 138.

```
//Sobald Kamera die Vor und Nachlaufzeiten und das StartVideoaufnahme File bekommt, arbeitet sie mit
//den überarbeiteten Zeiten - kein Neustart notwendig
for (int kameracnt = 0; kameracnt < Einstellungen.Kameraanzahl; kameracnt++)
{
    //temporär ein File erstellen, das an den RPI gesendet werden kann
    //Die Zeiten werden in eine Zeile, zwei Spalten gespeichert
    StreamWriter outfile = new StreamWriter(AlarmPreRecMain.projektFolderPath + "\\VorNachLaufZeit.csv");
    string content = "";
    for (int y = 0; y < Einstellungen.SPSVorNachLaufArray.GetLength(1); y++)
    {
        content += Einstellungen.SPSVorNachLaufArray[kameracnt, y].ToString() + ",";
    }
    //Versuch, Daten in CSV-File zu schreiben
    outfile.WriteLine(content);
    outfile.Close();
}
```

Abbildung 138: VorNachLaufZeit - File erstellen

Nachdem die *VorNachLaufZeit*-CSV-Datei an die Kamera, bzw. den Raspberry Pi, übertragen wurde, wird das temporäre File gelöscht. Damit die Kamera tatsächlich aktiv ist, muss noch ein *StartVideoAufnahme*-File übertragen werden. Dieses signalisiert der Kamera, dass die Zeiten übertragen wurden und abhängige Variablen erstellt werden können.

Nachdem dieses File erfolgreich übertragen wurde, ist das Kamerasystem aktiv und es wird auf alle ausgewählten Meldungen getriggert.

8.4.2 Videoaufnahme am Raspberry Pi anfragen

Die Methode *SendRequestToStoreVideo()* übergibt der angegebenen Kamera das Trigger-File, das die Aufnahme eines Videos beantragt und beginnen lässt. Existiert bereits eine Anfrage zur Aufnahme, wird die Anfrage nicht erneut übertragen.

8.5 Automatisierter Ablauf - Datenbaustein

8.5.1 Einführung

Dieser automatisierte Ablauf liest sekundlich die Datenbits der SPS ein und lädt, wenn nötig, alle 20 Sekunden ein Video von der entsprechenden Kamera.

Der grobe Ablauf wurde bereits im Kapitel 5 Kamerasystem mit Raspberry Pi und Webcam erklärt, wird aber im Flussdiagramm, siehe Abbildung 139, nochmals genauer erläutert.

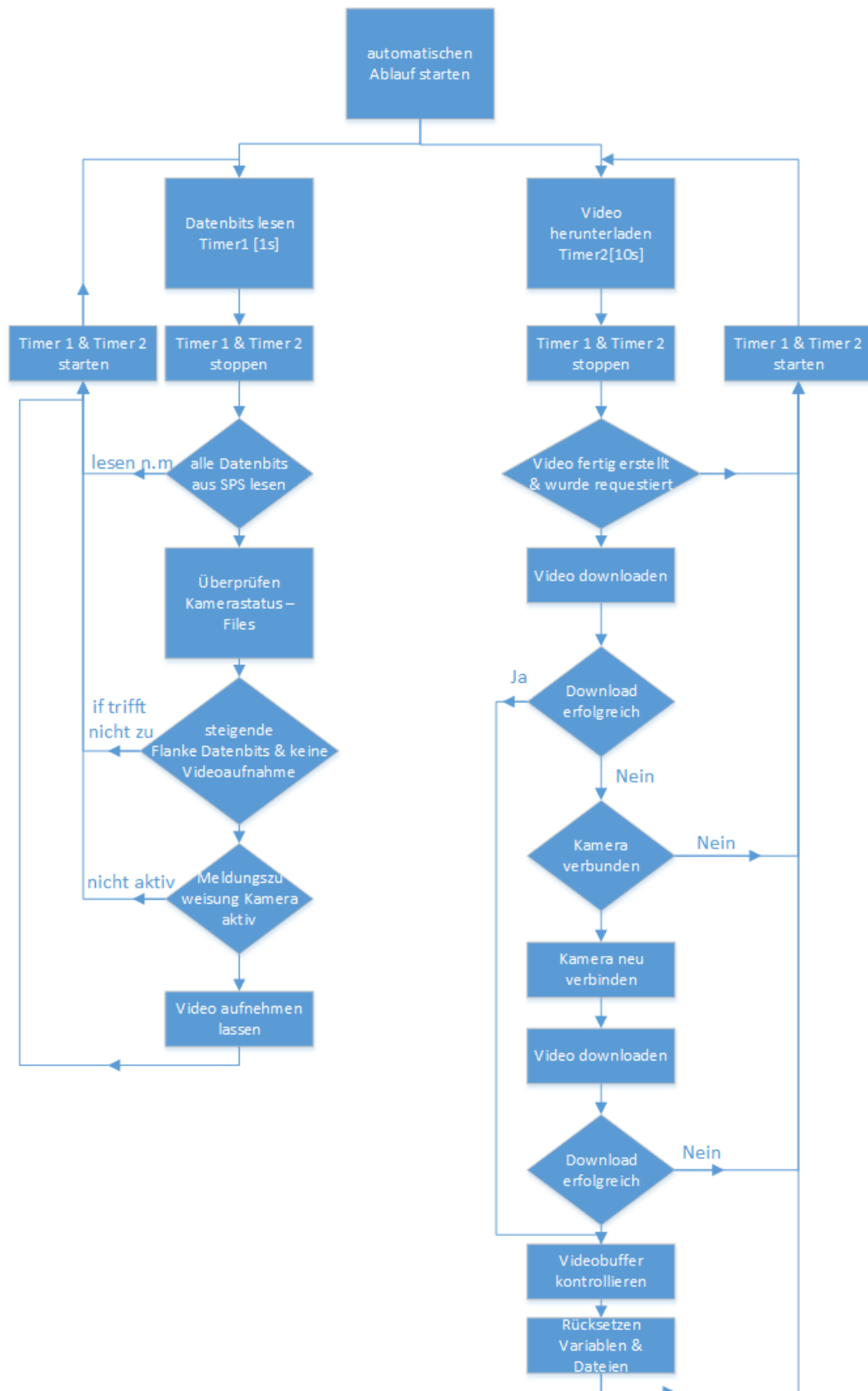


Abbildung 139: Flussdiagramm automatisierter Ablauf Teil 2

8.5.2 Initialisieren des Ablaufes

Für diese Abläufe werden System-Timer verwendet.

Hier werden die Timer mit ihren Event Methoden initialisiert.

Weiters wird auch hier die Größe der Variable der letzten Bitzustände eingestellt. Da die SPS Datenbits gepollt werden müssen, muss überprüft werden, ob die Datenbits eine steigende Flanke aufweisen. Diese Größe der letzten Bitzustände entspricht der Länge der Datenbitzustände. Wenn sich die Länge der Datenbitzustände verändert hat, muss auch die Länge der letzten Bitzustände-Variable geändert werden.

Diese Änderung kann nicht durch die *Resize Array*-Methode der Einstellungen durchgeführt werden, da die Bitzustände keine *strings*, sondern *bool write values* sind und die erstellte Methode nur mit *strings* arbeitet. Daher muss der Veränderungsprozess erneut implementiert werden. Diese Implementierung überprüft, ob die neue Datenbitzustandslänge größer oder kleiner wurde, und verändert demnach die Größe der letzten Bitzustands-Variable, siehe Abbildung 140.

```
if (lastBitValue == null)
{
    lastBitValue = new int[Einstellungen.DBZustandbeiBit.GetLength(0)];
}
else if (lastBitValue.GetLength(0) != Einstellungen.DBZustandbeiBit.GetLength(0))
{
    int[] newarray = new int[Einstellungen.DBZustandbeiBit.GetLength(0)];
    if (lastBitValue.GetLength(0) < Einstellungen.DBZustandbeiBit.GetLength(0))
    {
        for (int j = 0; j < lastBitValue.GetLength(0); j++)
            newarray[j] = lastBitValue[j];
        for (int j = lastBitValue.GetLength(0); j < Einstellungen.DBZustandbeiBit.GetLength(0); j++)
            newarray[j] = 0;
    }
    else if (lastBitValue.GetLength(0) > Einstellungen.DBZustandbeiBit.GetLength(0))
    {
        for (int j = 0; j < Einstellungen.DBZustandbeiBit.GetLength(0); j++)
            newarray[j] = lastBitValue[j];
    }
    lastBitValue = null;
    lastBitValue = newarray;
}
```

Abbildung 140: Verändern der Größe der letzten Bitzustände

8.5.3 Sekündliches Abfragen der Datenbits

Sobald der Timer abgelaufen ist und die Methode aufgerufen wurde, wird der Timer gestoppt. Die Datenbits der SPS werden ausgelesen und wenn das Auslesen erfolgreich war, in den Datenbitarray gespeichert. Sollte das Auslesen nicht erfolgreich

sein, wird eine Fehlermeldung ausgegeben, siehe Abbildung 141. Beim Auslesen muss angegeben werden in welchem Datenbaustein und welchem Datenbyte sich das auszulesende Datenbit befindet. Die Datenbausteinnummer, Datenbytenummer und die Datenbitnummer sind im entsprechenden Datenzustandsarray als *string* gespeichert und müssen demnach noch in einen verwertbaren Integer-Wert umgewandelt werden. Der Bitwert, der aus der SPS gelesen wurde, befindet sich im AGLink Array *rwfield*.

```
//liest die merkerbits jeder Meldung/zeile im Array
int result = ReadBitFromDatenBaustein(0, SPSConnection.timeout, Int16.Parse(
    Einstellungen.DBBitzeichenfolge[i, 1]), Int16.Parse(Einstellungen.DBBitzeichenfolge[i, 2]),
    Int16.Parse(Einstellungen.DBBitzeichenfolge[i, 3]));
if (result == AGL4.AGL40_SUCCESS)
{
    //Speichert den Zustand des Bits in einen Array, welcher später mit den Einstellungen der Kamera
    //verglichen wird.
    Einstellungen.DBZustandbeiBit[i] = rwfield[0].B[0];
}
else
{
    Einstellungen.ViewVideoAufnahme.changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme("Die SPS" +
        " ist nicht verbunden. Die Meldung " + i + " konnte gelesen werden.", System.Drawing.Color.Red);
    TimerCheckIFVideoWasProduced.Start();
    TimerCheckDB.Start();
    return;
}
```

Abbildung 141: Auslesen der Datenbits – sekundlich

Danach werden die Zustände jeder Kamera überprüft. Falls eine Datei auf der Kamera existiert, die hier überprüft wird, so wird die entsprechende Meldung ausgegeben. Die Reihenfolge der Abfrage der Meldungen ist hierbei von Bedeutung, da einige Files immer auf dem Raspberry Pi existieren, diese aber trotzdem abgefragt werden müssen. Sollen nun nur die wichtigsten Files ausgelesen und die entsprechenden Meldungen angezeigt werden, müssen diese in der richtigen *if – else if* Reihenfolge abgefragt werden, ein Beispiel hierfür ist in Abbildung 142 zu sehen.

```
if (KameraVerbindung.FileExists(k, "/home/pi/Documents/FileEditing/AufnehmenMitPolling/" +  
    "VideoFinished.txt"))  
{  
    Videowird_wurde_erstellt[k, 1] = true;  
  
    Einstellungen.ViewVideoAufnahme.changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme("Kamera "  
        + (k + 1) + " hat das Video fertiggestellt.",  
        System.Drawing.Color.LightGreen);  
}  
//Überprüfen, ob bei Video erstellt  
else if (KameraVerbindung.FileExists(k, "/home/pi/Documents/FileEditing/AufnehmenMitPolling/" +  
    "WritingVideo.txt"))  
{  
    Videowird_wurde_erstellt[k, 0] = true; //Rücksetzen der Variable, wenn Video gedownloaded wird  
    Einstellungen.ViewVideoAufnahme.changeCurrentWorkStatus_Meldungsfeld_Videoaufnahme("Kamera "  
        + (k + 1) + " erstellt ein Video",  
        System.Drawing.Color.White);  
}
```

Abbildung 142: Auslesen des Zustands des Raspberry Pis

Nachdem die Zustände ausgelesen wurden, wird auf die steigende Flanke der Datenbitänderung getriggert. Dieses Polling erfolgt durch ein sekundliches Abfragen der Datenbitänderung. Hierbei muss jedes Datenbit einzeln auf seinen Zustand überprüft werden, siehe Abbildung 143.

```
//steigende Flanken Detektion - Wenn Bit steigende Flanke gesetzt wurde, dann Senden Videorequest  
for (int i2 = 0; i2 < (Einstellungen.DBZustandbeiBit.GetLength(0)); i2++)  
{  
    if (Einstellungen.DBZustandbeiBit[i2] != lastBitValue[i2])  
    {  
        lastBitValue[i2] = Einstellungen.DBZustandbeiBit[i2];  
  
        if (Einstellungen.DBZustandbeiBit[i2] == 1)  
        {  
            // ...  
        }  
    }  
}
```

Abbildung 143: Überprüfen der steigenden Bitflanke

Danach wird jede Kamera darauf überprüft, ob bei diesem Meldebit ein Video aufgenommen werden soll. Ist dies der Fall, muss gewartet werden, bis das File *FirstRead* auf dem entsprechenden Raspberry Pi gelöscht wurde. Dieses File zeigt an, dass die Kamera noch nicht ihren Vorlaufpuffer befüllt hat und kein Video aufgenommen werden kann, siehe Abbildung 144 und Kapitel 5 Kamerasystem mit Raspberry Pi und Webcam. Wenn das File vom Raspberry Pi gelöscht wurde, kann eine Videoaufnahme beauftragt werden. Hier wird auch der Zeitpunkt der eingetroffenen Datenbitänderung festgehalten. Dieser Zeitpunkt entspricht der Detektion des Fehlers durch das Benutzerprogramm.

Wenn der Vorlaufpuffer der Kamera befüllt wurde und ein Video angefertigt werden darf, wird dies getan und das AnfrageBit des *Videowird_wurde_erstellt*-Arrays gesetzt.

```
for (int k = 0; k < (int)Einstellungen.Kameraanzahl; k++)
{
    if (Videowird_wurde_erstellt[k, 0] == false) //Man darf nur ein Video für eine Kamera
        //anfordern, wenn diese Kamera nicht gerade ein Video erstellt
        {
            //wenn Meldung gesetzt
            //Array durchgehen - alle Meldungen
            if (Einstellungen.SPSWarnungsEinstellungArray[i2, k].Equals("True"))
            {
                //Rename File nach YYMMDD_HH-mm-ss.avi - momentante Zeit als neuen Videonamen speichern
                //- HH is das 24h Format
                newFileName[k] = DateTime.Now.ToString("yyMMdd_HH-mm-ss");
                //Wenn Firstread existiert ist der Kamerabuffer noch nicht gefüllt und es dürfen noch
                //keine Videos angefordert werden, wenn ein Video gerade geschrieben wird, darf kein
                //neues angefordert werden - nur um sicher zu gehen nochmal abfragen
                while (KameraVerbindung.FileExists(k, "/home/pi/Documents/FileEditing/" +
                    "AufnehmenMitPolling/FirstRead.txt"))
                {
                }
            }
        }
}
```

Abbildung 144: Kameraaufnahme – *FirstRead* abwarten

Wenn die Kamera bereits ein Video aufnimmt, kann kein weiteres Video aufgenommen werden. Eine Warnung wird ausgegeben. Das zuvor bereits aufgenommene Video wird nach dessen Fertigstellung heruntergeladen. Danach kann wieder ein Video auf der Kamera aufgenommen werden. Am Ende der Methode wird der sekundliche Timer wieder gestartet.

8.5.4 Datenbits aus der SPS lesen

Um die Daten aus der SPS lesen zu können, wurden Testfunktionen mit dem mitgelieferten API- Guide, siehe Kapitel 6 AGLINK4 – SPS Verbindung, durchgeführt. Dieser Guide veranschaulicht, dass nur bestimmte Werte von der SPS gelesen werden können. Die Meldebits können nur bitweise ausgelesen werden. Ein zyklisches Auslesen, mittels Interrupt gesteuertem Auslesen ist nicht vorhanden, beziehungsweise wird dies nicht unterstützt. Deswegen müssen die Datenbits gepollt und einzeln ausgelesen werden, (vgl. SPS-Forum - Auslesen der Datenbits, 2019). Gepolltes Auslesen wird durch ein zyklisches Bit-für-Bit Auslesen implementiert.

Um die Daten richtig auslesen zu können, müssen die Bitnummer, sowie die AGLink-Nummer übergeben werden. Weitere Einstellungen wurden von dem API-Guide übernommen, wie der folgenden Abbildung 145 entnommen werden kann.


```
rwfield = new AGL4.DATA_Rw40[1];  
//Festlegen welche PLC SPS gelesen werden soll  
Int32 result = AGL4.AGL40_PARAMETER_ERROR;  
rwfield[0] = new AGL4.DATA_Rw40();  
rwfield[0].BitNr = (ushort)bitnr_uebergabe;  
rwfield[0].DBNr = (ushort)DBnr_uebergabe;  
rwfield[0].Offset = (ushort)offset_uebergabe;  
//Der erwartete Wert für "OpAnz" für ReadMix, WriteMix, OptReadMix oder OptWriteMix ist 1  
rwfield[0].OpAnz = 1;  
rwfield[0].OpArea = AGL4.AREA_DATA;  
rwfield[0].OpType = AGL4.TYP_BIT;  
rwfield[0].Result = 0;  
rwfield[0].B = new Byte[rwfield[0].OpAnz];  
rwfield[0].B[0] = 0;
```

Abbildung 145: Einstellungen API-Guide

Danach wird mittels *ReadMIX* das einzelne Bit ausgelesen. Wenn das Auslesen erfolgreich war, wird dies über den Rückgabeparameter mitgeteilt. Wenn aber ein Fehler auftrat, wird dieser Fehler zurückgegeben und eine Fehlermeldung ausgegeben, siehe Abbildung 146. Das ausgelesene Bit wird in einer einzigen Zelle des *rwfield* gespeichert. Diese wird in der sekundlichen Timermethode ausgewertet und gespeichert.

```
//Liest die Daten vom DB  
result = AGL4.ReadMix(connnr, rwfield, rwfield.Length, timeout);  
if (result != AGL4.AGL40_SUCCESS)  
{  
    // Error  
    String errorMsg = "";  
    AGL4.GetErrorMsg(result, out errorMsg);  
}
```

Abbildung 146: Auslesen der Datenbits

8.5.5 Video downloaden

Diese Methode wird alle 20 Sekunden aufgerufen. Nach Auftreten der Methode werden beide Timer, der sekundliche und 20-sekundliche, gestoppt, da während eines Zugriffes auf den Raspberry Pi kein weiterer Zugriff auf einen anderen Raspberry Pi geschehen darf.

Beim sekundlichen Timer wird das Fertigstellen eines Videos des Raspberry Pis mit dem Setzen des Fertigstellenbits des *Videowird_wurde_erstellt* Array signalisiert.

Wenn das Anfragebit und das Fertigstellenbit beide *true* für den entsprechenden Raspberry Pi sind, wird versucht das Video von dem Raspberry Pi herunterzuladen. Das Herunterladen friert die grafische Oberfläche so lange ein, bis es erfolgreich

durchgeführt werden konnte. Deswegen wird vor dem Download eine Meldung ausgegeben, dass nun gedownloadet wird.

Das Video wird vom Raspberry Pi in den entsprechenden Ordner geladen und mit dem entsprechenden Namen abgespeichert. Der Ordner entspricht dem Kameranamen und der Videoname entspricht dem Zeitpunkt des Setzen des Datenbits, siehe Abbildung 147, (vgl. Stackoverflow - File erstellen mittels Datum, 2019).

```
bool downloadresult = KameraVerbindung.DownloadFile(kameracnt, "/home/pi/Documents/FileEditing/" +  
    "AufnehmenMitPolling/out.avi", ("\\Videosequenz-Kamera" + (kameracnt + 1) + "\\") +  
    newFileName[kameracnt] + "_Kamera" + (kameracnt + 1) + ".avi"), false);
```

Abbildung 147: Download des Videos

Wenn das Video nicht heruntergeladen werden konnte, wird die Verbindung zur Kamera überprüft. Ist die Kamera nicht verbunden, wird diese einmal neu verbunden und der Download erneut versucht, siehe Abbildung 148.

```
if (!KameraVerbindung.KameraIsConnected(kameracnt))  
{  
    KameraVerbindung.reconnectKamera(kameracnt);  
    bool downloadresult_1 = KameraVerbindung.DownloadFile(kameracnt, "/home/pi/Documents/" +  
        "FileEditing/AufnehmenMitPolling/out.avi", ("\\Videosequenz-Kamera" + (kameracnt + 1) +  
        "\\") + newFileName[kameracnt] + "_Kamera" + (kameracnt + 1) + ".avi"), false);
```

Abbildung 148: Kamera neu verbinden

Konnte nach dem Neuverbinden das Video wieder nicht heruntergeladen werden, wird der Download abgebrochen und eine Meldung ausgegeben.

Konnte das Video hingegen erfolgreich heruntergeladen werden, wird dies ebenso mit einer Meldung bekanntgemacht.

Weiters wird jedes Video des Projektordners gelöscht, das älter als das zehnt-jüngste Video ist. Somit wird eine maximale Videoanzahl von zehn im Projektordner erreicht, wie Abbildung 149 entnommen werden kann (vgl. Stackoverflow - Videobuffer, 2019).

```
//Overflow Buffer für Videofiles am PC - es dürfen maximal 10 Files im Ordner entstehen -  
//ältestes File wird gelöscht -  
//überspringt die ersten 11 elemente und löscht dann das 11. weil out.avi noch dazu kommt  
//muss man um 1 inkrementieren  
foreach (var fi in new DirectoryInfo((AlarmPreRecMain.projektFolderPath + "\\Videosequenz-" +  
    "Kamera" + (kameracnt + 1) + "\\").GetFiles()  
    .OrderByDescending(x => x.CreationTime).Skip(10))  
    fi.Delete();
```

Abbildung 149: Ringpuffer der Videos am Leitrechner

Danach werden das Videofile und das Meldefile, das signalisiert, dass das Video fertiggestellt wurde, vom Raspberry Pi gelöscht und die Timer neu gestartet.

8.6 Kamerastatus

Diese Anzeige stellt den Zustand der Verbindung zu den Kameras dar. Die Anzeige wird alle zwei Sekunden mit einem Timer aktualisiert. Dieser Timer wird beim Schließen der Form gestoppt. Die Form entspricht dem Layout einer Tabelle und für jede Kamera wird eine Reihe hinzugefügt. Diese Reihe hat als Spaltenwert den Zustand der Kamera.

In der Methode der Aktualisierung des grafischen Elements muss wieder eine Reference verwendet werden, da das Anzeigeelement nicht statisch, aber die Methode, bzw. dessen Aufruf, statisch ist. Hier werden alle Verbindungen zu den Kameras überprüft und etwaige Fehlermeldungen angezeigt, siehe Abbildung 150.

```
int rows;
for (rows = 0; rows < Einstellungen.Kameraanzahl; rows++)
{
    //Defaultaussehen der Zustände
    Kameradatagridref.Rows[rows].Cells[1].Value = "Gestoppt";
    Kameradatagridref.Rows[rows].Cells[1].Style.BackColor = Color.Red;
    //wenn die Kamera vor dem Klicken des Menüpunktes verbunden ist, wird die jeweilige Farbe angezeigt
    if (Videoaufnahme.KameraXIsConnected != null)
    {
        if (Videoaufnahme.KameraXIsConnected[rows] == true)
        {
            Kameradatagridref.Rows[rows].Cells[1].Value = "Gestartet";
            Kameradatagridref.Rows[rows].Cells[1].Style.BackColor = Color.Lime;
        }
    }
}
```

Abbildung 150: Kamerastatus - Überprüfen des Kamerazustandes

8.7 Videoanzeige mittels Windows Media Player

Die Videoanzeige der erstellten Videos funktioniert über dieselbe Form wie in *Alarm Pre-Recording*-Einstellungen. Es wird durch den Klick auf den Menüunterpunkt eine neue *Videoauswertung*-Form erstellt und diese im Panelsystem angezeigt.

Die Funktionsweise ist im Kapitel 7.9 Videoanzeige mittels Windows Media Player erklärt.

9 Bedienungsanleitung Benutzerprogramm

Beim Start des Programmes wird der Startbildschirm aufgerufen.



Abbildung 151: Startbildschirm

Unter dem Menüunterpunkt *Datei – Projekt erstellen* kann ein neues Projekt erstellt werden.

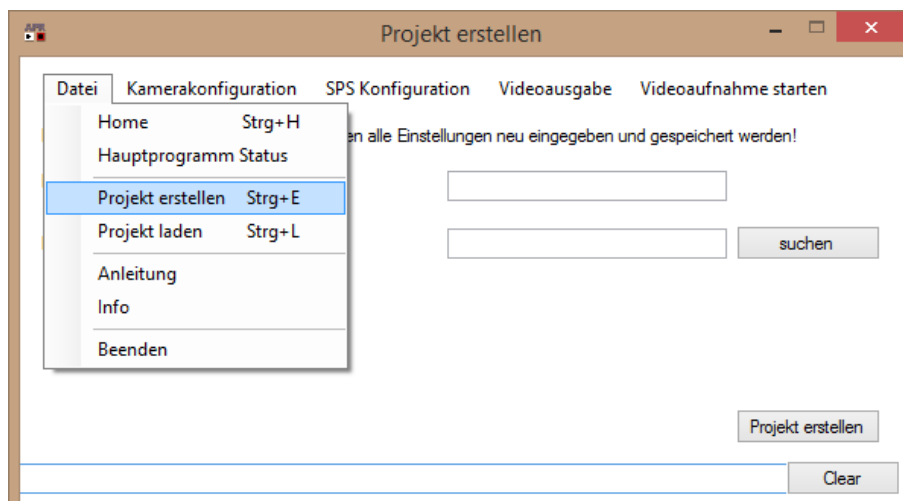


Abbildung 152: Projekt erstellen

Unter *Projektname angeben* kann ein Name für das zu erstellende Projekt angegeben werden. Mit dem Button *suchen* kann ein Projektpfad ausgewählt werden.

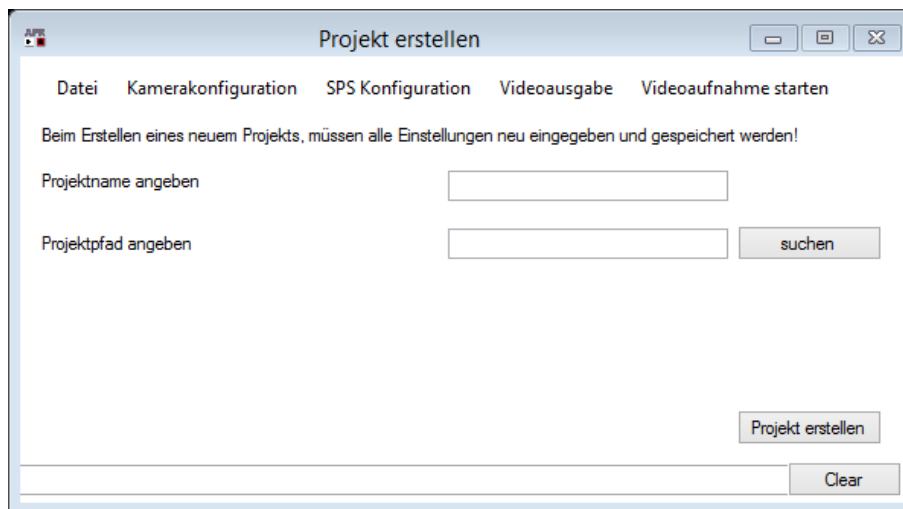


Abbildung 153: Neues Projekt erstellen

Mit dem Button *Projekt erstellen* wird das Projekt erstellt und zur Statusseite des Projektes weitergeleitet.

Wenn ein bereits existierendes Projekt geöffnet werden soll, kann im Menüunterpunkt *Datei – Projekt laden* dieses aufgerufen werden.

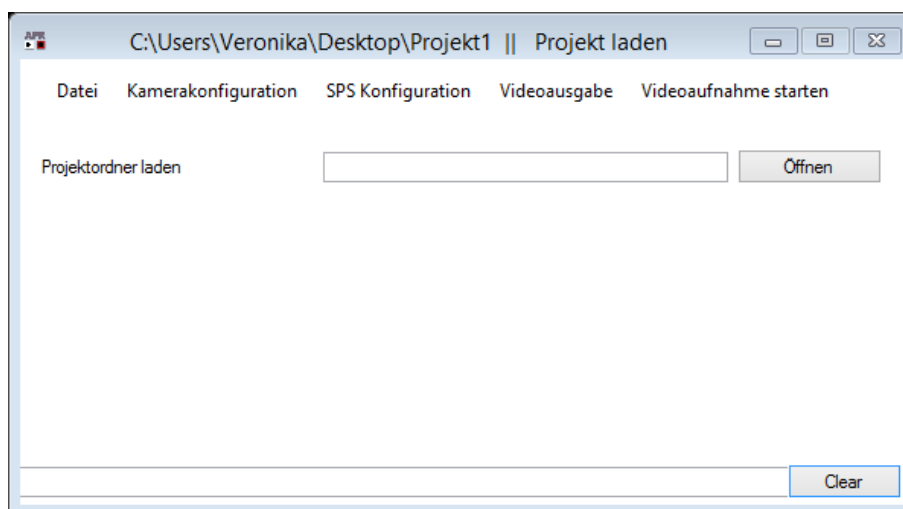


Abbildung 154: Projekt laden

Die Statusseite des Projekts zeigt an, welche Punkte noch bearbeitet werden müssen. (grün: in Ordnung, rot: noch zu bearbeiten). Die Unterpunkte können direkt über die Buttons der Statusseite erreicht werden oder über die Menüpunkte.

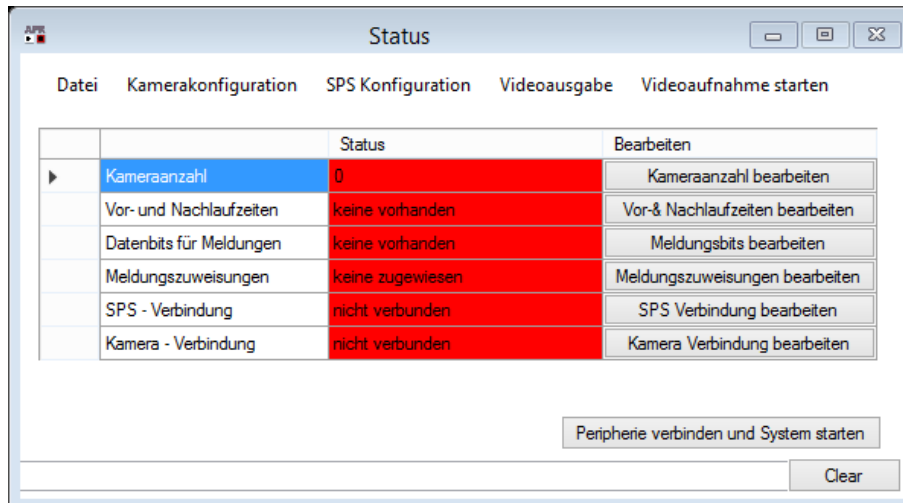


Abbildung 155: unbearbeitete Statusseite

Mit Auswahl des Buttons „Kameraanzahl bearbeiten“ kann die Anzahl der Kameras angegeben werden. Dieses Fenster kann auch durch Auswahl des Menüunterpunkts „Kamerakonfiguration – Anzahl der Kameras“ erreicht werden. Durch Anklicken des Buttons „Speichern“ wird die Anzahl gespeichert und das Projekt kehrt zur Statusseite des Projekts zurück.

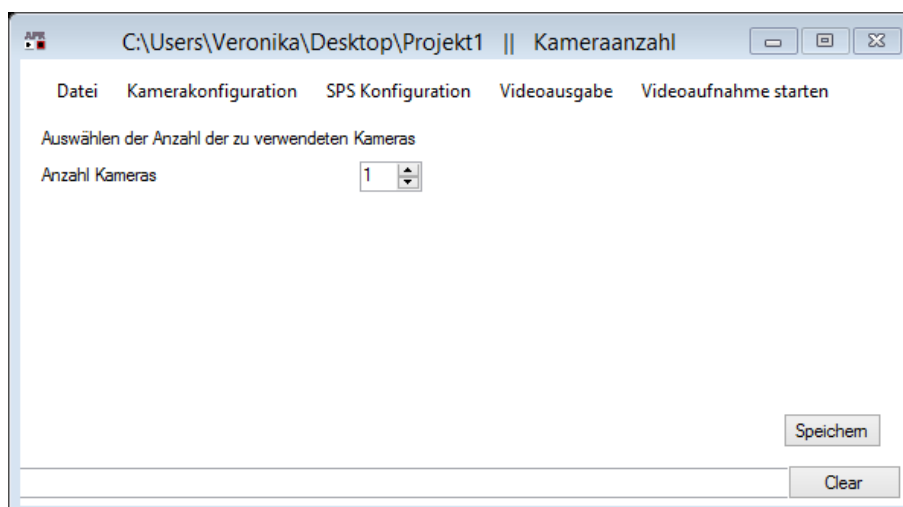


Abbildung 156: Kameraanzahl

Die Statuszeile an der Unterkante des Fensters zeigt an, ob die Kameraanzahl gespeichert werden konnte.

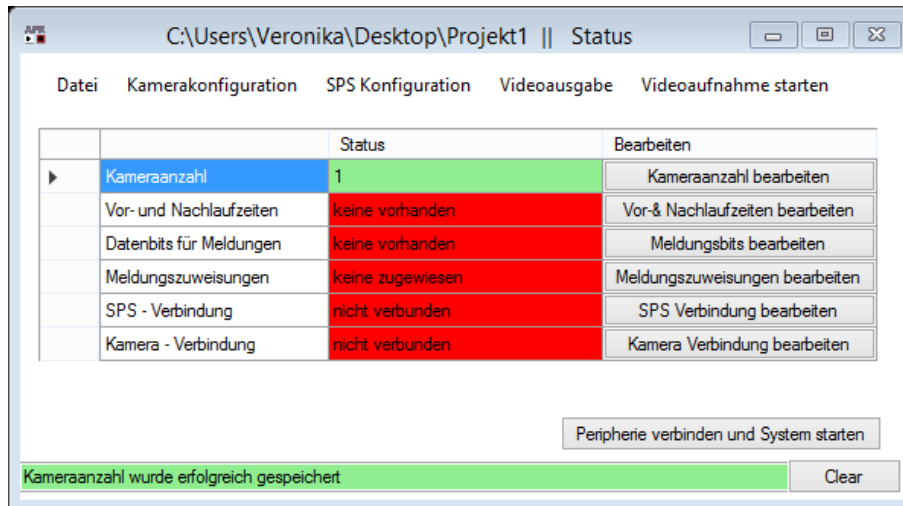


Abbildung 157: Statusseite mit Meldezeile

Durch Auswahl des Buttons „*Vor-&Nachlaufzeiten bearbeiten*“ können eben diese Zeiten eingestellt und durch Anklicken des Buttons „*Speichern*“ gespeichert werden. Das Projekt kehrt wieder zur Statusseite zurück. Das Fenster zum Einstellen der Zeiten kann auch durch den Menüunterpunkt „*Kamerakonfiguration – Vorlaufzeit und Nachlaufzeit der Kamera*“ erreicht werden.

Es ist zu beachten, dass die Vor- und Nachlaufzeit in Summe nur 60 Sekunden betragen kann.

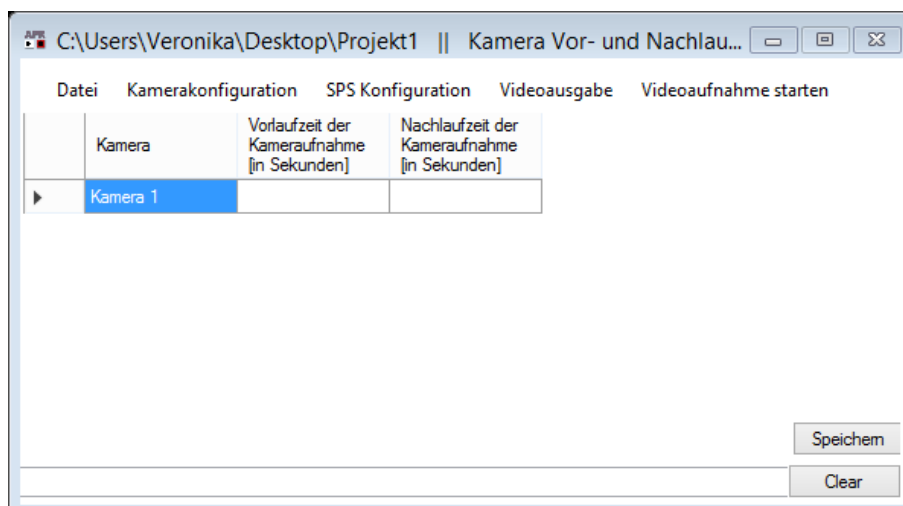


Abbildung 158: Einstellung Vor- und Nachlaufzeiten

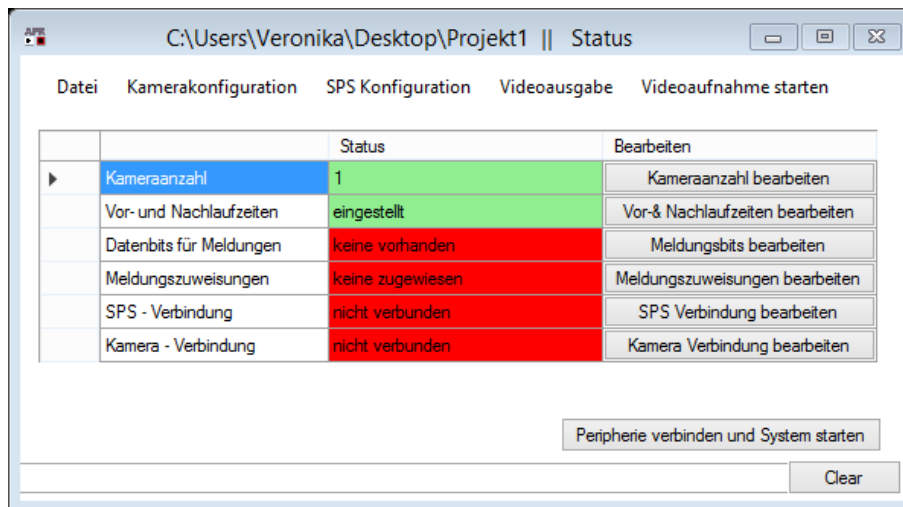


Abbildung 159: Statusseite

Durch Auswahl des Buttons „*Meldungsbits bearbeiten*“ können der Name der Meldung, die Datenbausteinnummer der Fehler, sowie die Byte- und Bitnummer der Fehlermeldungen der SPS angegeben werden. Durch Auswahl des Buttons „*Speichern*“ wird die Auswahl gespeichert und das Projekt kehrt zur Statusseite zurück. Durch Auswahl des Menüunterpunkts „SPS Konfiguration – Datenbausteine“ kann dieses Fenster auch erreicht werden.

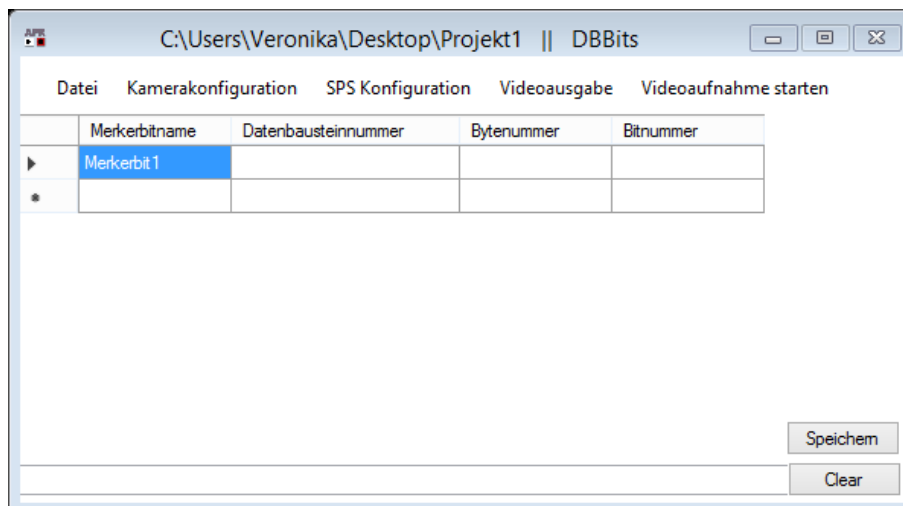


Abbildung 160: Eintragen Datenbausteine

Durch Auswahl des Buttons „*Meldungszuweisungen bearbeiten*“ kann in der Tabelle angehakt werden, welche Kamera bei welcher Meldung ein Video aufnehmen soll. Diese Auswahl kann wiederum mit dem Button „*Speichern*“ gespeichert werden und das Projekt kehrt zur Statusseite zurück. Ein weiterer Weg zum Erreichen dieses Fensters ist die Auswahl des Menüunterpunkts „*Kamerakonfiguration – Kamerazuweisung Meldung*“.

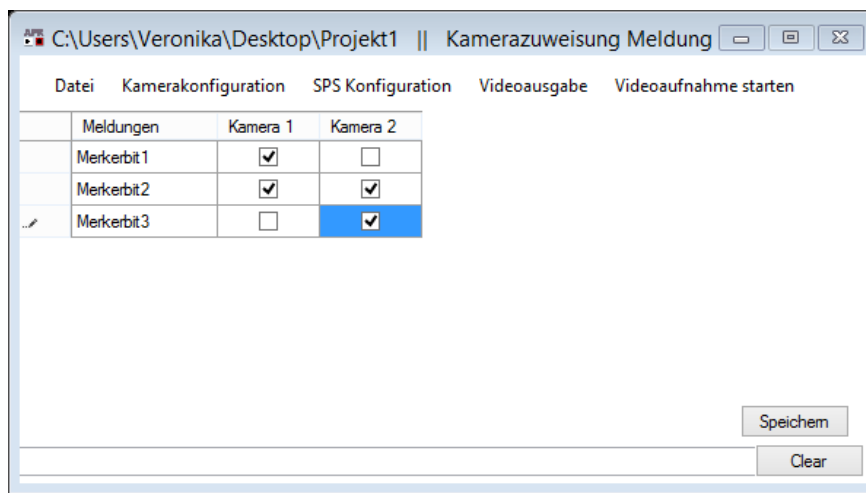


Abbildung 161: Kamerazuweisung Meldungen

Durch Auswahl des Buttons „*SPS-Verbindung*“ oder des Menüunterpunkts „*SPS Konfiguration – SPS Verbindung*“ kann, bei Erstellen eines neuen Projekts, der SPS-Verbindungskonfigurator aufgerufen werden. Dort kann der Verbindungstyp angegeben werden. Weiters können die AG-Nummer, IP-Adresse, Rack, Slot und der Typ der SPS angegeben werden. Unter dem Menüpunkt Test kann die Verbindung mit der SPS getestet werden.

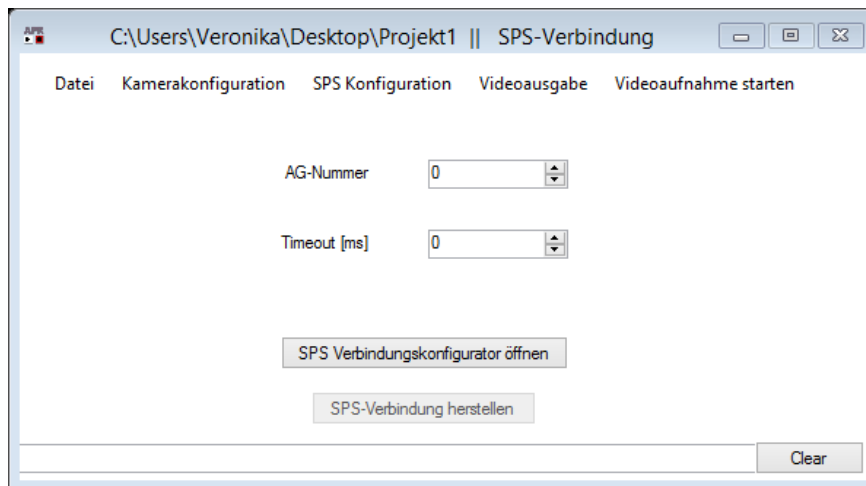


Abbildung 162: SPS-Verbindungs-Konfiguration

Nach Einstellen dieser Parameter kann im Benutzerprogramm eine Verbindung mit der SPS hergestellt werden. Die Verbindung wird durch Anwahl des Buttons „*SPS-Verbindung herstellen*“ hergestellt und das Projekt kehrt zur Statusseite zurück.

Bei einem bereits bestehenden Projekt mit bereits existierenden Parametern kann der Aufruf des Verbindungskonfigurators übersprungen werden.

Durch Auswahl des Buttons „*Kamera Verbindung bearbeiten*“ oder Auswahl des Menüpunkts „*Videoaufnahme*“ und „*Verbindung zu den Kameras – Raspberry Einstellungen*“ können die Daten des Raspberry Pis angegeben und eine Verbindung hergestellt werden. Für eine Verbindung wird die IP-Adresse des Raspberry Pi benötigt, sowie der Benutzername und das Passwort. Durch Anwahl des Buttons „*Weiter*“ wird eine Verbindung mit der Kamera hergestellt. Bei erfolgreicher Verbindung wird dieselbe Seite für die Daten der nächsten Kamera aufgerufen. Wenn bereits alle Konfigurationen angegeben wurden und die Kameras verbunden sind, müssen noch die Vor- und Nachlaufzeiten an die Kamera gesendet werden. Diese Option ist auch über den Menüunterpunkt „*Verbindung zu den Kameras – Vor-Nachlaufzeiten Files senden*“ erreichbar. Wenn diese erfolgreich übertragen werden konnten, beginnt die Kamera mit der Aufnahme.

APR C:\Users\Veronika\Desktop\Projekt1 || Raspberry Pi Einstellungen

Datei Verbindung zu den Kameras Kamerastatus APR Einstellungen Videoausgabe

Kamera 1

IP-Adresse des RaspberryPi der Kamera (z.B: 192.168.1.1)

Benutzername des RaspberryPi

Password des RaspberryPi

Weiter

Clear

Abbildung 163: Raspberry Pi – Einstellungen

Auf der folgenden Seite des Projektes wird der Status der Kameras angezeigt. Diese Seite ist auch über den Menüpunkt „*Kamerastatus*“ erreichbar.

APR D:\Desktop\AlarmPreRecording1 || Kamera Status

Datei Verbindung zu den Kameras Kamerastatus APR Einstellungen Videoausgabe

Kameras	Status
Kamera1	Gestartet
Kamera2	Gestartet
Trigger auf Meldu...	Aktiv

Kamera 1 erstellt ein Video

Clear

Abbildung 164: Kamerastatus

Die Statuszeile an der Unterseite des Projektes zeigt an, wann und von welcher Kamera ein Video aufgenommen wird. Es gibt zudem Auskunft über den Status des Videos.

Mit dem Menüpunkt „*APR Einstellungen*“ kehrt das Projekt zur Statusseite des Projektes zurück. Hierfür müssen jedoch die Verbindungen zu den Kameras getrennt werden. Die Einstellungen des Projektes können geändert werden. Durch Auswahl des Menüpunkts „*Videoaufnahme starten*“ werden die Kameras wieder verbunden.

Durch Anwahl des Menüpunkts „*Videoausgabe*“ können zuvor aufgenommene Videos angezeigt werden. Durch Auswahl des Buttons „*Open*“ kann nach den Videodateien gesucht werden.

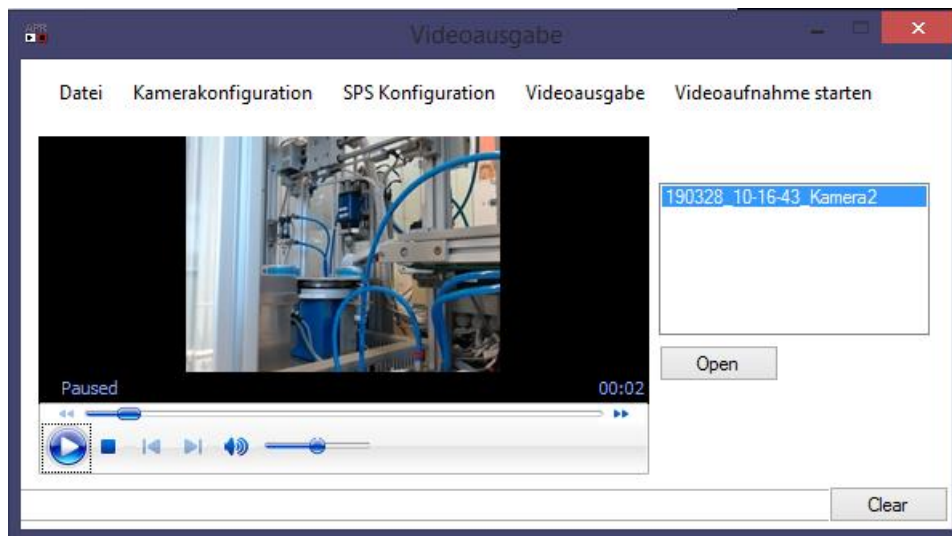


Abbildung 165: Videoausgabe

10 Realaufbau

Der reale Aufbau der Arbeit veranschaulicht die gegebene Problematik genauer. Die zu beobachtende Anlage ist eine ehemalige Diplomarbeit der HTL Pinkafeld. Der Aufbau beinhaltet die SPS, siehe Abbildung 166, die Anlage, siehe Abbildung 167, und das Kamerasystem, welches in Abbildung 168 ersichtlich ist.



Abbildung 166: Realaufbau der SPS



Abbildung 167: Realaufbau der Anlage

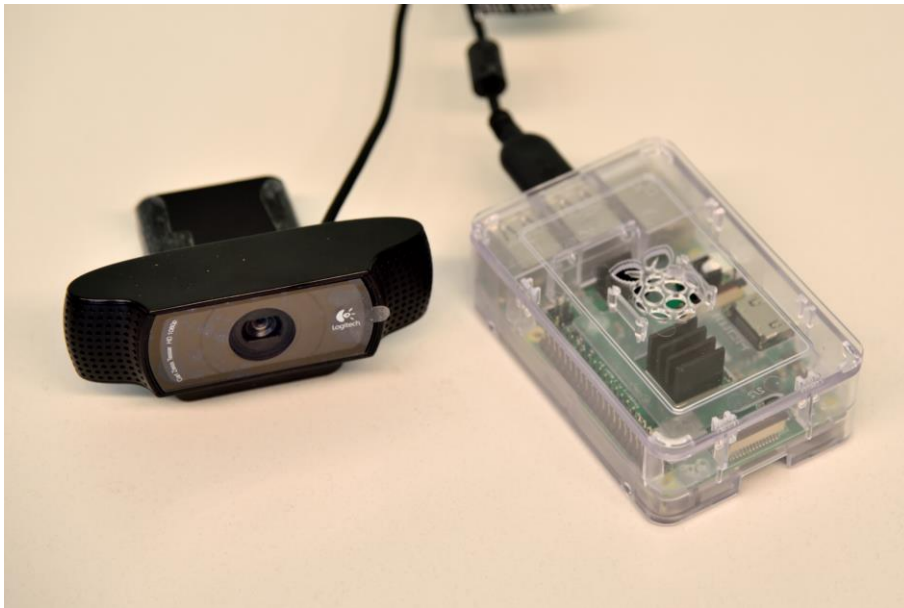


Abbildung 168: Realaufbau des Kamerasystems

Ein fehlerloser Ablauf der Maschine ist in Abbildung 169 ersichtlich. Hier fällt das Produkt passgenau in die Form. Wie in Abbildung 170 erkennbar ist, liegt das Produkt nicht korrekt in der Passform und verursacht eine Fehlermeldung, siehe Abbildung 171.



Abbildung 169: Korrekte Ausführung des Realaufbaues



Abbildung 170: Fehler des Realaufbaues



Abbildung 171: Fehlermeldung des HMI-Panels

Anhand dieser Fehlermeldung erhält das Benutzerprogramm das Signal, eine Videosequenz aufzunehmen. Das Video enthält die Bilddaten vor und nach dem Fehlereintritt und ermöglicht somit einen nachvollziehbaren Rückschluss auf den Fehlerhergang. Durch den veranschaulichten Störhergang ist eine nachhaltige Fehlerbeseitigung möglich.

11 Zusammenfassung

11.1 Ergebnisse

Das Benutzerprogramm wurde als Windows Anwendung in der Programmiersprache C# implementiert und speichert die Kamerakonfigurationen in einem CSV-File. Dieses Benutzerprogramm teilt sich in einen Konfigurationsmodus und einen automatisierten Modus. Die Konfigurationen der Vor- und Nachlaufzeiten der Videoaufnahme einer Störungsmeldung müssen vorgenommen, sowie die Anzahl der Kameras festgelegt werden. Des Weiteren wird hier bestimmt, auf welche Störungsmeldungen das Kamerasystem reagieren soll. Das Kamerasystem funktioniert mit Webcams und einem Raspberry Pi pro Webcam. Die Verbindung zu den Kameras und der SPS wird über das Benutzerprogramm hergestellt, welches auch deren Konfiguration übernimmt. Im automatisierten Modus werden die Daten der SPS zyklisch ausgelesen und verarbeitet. Bei einem Störungsauftritt wird eine Videosequenz aufgenommen und auf dem Raspberry Pi gespeichert. Nach der Fertigstellung des Videos wird dieses vom Raspberry Pi auf den Leitrechner geladen und es wird auf neue Störungsmeldungen reagiert.

Die Diplomarbeit wurde erfolgreich am Beispielaufbau, siehe Kapitel 10 Realaufbau, mit zwei Kameras getestet. Die SPS, die die Störungen in diesem Beispiel meldet, ist eine Siemens S7-1200 SPS.

11.2 Erkenntnisse

Durch die Diplomarbeit konnte viel im Bereich der Selbstständigkeit, der Kreativität und der zeitlichen Organisation gelernt werden. Es mussten verschiedene Ansätze evaluiert werden, welche für die Problemlösung im Bereich dieser Diplomarbeit von Relevanz sind. Des Weiteren musste der zeitliche Rahmen der Diplomarbeit eingehalten werden. Ein weiterer Aspekt, der im Laufe der Diplomarbeit erkannt wurde, ist die Wichtigkeit des klaren Trennens von Teilaufgaben. Hierbei muss beachtet werden, dass ein paralleles Arbeiten nur möglich ist, wenn das klare Trennen und Vordefinieren von Teilaufgaben vorab durchgeführt wird. Des Weiteren konnte durch das Arbeiten an der Diplomarbeit die Teamfähigkeit gestärkt und erkannt werden, wie die Arbeit des Einzelnen von der des anderen abhängig sein kann.

11.3 Zukunftsblick

Im Zuge der Diplomarbeit wurden mögliche Aspekte einer Optimierung ersichtlich. Eine Optimierung des Programmierstils im Bereich der Benutzeroberfläche sowie eine Vereinfachung der Programmstruktur wäre möglich. Des Weiteren kann die Aufnahme der Videosignale verbessert werden. Es wurde ersichtlich, dass durch die Aufnahme des Videosignals der Raspberry Pi einer zu großen Rechenleistung ausgesetzt ist und sich somit eine große, nicht vorgesehene Temperatur am Prozessor entwickelt. Dies könnte umgangen werden, wenn das Kamerasystem mit seinen Komponenten überdacht wird.

Des Weiteren ist es möglich, die Videoaufnahme ohne Raspberry Pis zu erledigen; diese könnte direkt über den PC vorgenommen werden.

Ein weiterer zu optimierender Aspekt ist die Datenspeicherung. Diese könnte in einer Datenbank, anstatt am Leitrechner, stattfinden. Hierbei könnte man auch die Speicherung der Konfigurationen in einem CSV-File überdenken.

Aufgrund finanzieller Mittel war es nicht möglich, das System an vielen Kameras gleichzeitig zu testen. Ebenso wurde es noch nicht im Dauerbetrieb getestet. Das Verhalten des Kamerasystems könnte im großen Stil überwacht werden und Rückschlüsse auf mögliche Daten- und Geschwindigkeitseinschränkungen könnten gezogen werden.

12 Abbildungsverzeichnis

Abbildung 1: Blockschaltbild Aufgabenstellung	12
Abbildung 2: Teilschritte und Meilensteine	15
Abbildung 3: Balkenplan	15
Abbildung 4: Stundenschätzung und Zuweisung in SAP	16
Abbildung 5: Soll/Ist – Abweichung der Kosten	17
Abbildung 6: Soll/Ist - Abweichung der Stundenanzahl.....	17
Abbildung 7: Aufbau.....	18
Abbildung 8: Aufbau der Alarmkamera DS-2CD2125FWD-IS.....	19
Abbildung 9: Das Benutzerprogramm kann über den Webbrowser aufgerufen werden.	20
Abbildung 10: Eventsettings Webbrowser – Aufnahme Kamera1 wird bei Auslösen des Alarmeinangs getriggert.....	21
Abbildung 11: Eventsettings Benutzerprogramm	21
Abbildung 12: Konfigurationsmöglichkeiten.....	22
Abbildung 13: Aufbau Raspberry Pi.....	23
Abbildung 14: Technische Spezifikationen Webcam	24
Abbildung 15: C920 HD Pro Webcam	24
Abbildung 16: Putty – Anmeldung grafische Oberfläche	26
Abbildung 17: Anmeldung WinSCP.....	27
Abbildung 18: Ordneransicht WinSCP	27
Abbildung 19: SessionOptions.....	28
Abbildung 20: Codesequenz zum Übertragen von Daten	29
Abbildung 21: Herunterladen von Daten	29
Abbildung 22: Flussdiagramm Kameraprogramm.....	32
Abbildung 23: CSV-File auslesen.....	33
Abbildung 24: FilePolling.....	34
Abbildung 25: Löschen eines Files	34
Abbildung 26: File erzeugen und schließen.....	34
Abbildung 27: Kamera-spezifische Werte auslesen.....	35
Abbildung 28: Puffer-Array	35
Abbildung 29: Methode <i>copyTo()</i>	35
Abbildung 30: Video erstellen	36
Abbildung 31: Methode <i>resize()</i>	36
Abbildung 32: Servicedatei.....	37
Abbildung 33: Flussdiagramm der Kommunikation mit der SPS.....	39
Abbildung 34: AGLink API-Guide.....	40
Abbildung 35: SPS-Verbindungskonfigurator	40
Abbildung 36: Verbindungstest Verbindungskonfigurator.....	41
Abbildung 37: Flussdiagramm Ablauf – Hauptprogramm.....	45
Abbildung 38: Flussdiagramm - automatisierter Ablauf.....	46
Abbildung 39: Hauptprogramm – Panelansicht	47
Abbildung 40: Menüunterpunkte des Hauptprogramms.....	47
Abbildung 41: Erstellen eines Timers – Hauptprogramm.....	48
Abbildung 42: Starten des Timers – Hauptprogramm	48
Abbildung 43: FormBorderStyle der Form entfernen.....	49
Abbildung 44: Hauptprogrammstatus-Timer stoppen.....	50
Abbildung 45: Dem Panel wird eine Windows Form hinzugefügt	50

Abbildung 46: Überprüfen der Einstellungen zum Öffnen der <i>Videoaufnahme starten</i> Form	51
Abbildung 47: Funktionsbeschreibung <i>changeCurrentWorkStatus_Meldungsfeld</i>	52
Abbildung 48: Erklärung Threads (vgl. Microsoft - Threads, 2019)	52
Abbildung 49: Delegateerklärung (vgl. Microsoft - Delegate, 2019)	52
Abbildung 50: Leeren der Meldezeile mittels Delegate	53
Abbildung 51: Verbindungen zu externen Geräten trennen	53
Abbildung 52: Erklärung <i>References</i> - get; (vgl. Microsoft - References, 2019)	54
Abbildung 53: Reference nichtstatischer Formen	54
Abbildung 54: Schema der Datenspeicherung in einem CSV-File	56
Abbildung 55: Beispiel einer fertigen Einstellungsdatei	56
Abbildung 56: Speichern der Kameraanzahl	57
Abbildung 57: Speichern der Kamera IP-Adressen	57
Abbildung 58: Speichern der Vor- und Nachlaufzeiten	57
Abbildung 59: Speichern der Kamerakonfiguration in Hinsicht auf die IP-Adressen	58
Abbildung 60: Speichern der Meldebits	58
Abbildung 61: Der Speicherungsliste werden die Datenbits hinzugefügt	58
Abbildung 62: Speichern der Konfigurationsdaten in die lokale Einstellungen Datei	59
Abbildung 63: Methode <i>WriteLine</i> (vgl. Microsoft - StreamWriter, 2019)	59
Abbildung 64: Einlesen des Einstellungsfiles	60
Abbildung 65: Festlegen der Spaltenwerte des Einstellungsfiles	60
Abbildung 66: Einlesen der Kamera IP-Adressen	60
Abbildung 67: Zugriffe auf die Menüpunkte festlegen	61
Abbildung 68: Verändern der Größe eines Arrays	62
Abbildung 69: Die Größe der Kamera IP-Adressen verändern	62
Abbildung 70: Projektpfad in der Textbox anzeigen lassen	63
Abbildung 71: Sprung zum Projektpfad im Ordnerverzeichnis	63
Abbildung 72: Anzeigenlassen des momentanen Projektpfades	64
Abbildung 73: Grafische Oberfläche - Projekt erstellen	64
Abbildung 74: Projektname und -Pfad überprüfen	64
Abbildung 75: Projektverzeichnis erstellen	65
Abbildung 76: Erstellen eines Ordners (vgl. Microsoft - Create Directory, 2019)	65
Abbildung 77: Projekt laden - grafische Oberfläche	66
Abbildung 78: Hauptprogrammstatus	66
Abbildung 79: Datagridview-Elemente (vgl. Microsoft - DataGridView-Elemente, 2019)	67
Abbildung 80: Öffnen des Menüunterpunktes - Hauptprogramm-Status	67
Abbildung 81: Grafische Oberfläche - Einstellungen der Kameraanzahl	68
Abbildung 82: Festlegung des Minimums und Maximums der Kameraanzahl	68
Abbildung 83: Überprüfung des Projektpfades bei der Kameraanzahl	68
Abbildung 84: Kameraanzahl speichern	69
Abbildung 85: Aktivierung der Menüunterpunkte	69
Abbildung 86: Kameraanzahl konnte nicht gespeichert werden	70
Abbildung 87: Grafische Darstellung – Meldebits	70
Abbildung 88: Standardeinstellungen der grafischen Oberfläche der Datenbits	70
Abbildung 89: Hinzufügen der Spalten der Datenbits	71
Abbildung 90: Parsen der Zahlenwerte	71
Abbildung 91: Speichern der Meldebits	72
Abbildung 92: Überprüfen ob die AGLINK Einstellungen bereits existieren	72
Abbildung 93: Grafische Oberfläche der SPS-Verbindung	72
Abbildung 94: Öffnen des Verbindungskonfigurators	73

Abbildung 95: Verbindungskonfigurator	73
Abbildung 96: Verbindung zur SPS testen	74
Abbildung 97: AGLINK Einstellungen erfolgreich gespeichert.....	74
Abbildung 98: Fehlermeldung bei keine gültigen AGLINK Einstellungen	74
Abbildung 99: Erfolgreiches Verbinden mit der SPS	75
Abbildung 100: Definierung der Verbindungsinstanz	75
Abbildung 101: Instanz zur Verbindung der SPS anlegen.....	75
Abbildung 102: Verbindungsaufbau mit der SPS	76
Abbildung 103: Trennung der SPS Verbindung	77
Abbildung 104: Aktivieren des Windows Media Players in Visual Studio	77
Abbildung 105: Festlegung des Suchpfades für die Videos	78
Abbildung 106: Einzelne Videofiles in das Auswahlelement einspeisen	78
Abbildung 107: Abarbeitung der Videoliste	79
Abbildung 108: Videoanzeige - Videofiles Informationen speichern	79
Abbildung 109: Videoanzeige - Klasse MediaPlayer	79
Abbildung 110: Videoanzeige - Anzeigeelement mit Daten speisen.....	80
Abbildung 111: Einstellen der Triggermethode DisplayFolder_SelectedIndexChanged	80
Abbildung 112: Laden der Videos in das Anzeigeelement.....	80
Abbildung 113: Erstellen eines Datatables für das Datagridview- Element der Vor- und Nachlaufzeiten.....	81
Abbildung 114: Laden der Vor- und Nachlaufzeiten in das Datagridview-Element	82
Abbildung 115: Überprüfung der Zellenwerte der Vor- und Nachlaufzeiten	83
Abbildung 116: Überprüfung der Zeitenlänge pro Kamera	84
Abbildung 117: Aufruf der Speichermethode der Einstellungen.....	84
Abbildung 118: Zuweisung der Meldebites	85
Abbildung 119: Erstellen der Checkboxen und die Spalten als schreibgeschützt festlegen	85
Abbildung 120: Datagridview-Elemente Binden (vgl. Microsoft - Binden der Datagridview- Elemente, 2019)	85
Abbildung 121: Einstellen der Spaltengröße	85
Abbildung 122: Zuweisungen grafisch laden.....	86
Abbildung 123: Zuweisung der Meldungen aus den gespeicherten Einstellungen.....	87
Abbildung 124: Speichern der Konfigurationswerte im Einstellungsfile	87
Abbildung 125: Überprüfen des erfolgreichen Speicherns der Zuweisungen	88
Abbildung 126: Erstladen der Videoaufnahme Programminstanz.....	89
Abbildung 127: Fehlermeldung der Crossthreads.....	91
Abbildung 128: Invoke (vgl. Microsoft - Invokes, 2019)	91
Abbildung 129: Cross Threads - Methodenaufruf - Meldezeile ändert Text.....	92
Abbildung 130: Zurückkehren zu den Kamerakonfigurationen.....	93
Abbildung 131: Schließen des gesamten Programms	93
Abbildung 132: Übertragungsparameter der Kameraverbindung festlegen.....	94
Abbildung 133: Speichern der Sessions in Listen.....	94
Abbildung 134: Festlegen der SessionOptions	94
Abbildung 135: Session eröffnen	95
Abbildung 136: Trennen der Verbindung	95
Abbildung 137: Datenübertragung auf den Raspberry Pi	96
Abbildung 138: VorNachLaufZeit - File erstellen	98
Abbildung 139: Flussdiagramm automatisierter Ablauf Teil 2	100
Abbildung 140: Verändern der Größe der letzten Bitzustände	101
Abbildung 141: Auslesen der Datenbits – sekundlich	102

Abbildung 142: Auslesen des Zustands des Raspberry Pis.....	103
Abbildung 143: Überprüfen der steigenden Bitflanke	103
Abbildung 144: Kameraaufnahme – <i>FirstRead</i> abwarten.....	104
Abbildung 145: Einstellungen API-Guide	105
Abbildung 146: Auslesen der Datenbits.....	105
Abbildung 147: Download des Videos	106
Abbildung 148: Kamera neu verbinden	106
Abbildung 149: Ringpuffer der Videos am Leitrechner	106
Abbildung 150: Kamerastatus - Überprüfen des Kamerazustandes.....	107
Abbildung 151: Startbildschirm	108
Abbildung 152: Projekt erstellen.....	108
Abbildung 153: Neues Projekt erstellen.....	109
Abbildung 154: Projekt laden	109
Abbildung 155: unbearbeitete Statusseite	110
Abbildung 156: Kameraanzahl.....	110
Abbildung 157: Statusseite mit Meldezeile	111
Abbildung 158: Einstellung Vor- und Nachlaufzeiten.....	111
Abbildung 159: Statusseite.....	112
Abbildung 160: Eintragen Datenbausteine.....	112
Abbildung 161: Kamerazuweisung Meldungen.....	113
Abbildung 162: SPS-Verbindungs-Konfiguration	114
Abbildung 163: Raspberry Pi – Einstellungen	115
Abbildung 164: Kamerastatus	115
Abbildung 165: Videoausgabe	116
Abbildung 166: Realaufbau der SPS.....	117
Abbildung 167: Realaufbau der Anlage	117
Abbildung 168: Realaufbau des Kamerasystems	118
Abbildung 169: Korrekte Ausführung des Realaufbaues	118
Abbildung 170: Fehler des Realaufbaues.....	119
Abbildung 171: Fehlermeldung des HMI-Panels.....	119

13 Literaturverzeichnis

- vgl. *Github*. (15. März 2019). Von https://github.com/cggos/dip_cvqt/issues/1 abgerufen
- vgl. *HIKVision*. (10. Oktober 2018). Von <https://www.hikvision.com/de/Support/Downloads/Client-Software> abgerufen
- vgl. *Logitech Support*. (15. März 2019). Von https://support.logitech.com/de_de/product/hd-pro-webcam-c920/specs abgerufen
- vgl. *Microsoft - Binden der DataGridView-Elemente*. (27. März 2019). Von Online: <https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-bind-data-to-the-windows-forms-datagridview-control> abgerufen
- vgl. *Microsoft - Create Directory*. (05. März 2019). Von Online: <https://docs.microsoft.com/en-us/dotnet/api/system.io.directory.createdirectory?view=netframework-4.7.2> abgerufen
- vgl. *Microsoft - DataGridView-Elemente*. (27. März 2019). Von Online: <https://docs.microsoft.com/de-at/dotnet/framework/winforms/controls/datagridview-control-windows-forms> abgerufen
- vgl. *Microsoft - Delegate*. (26. März 2019). Von Online: <https://docs.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/delegates> abgerufen
- vgl. *Microsoft - Invokes*. (27. März 2019). Von Online: <https://docs.microsoft.com/en-us/cpp/assembler/masm/invoke?view=vs-2017> abgerufen
- vgl. *Microsoft - Multithreading*. (05. März 2019). Von Online: <https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-make-thread-safe-calls-to-windows-forms-controls> abgerufen
- vgl. *Microsoft - Panels*. (26. März 2019). Von Online: <https://docs.microsoft.com/de-de/dotnet/api/system.windows.forms.panel?view=netframework-4.7.2> abgerufen
- vgl. *Microsoft - References*. (26. März 2019). Von Online: <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/get> abgerufen
- vgl. *Microsoft - StreamWriter*. (05. März 2019). Von Online: <https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=netframework-4.7.2> abgerufen
- vgl. *Microsoft - Threads*. (26. März 2019). Von Online: <https://docs.microsoft.com/de-de/dotnet/standard/threading/threads-and-threading> abgerufen
- vgl. *Microsoft - TopLevel*. (05. März 2019). Von Online: <https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.form.toplevel?view=netframework-4.7.2> abgerufen
- vgl. *Microsoft - TryParse*. (05. März 2019). Von Online: https://docs.microsoft.com/en-us/dotnet/api/system.decimal.tryparse?redirectedfrom=MSDN&view=netframework-4.7.2#System_Decimal_TryParse_System_String_System_Decimal__ abgerufen
- vgl. *Microsoft - Windows Forms*. (26. März 2019). Von Online: <https://docs.microsoft.com/de-de/dotnet/api/system.windows.forms?view=netframework-4.7.2> abgerufen
- vgl. *OpenCV*. (16. März 2019). Von <https://opencv.org/> abgerufen
- vgl. *Raspberry Projects*. (15. März 2019). Von <https://raspberrypi-projects.com/pi/programming-in-c/file-input-and-output/working-with-files> abgerufen
- vgl. *Raspberry Tips*. (15. März 2019). Von <https://raspberrypi-tips.einsteiger/raspberry-pi-autostart-von-skripten-und-programmen-einrichten> abgerufen
- vgl. *SPS-Forum - Auslesen der Datenbits*. (05. März 2019). Von Online: <https://www.sps-forum.de/simatic/94531-datenbausteine-inklusive-deren-inhalt-auslesen-wie-mit-aglink-post709754.html#post709754> abgerufen

- vgl. *Stackify - Exception Handlings*. (05. März 2019). Von Online:
<https://stackify.com/csharp-exception-handling-best-practices/> abgerufen
- vgl. *Stackoverflow - Arraygrößen verändern*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/6539571/how-to-resize-multidimensional-2d-array-in-c> abgerufen
- vgl. *Stackoverflow - CSV Files speichern*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/18806757/parsing-csv-file-into-2d-array> abgerufen
- vgl. *Stackoverflow - DialogResult*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/3036829/how-do-i-create-a-message-box-with-yes-no-choices-and-a-dialogresult> abgerufen
- vgl. *Stackoverflow - File erstellen mittels Datum*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/5046686/how-do-i-create-a-file-based-on-the-date> abgerufen
- vgl. *Stackoverflow - Ordner auswählen*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/7330111/select-folder-path-with-savefiledialog> abgerufen
- vgl. *Stackoverflow - Readonly DataGridView*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/11282729/how-to-make-a-specific-column-uneditable-in-datagridview> abgerufen
- vgl. *Stackoverflow - Videobuffer*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/19949459/delete-oldest-files-in-directory/19949505> abgerufen
- vgl. *Stackoverflow - WinSCP Verwendung*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/28027074/winscp-c-sharp-first-use> abgerufen
- vgl. *Stackoverflow - Zugriff nichtstatischer Methoden*. (05. März 2019). Von Online:
<https://stackoverflow.com/questions/19894162/how-to-call-non-static-method-on-form1-from-form2> abgerufen
- vgl. *Water Programming*. (15. März 2019). Von
<https://waterprogramming.wordpress.com/2017/08/20/reading-csv-files-in-c/> abgerufen
- vgl. *WinSCP - Verbindungen*. (05. März 2019). Von Online:
https://winscp.net/eng/docs/task_connections abgerufen
- vgl. *WinSCP*. (15. März 2019). Von <https://winscp.net/eng/docs/lang:de> abgerufen
- vgl. *Youtube - Leitvideo zum Hinzufügen eines Mediaplayers*. (05. März 2019). Von Online:
<https://www.youtube.com/watch?v=K7l1OGHNfeE> abgerufen
- vgl. *Youtube - Panels erstellen*. (05. März 2019). Von Online:
<https://www.youtube.com/watch?v=qOQj2VE6di4> abgerufen
- Abbildung 8:
10729_FFTPspec2SeriesG1(2XX5supportH.265plus)Baseline21SeriesNetworkDomeCamera QuickStartGuide.pdf – Seite 13/14
- Abbildung 13: https://en.wikipedia.org/wiki/Raspberry_Pi
- Abbildung 14:
https://support.logitech.com/en_ch/product/hd-pro-webcam-c920/specs 4.3.2019 14:25
- Abbildung 33: *ACCON-AGLink Grundlagen_HB_de.pdf*