



Studienarbeit

PC-Remote-Controlled Car

Schlussbericht



Roman Baumann
Lukas Schwendener

Betreuer: Prof. Dr. G. Schuster

1 Vorwort

Der Bereich Bildverarbeitung steht nicht im Lehrplan. Wir haben uns trotzdem für eine Arbeit aus diesem Gebiet entschieden, um einen Einblick zu erhalten. Da uns die Thematik gezwungenermassen unbekannt war, wussten wir zu Beginn noch nicht genau, was uns erwarten würde.

In der Zwischenzeit haben wir uns in die Kantendetektion, ein Teilgebiet der Bildverarbeitung, eingearbeitet. Das Resultat ist der vorliegende Bericht. Dieser richtet sich an Studenten, welche ebenfalls eine Arbeit in Bildverarbeitung gewählt haben sowie an interessierte Laien. Der Bericht ist so formuliert, dass auch sie die Grundlagen verstehen können. Es war darum unser Ziel, diese Dokumentation so übersichtlich wie möglich zu gestalten.

Bei den vertiefenden Kapiteln werden gewisse Kenntnisse vorausgesetzt; sie sind für den fachlich versierten Leser gedacht.

Nebst der Bildverarbeitung bestand unsere Hauptarbeit darin, mit der zur Verfügung stehenden Hard- und Software zurechtzukommen. Namentlich waren das ein als *PCMCIA*-Karte realisierter D/A-Konverter und eine Wireless-Cam. Beides steuern wir mit *Visual C++* an.

Bemerkung zum Titel:

Wir sind ursprünglich davon ausgegangen, dass unser Modellauto nicht kontinuierlich fahren kann. Vielmehr würde es ein Hüpfen werden. Wie ein Frosch also (engl. FROG). Nun fährt es doch kontinuierlich, folglich heisst es jetzt Advanced FROG...

2 Inhaltsverzeichnis

1	Vorwort	2
2	Inhaltsverzeichnis	3
3	Aufgabenstellung	5
3.1	Kurzbeschreibung	5
3.2	Ausführliche Beschreibung	5
4	Ausgangslage	6
4.1	Resultate aus früheren Arbeiten	6
4.2	Unsere Hardware	6
4.3	Informationsfluss	6
5	Theorie zur Kantendetektion	8
6	Realisierung	10
6.1	Wahl des Algorithmus zur Kantendetektion	10
6.2	Randprobleme	11
6.3	Ergebnisse der Kantendetektion	11
6.4	Bilderfassung	12
7	Messungen am Modell	13
8	Die GUIs	15
8.1	Das GUI <i>Untitled - FROG</i>	15
8.1.1	Zweck	15
8.1.2	Funktionalität	15
8.1.3	Umsetzung	16
8.2	Das GUI <i>FrameGrabber</i>	16
8.2.1	Zweck	16
8.2.2	Funktionalität	16
8.2.3	Umsetzung	17
8.3	Das GUI <i>Position</i>	17
8.3.1	Zweck	17
8.3.2	Funktionalität	17
8.3.3	Umsetzung	17
8.4	Das GUI <i>FROG</i>	18
8.4.1	Zweck	18
8.4.2	Funktionalität	18
8.4.3	Umsetzung	19
9	Zeitplan	20
9.1	Theoretische Vorarbeiten	20
9.2	Praktische Arbeiten	20
9.3	Kommentar zu den praktischen Arbeiten	20
9.3.1	Bezug Laborarbeitsplatz	20
9.3.2	Organisation der Hard- und Software	20
9.3.3	Auto mittels DAC-Karte ansteuern	20
9.3.4	Bild von Kamera in C++ einlesen	21
9.3.5	Algorithmen und GUI implementieren	21
9.3.6	Test und Optimierung	21
9.3.7	Dokumentation	21
10	Schlussfolgerungen	22
10.1	Was wir erreicht haben	22
10.2	Probleme	22

10.3	Erkenntnisse.....	22
10.4	Vorschläge für zukünftige Arbeiten.....	22
10.5	Dank.....	23
11	Anhang.....	24
11.1	Anhang A1 – <i>Untitled FROG</i>	24
11.1.1	Ansteuerung der DAC-Karte in Visual C++.....	24
11.2	Anhang A2 - <i>FrameGrabber</i>	25
11.3	Anhang A3 - <i>Position</i>	27
11.3.1	Kontrollmechanismus I – <i>störende Pixel</i>	27
11.3.2	Kontrollmechanismus II – <i>Strassenbreite</i>	28
11.3.3	Kontrollmechanismus III – <i>abgeschnittene Strasse</i>	28
11.3.4	Kontrollmechanismus IV – <i>andere Kanten</i>	29
11.3.5	Kontrollmechanismus V – <i>Übertragungsstörung</i>	30
11.4	Anhang A4 - <i>FROG</i>	32
11.4.1	Material Check.....	32
11.4.2	Inbetriebnahme.....	32
11.4.3	Betrieb.....	32
11.4.4	Stoppen und Ausschalten.....	32
11.4.5	Störungsbehebung.....	33
12	Listings.....	35
12.1	Listing L1 – <i>Untitled FROG</i>	35
12.1.1	Headerfile <i>FROGView.h</i>	35
12.1.2	Implementationsfile <i>FROGView.cpp</i>	37
12.2	Listing L2 – <i>FrameGrabber</i>	41
12.2.1	Headerfile <i>FrameGrabberView.h</i>	41
12.2.2	Implementationsfile <i>FrameGrabberView.cpp</i>	43
12.3	Listing L3 – <i>Position</i>	50
12.3.1	Headerfile <i>PositionDlg.h</i>	50
12.3.2	Implementationsfile <i>PositionDlg.cpp</i>	52
12.4	Listing L4 – <i>FROG</i>	63
12.4.1	Headerfile <i>FROGDlg.h</i>	63
12.4.2	Implementationsfile <i>FROGDlg.cpp</i>	65

3 Aufgabenstellung

In diesem Kapitel beschreiben wir die grundlegenden Anforderungen an unsere Arbeit.

3.1 Kurzbeschreibung

Ein Modellauto soll mit Hilfe einer Kamera selbständig einem Strassenverlauf folgen.

3.2 Ausführliche Beschreibung

Das Modellauto ist ein handelsüblicher Bausatz. Die Kamera wird mittels Klettband auf der Frontscheibe befestigt und von einer 9V-Batterie gespeist. Das Bild wird per Funk auf eine Basisstation übertragen. Dann wird es auf einem Laptop verarbeitet und schliesslich ein Steuersignal für die Fernbedienung des Autos erzeugt.

Der Strassenverlauf ist durch ein 2cm breites Malerклеbeband gekennzeichnet. Dabei sollte der Kontrast zwischen dem Boden und dem Klebeband nicht zu klein sein.

Ein weiteres Kriterium stellt die Geschwindigkeit dar. Das Endziel ist eine kontinuierliche Fahrt. Aufgrund der Hard- und Software werden wir vermutlich nur ein ruckartiges Fahren zustande bringen.

Unsere tatsächliche Aufgabe besteht daraus, das Bild von der *USB*-Schnittstelle einzulesen und es mit einem geeigneten Algorithmus zu verarbeiten. Anschliessend wird das Steuersignal über einen Digital-Analog-Wandler ausgegeben. Probleme wie die Funkübertragung von und zum Auto gehören nicht zu unserer Aufgabenstellung.

4 Ausgangslage

Hier erläutern wir, was an Hardware und Informationen bereits vorhanden ist.

4.1 Resultate aus früheren Arbeiten

Diese Aufgabe wurde bereits früher im Rahmen einer Studienarbeit umgesetzt. Damals wurde der Algorithmus in Matlab implementiert. Dabei hat sich gezeigt, dass diese Variante zu langsam ist. Deshalb werden wir unsere Arbeit in *Visual C++* implementieren.

Die Erkenntnisse der früheren Arbeit dienten uns als Grundlage, vor allem in Bezug auf die Kantenerkennung.

4.2 Unsere Hardware

Wireless-Cam inkl. Empfänger und Grabber (*Xray Vision* von *XI0*)

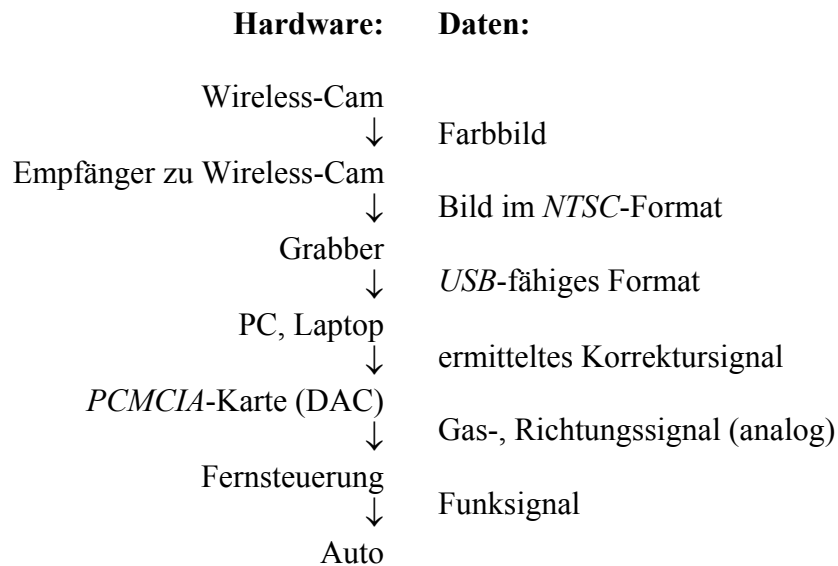
Fernsteuerung

Modellauto (Bausatz von *Kyosho*)

Laptop *Pentium II* 400

DA-Wandler in Form einer *PCMCIA*-Karte (*DAQCard-AO-2DC* von *National Instruments*)

4.3. Informationsfluss



Dieser Informationsfluss stellt einen Regelkreis dar:

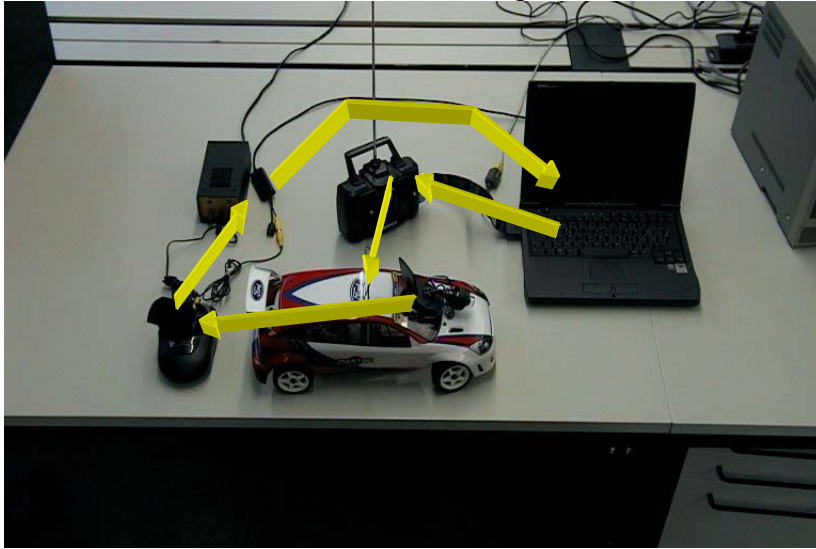


Abbildung 1 - Regelkreis

5 Theorie zur Kantendetektion

In diesem Kapitel beschreiben wir die Grundlagen der Kantendetektion anhand des Sobel-Algorithmus'.

Bei der Bildverarbeitung geht es darum, Objekte auf einem Bild zu erkennen. Dafür ist es notwendig, die Kanten zu detektieren. Unter einer Kante versteht man einen grossen Helligkeitsunterschied.

Die Kantendetektion sei hier am Beispiel des Sobel-Operators und eines Schwarz-Weiss-Bildes erklärt:

Um die Kanten zu detektieren wird das Bild, welches in Matrizenform vorliegt, mit dem Sobel-Operator gefaltet. Der Operator wird dafür über die ganze Matrize verschoben. An jeder Position werden die Werte des Operators mit den entsprechenden Werten des Bildes multipliziert und anschliessend aufsummiert. Der so errechnete Wert gilt jeweils für das Pixel des Bildes, welches in der Mitte des Operators liegt. Siehe dazu Gleichung 1.

Die Pixel am Rand des Bildes erfordern eine besondere Behandlung (Kapitel Randprobleme, Seite 11).

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Abbildung 2 - Die Sobel-Operatoren in x/y-Richtung

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	-1*0	0*0	1*0	0	0	0
0	0	0	-2*0	0*1	2*1	1	1	1
0	0	0	-1*0	0*1	1*1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1

zu bewertendes Pixel

Abbildung 3 – Beispiel einer Kantendetektion mit dem Sobel-Operator in x-Richtung

$$G_x(x, y) = (-1) \cdot P(x-1, y-1) + (-2) \cdot P(x-1, y) + (-1) \cdot P(x-1, y+1) \\ + 0 \cdot P(x, y-1) + 0 \cdot P(x, y) + 0 \cdot P(x, y+1) \\ + 1 \cdot P(x+1, y-1) + 2 \cdot P(x+1, y) + 1 \cdot P(x+1, y+1)$$

Gleichung 1 - Gradient G_x

Der errechnete Wert stellt nun den Gradienten G_x in x-Richtung dar. Um ein vollständiges Kantenbild zu erhalten, muss der Betrag des Gradienten G bestimmt werden. Dazu ist der Gradient G_y in y-Richtung ebenfalls zu berechnen. G wird dann nach Gleichung 2 bestimmt:

$$G = \sqrt{G_x^2 + G_y^2}$$

Gleichung 2 - Gradient G

In Abbildung 4 ist das Resultat der Kantendetektion zu sehen. Dabei zählen alle Pixel mit einem Wert grösser null zur Kante. Je grösser der Wert, desto deutlicher ist die detektierte Kante.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	3	4	4	4	4
0	0	0	3	4	4	4	4	4
0	0	0	4	4	0	0	0	0
0	0	0	4	4	0	0	0	0
0	0	0	4	4	0	0	0	0
0	0	0	4	4	0	0	0	0

Abbildung 4 – Werte nach der Kantendetektion (gerundet)

Tatsächlich wird das Farbbild der Kamera zuerst in ein Graustufenbild umgewandelt. Es kann dann genau wie das Schwarz-Weiss-Bild gefaltet werden, mit dem Unterschied, dass bei der Faltung mit einem 8-Bit Graustufenbild bedeutend höhere Zahlenwerte entstehen können.

Die ausführliche Theorie ist in der Studienarbeit "*Digitale Ecken- und Objekterkennung*" von Manuel Meier und Christian Rüegg dargelegt.

6 Realisierung

Der Abschnitt Realisierung beinhaltet einen Vergleich von drei verschiedenen Algorithmen und weiter Erkenntnisse und Probleme der Kantendetektion.

6.1 Wahl des Algorithmus zur Kantendetektion

Für die Auswahl eines geeigneten Algorithmus zur Kantendetektion haben wir uns auf die Resultate der eben genannten Studienarbeit gestützt. Wir haben dabei folgende Erkenntnisse gewonnen:

Der Canny-Algorithmus fällt ausser Betracht, da seine Berechnung zu lange dauert (in Matlab implementiert 20 Sekunden). Auch der Laplace-Algorithmus stellte sich als ungeeignet heraus, da er zu rauschempfindlich ist. Somit blieben noch die drei Algorithmen Sobel, Roberts und Prewitt. Nach einer ersten Betrachtung haben wir uns für den Roberts-Operator entschieden. Dieser hat den Vorteil, dass er mit einer 2×2 -Matrix arbeitet. Die anderen beiden Operatoren benützen eine 3×3 -Matrix und sind deshalb rechenintensiver, das heisst langsamer.

Da die Strassenlinie für unser Modellauto im Bild mehr oder weniger senkrecht verläuft, hatte unsere Vorgängergruppe den Sobel-Algorithmus gewählt. Allerdings verwendeten sie nur den Operator in x-Richtung. Auch wir haben den Sobel-Operator in x-Richtung verwendet. Trotz der etwas grösseren 3×3 -Matrix erhofften wir uns dadurch einen Geschwindigkeitsvorteil, weil der Gradient aus den beiden 2×2 -Matrizen des Roberts-Algorithmus nicht mehr berechnet werden muss.

Nachfolgend ist ein Vergleich der Operatoren zu sehen:

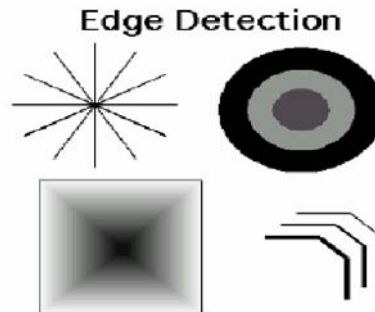


Abbildung 5 - Originalbild

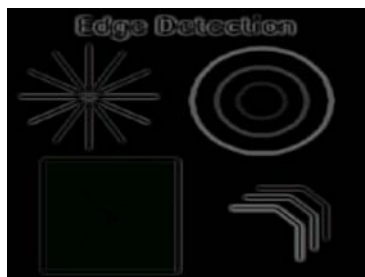


Abbildung 6 - Roberts-Algorithmus

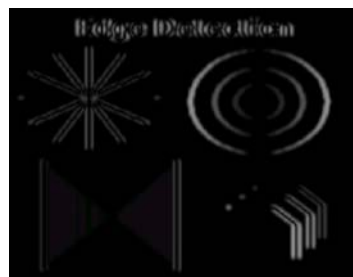


Abbildung 7 - Prewitt-Algorithmus in x-Richtung



Abbildung 8 - Sobel-Algorithmus in x-Richtung

Diese Bilder können mit dem *Matlab* Programm *kante.m* der Vorgängergruppe erstellt werden.

Der Unterschied zwischen dem Roberts-Algorithmus und den anderen beiden Algorithmen ist klar ersichtlich. Dies ist jedoch darauf zurückzuführen, dass der Roberts-Algorithmus mit beiden Operatoren ausgeführt wurde.

Eine Differenz zwischen dem Sobel- und dem Prewitt-Algorithmus ist auf diesen Bildern nicht zu erkennen. Aufgrund der grösseren Gewichtung der Matrize des Sobel-Operators erhoffen wir uns jedoch von diesem bessere Resultate.

Aus der Dokumentation unserer Vorgänger geht hervor, dass das Bild unabhängig vom Algorithmus mit einem Gauss-Filter zu bearbeiten sei, damit ein optimales Resultat erzielt wird.

Unsere Tests haben ergeben, dass der Sobel-Algorithmus zusammen mit der in Anhang A3 (Seite 27) beschriebenen Fehlerverhinderung auch ohne Gauss-Filter genügend gut funktioniert.

6.2 Randprobleme

Bei der Faltung der Bildmatrize mit dem jeweiligen Operator werden nebst dem zu bewertenden Pixel auch dessen direkte Nachbarn benötigt. Da die Randpixel weniger Nachbarn haben als Pixel im Innern des Bildes, können sie nicht gleich behandelt werden. Für unsere Anwendung spielen sie aber keine wesentliche Rolle. Deshalb weisen wir ihnen einen neutralen Wert zu.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	-1*0	0*0	1*0	0	0	0
0	0	0	-2*0	0*1	2*1	1	1	1
0	0	0	-1*0	0*1	1*1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1

Abbildung 9 zeigt ein 9x9-Matrixdiagramm zur Darstellung von Randproblemen bei der Faltung. Die Matrix enthält Werte von 0 bis 1, wobei die Randpixel (0) und die inneren Pixel (1) dargestellt sind. Ein Pfeil zeigt auf die linke Seite der zweiten Zeile, beschriftet mit 'Randpixel'. Ein weiterer Pfeil zeigt auf die rechte Seite der zweiten Zeile, ebenfalls beschriftet mit 'Randpixel'. Ein dritter Pfeil zeigt auf die rechte Seite der sechsten Zeile, beschriftet mit 'zu bewertendes Pixel'.

Abbildung 9 – Probleme mit den Randpixeln

6.3 Ergebnisse der Kantendetektion

Da unsere Strasse aus einem Malerклебband besteht, wird das Resultat der Kantendetektion aus zwei Linien bestehen. Dies hatte zur Folge, dass wir für die Auswertung entweder eine Linie vernachlässigen oder aber den Mittelwert der beiden Linien ermitteln mussten.

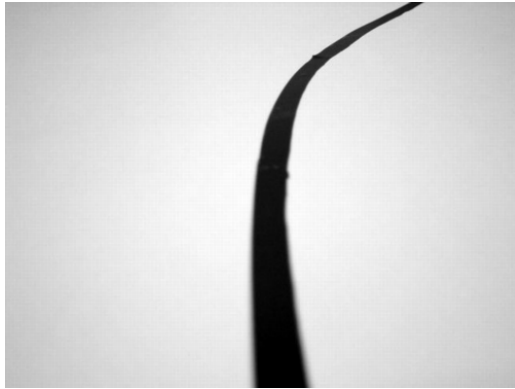


Abbildung 10 – Originalbild (schwarze Strasse)

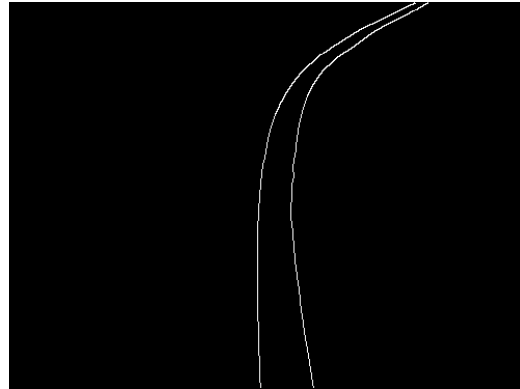


Abbildung 11 - bearbeitetes Bild

6.4 Bilderfassung

Zur Bilderfassung wurde die unter 4.2 beschriebene Wireless-Cam eingesetzt. Diese ist von unseren Vorgängern mit Klettband auf der Frontscheibe befestigt worden. Um eine bessere Positioniergenauigkeit zu erreichen, hatten wir uns überlegt, die Kamera auf ein Blech zu montieren. Dieses hätten wir dann bei den Karosseriehalterungen befestigen können. Bei genauerem Hinsehen bringt diese Lösung aber keine Nennenswerten Vorteile, da der Kamerakopf beweglich ist und somit die gewonnene Starrheit wieder zunichte gemacht würde.

Wir hatten uns daher entschieden, bis auf weiteres mit der Version der Vorgänger zu arbeiten. Um trotzdem eine gute Ausrichtung der Kamera zu ermöglichen, haben wir ein Testbild verwendet.

Mit einer Lampe könnten die unterschiedlichen Lichtverhältnisse ausgeglichen werden. Sollte diese Lösung einmal notwendig werden, müsste auf das Blech zurückgegriffen werden. Unsere Tests haben aber gezeigt, dass es auch ohne Lampe funktioniert.

7 Messungen am Modell

Bei der Fernbedienung kann wahlweise manueller oder PC-Betrieb eingestellt werden.

Im manuellen Betrieb wird die Steuerspannung von einem Potentiometer erzeugt. Beim PC-Betrieb hingegen werden die Signale von aussen eingespeist. Für das Lenksignal haben wir einen Spannungsbereich von 2.15V (links) bis 3.01V (rechts) gemessen. Die Neutralstellung liegt bei einer Spannung von 2.64V. Beim Geschwindigkeitssignal verhalten sich die Spannungen ähnlich. Der Bereich geht von 2.23V (vorwärts) bis 2.93V (rückwärts). Die Neutralstellung liegt hier bei 2.62V. Für den Radeinschlag haben wir folgende Spannungen gemessen:

Spannung in V	Radeinschlag in Grad
3.01	20 rechts
2.8	10 rechts
2.6	0
2.4	10 links
2.2	20 links

Tabelle 1 - Spannungswerte beim Radeinschlag

Somit ergibt sich die folgende Formel:

$$\text{Spannung} = 0.02 \frac{\text{V}}{\text{Grad}} \cdot \text{Radeinschlag} + 2.6\text{V}$$

Wir haben die *DAQCard* zuerst direkt an die Fernbedienung angeschlossen. Dabei hat es jeweils die Steuerspannung zusammengerissen (mit Digitalmultimeter und mit KO gemessen). Folglich mussten wir diese Problematik genauer betrachten.

Die Leerlaufspannung am Ausgang der *DAQCard* entsprach den in unserem Testprogramm eingestellten Werten.

Wenn die Fernbedienung auf PC-Betrieb gestellt ist, liegt an den Eingängen eine Gleichspannung von 2.63V. Zusätzlich enthält sie eine Brummspannung, die bis zu 3Vpp gross ist. Sie ist auf das hochfrequente Funksignal der Fernbedienung zurückzuführen. Durch diese wird Strom induziert. Zusammen mit den hohen Eingangswiderständen der Messgeräte ergibt das eine hohe Spannung.

Die Vorgängergruppe hatte ebenfalls solche Messwerte erhalten. Es hatte jedoch keinen Einfluss auf den Betrieb des Autos. Bei uns jedoch lagen die Gleichspannungswerte zu weit auseinander, es musste eine andere Lösung gefunden werden.

Als erste Schutzvariante gegen die hochfrequenten Störungen verwendeten wir anstelle der bestehenden Verkabelung abgeschirmte Twisted Pair Kabel. Dieses führten wir direkt zum PC: Die Umschaltung PC/Manuell hatten wir weggelassen. Als auch das nicht funktionierte, wickelten wir eine Alufolie um das Flachbandkabel zwischen Fernbedienung und Laptop. Die Folie legten wir auf das Groundpotential der *DAQCard*. Weitere Lösungsansätze waren ein Lastwiderstand, so dass die induzierten Ströme relativ klein sind.

Schliesslich bestand die Lösung darin, die Schaltung mit den Potentionmetern möglichst genau nachzubilden. Der Innenwiderstand der *DAQCard* ist sehr klein. Die Schaltung aber hat normalerweise einen Widerstand von etwa 2.5kΩ vom

Mittenabgriff nach Ground. Somit probierten wir diese Lösung aus ($R=2.7\text{ k}\Omega$). Sie führte schliesslich zum Ziel.

Unsere Annahme, dass nur die entsprechenden Spannungen an den Eingang gelegt werden müssen, erwies sich daher als falsch. Es könnte sein, dass die Schaltung der Fernbedienung den Widerstand als Teil eines Schwingkreises benutzt. Da wir kein Schema haben, bleibt dies eine Spekulation. Solange es so läuft muss uns das nicht weiter kümmern.

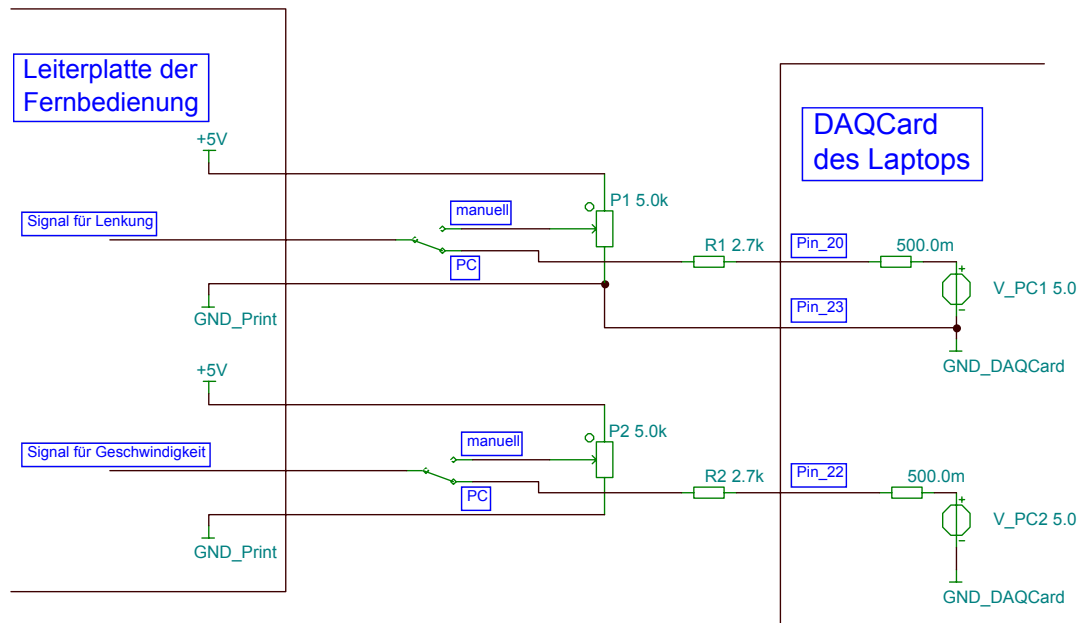


Abbildung 12 - Verbindung der Fernbedienung mit dem Laptop

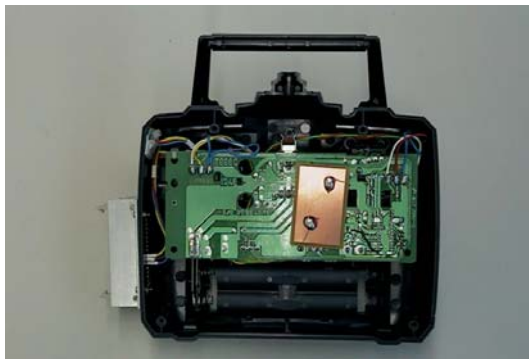


Abbildung 13 - Fernbedienung offen

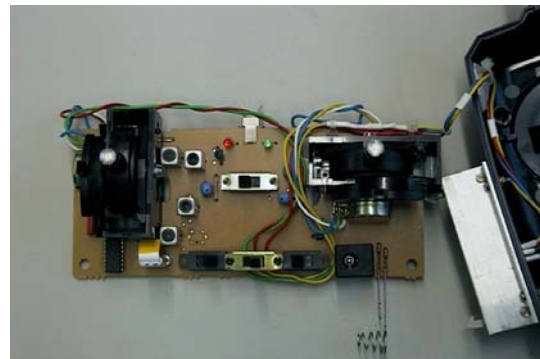


Abbildung 14 - Print Fernbedienung

Die in Tabelle 1 gemessenen Spannungen stimmen durch den Spannungsabfall am Widerstand nicht mehr mit dem Radeinschlag überein. Es ergibt sich die folgende überarbeitete Tabelle:

Spannung an DAQCard in V	Spannung nach dem R in V	Radeinschlag in Grad
3.0	2.8	10 rechts
2.6	2.6	0
2.2	2.4	10 links

Tabelle 2 – angepasste Spannungswerte beim Radeinschlag

8 Die GUIs

Um unsere Aufgabe etwas übersichtlicher zu machen, haben wir sie in Teilschritte gegliedert. Zu den einzelnen Teilen haben wir dann jeweils ein GUI¹ implementiert. In einem ersten Schritt ging es darum, die Verbindung vom Laptop zum Modellauto herzustellen. Um die entsprechenden Steuersignale für die Fernbedienung zu generieren, hatten wir eine D/A-Karte² zur Verfügung. Damit wir die D/A-Karte ansteuern konnten, haben wir das GUI *Untitled – FROG* implementiert. Die Verbindung vom Modellauto zum Laptop zurück geschieht mit einer Wireless-Cam. Die von der Kamera gesendeten Bilder haben wir in einem nächsten Schritt mit dem GUI *FrameGrabber* eingelesen und dann die Kanten detektiert. Schliesslich war auf den Kantenbildern noch die Position der Strasse zu bestimmen, um damit wiederum Steuersignale für die Fernbedienung zu errechnen. Dies haben wir mit dem GUI *Position* bewerkstelligt. Das Endprodukt unserer Arbeit, das GUI *FROG*, fasst die oben genannten Funktionalitäten zusammen.

Die folgenden Beschreibungen sind dazu gedacht, dem Leser einen schnellen Überblick über die verschiedenen GUIs zu geben. Für ausführlichere Erklärungen sei jeweils auf den Anhang und die Listings verwiesen.

8.1 Das GUI Untitled - FROG

8.1.1 Zweck

Dieses GUI ermöglicht es, das Modellauto mit der Maus zu steuern.

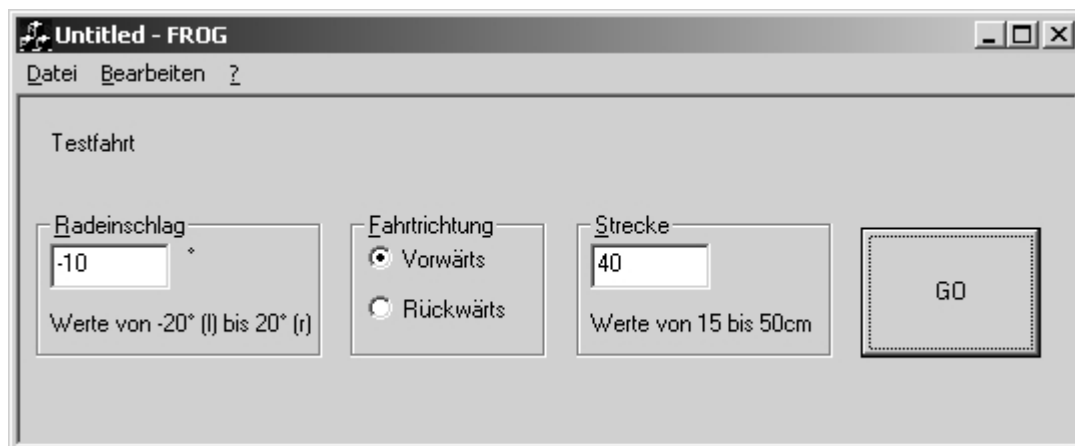


Abbildung 15 - Das GUI Untitled - FROG

8.1.2 Funktionalität

Unter *Radeinschlag* kann der gewünschte Einschlag in Grad angegeben werden. Die Beschränkung auf $\pm 20^\circ$ ist vom Fahrzeug gegeben.

¹ GUI = graphical user interface (graphische Benutzeroberfläche)

² D/A-Karte = wandelt ein digitales Signal in ein analoges Signal

Die Einstellung *Fahrtrichtung* ermöglicht es, sowohl vorwärts als auch rückwärts zu fahren.

Schliesslich kann noch die *Strecke* in cm vorgegeben werden, die das Modellauto zurücklegen soll. Allerdings ist dieser Wert ziemlich ungenau. Eine Limitation auf 50cm wäre nicht notwendig.

8.1.3 Umsetzung

Die D/A-Karte stellt eine Reihe von Funktionen zur Verfügung. Für unsere Arbeit war jedoch nur die Funktion `AO_VWrite` wichtig. Diese erlaubt es, eine Gleichspannung auszugeben. Für eine ausführliche Beschreibung der Ansteuerung der D/A-Karte in *Visual C++* sei auf den Anhang A1 (Seite 24) verwiesen.

8.2 Das GUI FrameGrabber

8.2.1 Zweck

Mit diesem GUI können Bilder von der *USB*-Schnittstelle eingelesen und deren Kanten detektiert werden.

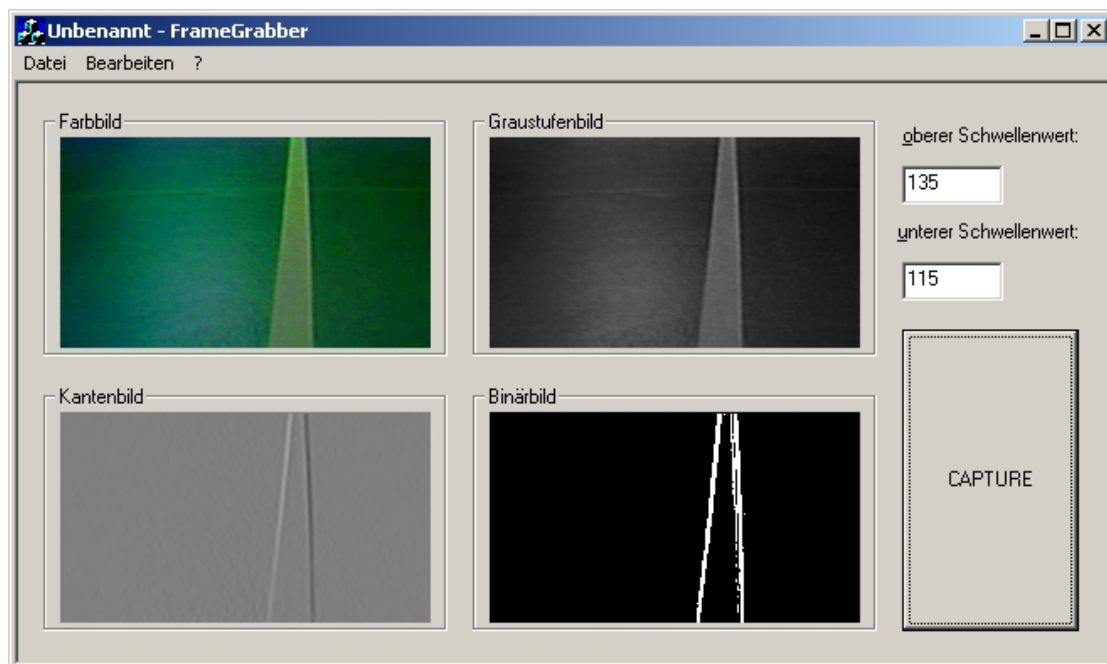


Abbildung 16 - Das GUI FrameGrabber

8.2.2 Funktionalität

Mit den Parametern *oberer Schwellenwert* und *unterer Schwellenwert* kann eingestellt werden, ab wann eine Kante im Kantenbild auch im Binärbild als Kante dargestellt wird. Das heisst je näher die Schwellenwerte bei der Zahl 127 liegen, desto feinere Farbunterschiede werden im Binärbild noch als Kanten dargestellt.

8.2.3 Umsetzung

Das eingelesene Farbbild wird zuerst in ein Graustufenbild umgewandelt. Dieses wird dann mit dem Sobel-Operator (siehe Kap. 5, Seite 8) gefaltet, wodurch das Kantenbild entsteht. Mit den oben erwähnten Schwellenwerten kann dann schliesslich noch das Binärbild erzeugt werden.

Für die genaue Implementation in *Visual C++* sei auf den Anhang A2 (Seite 25) und das Listing L2 (Seite 41) verwiesen.

8.3 Das GUI *Position*

8.3.1 Zweck

Dieses GUI bestimmt die Position der Strasse in einem gegebenen Binärbild.

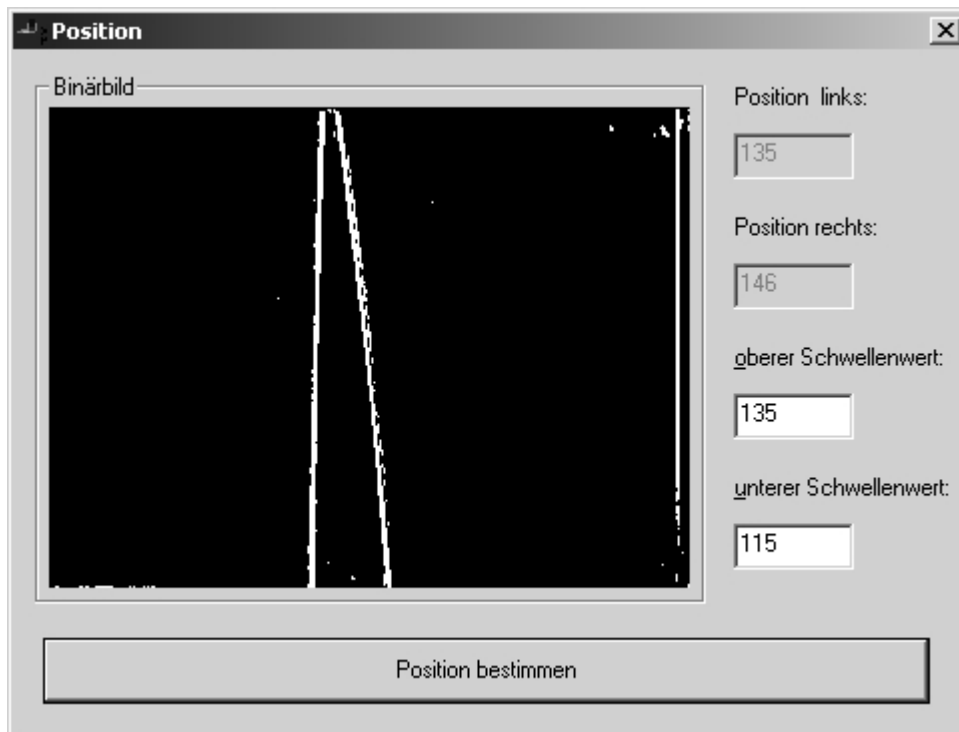


Abbildung 17 - Das GUI *Position*

8.3.2 Funktionalität

Den Parametern *oberer Schwellenwert* und *unterer Schwellenwert* kommt die gleiche Bedeutung zu wie beim GUI *FrameGrabber*. Die *Position links* bezieht sich auf den Abstand (in Pixel) zwischen linkem Bildrand und linker Kante. Die *Position rechts* wird ebenfalls vom linken Bildrand aus gemessen, jedoch bis zur rechten Kante.

8.3.3 Umsetzung

Unser Algorithmus sucht zuerst die Position der Strasse am unteren Bildrand. Hat er diese gefunden, so wird der Vorgang etwas weiter oben wiederholt, wobei jedoch nicht mehr die ganze Bildbreite abgesucht wird, sondern nur noch der relevante Teil. Dadurch kann das Risiko eine falsche Kante als Strasse zu detektieren verringert

werden. Ziel ist es, die Position der Strasse am oberen Bildrand zu bestimmen. Für einen vertieften Einblick in die Funktionsweise unseres Algorithmus sei auf den Anhang A3 (Seite 27) verwiesen.

Nebenbei: Der Strich am rechten Bildrand ist auf einen Defekt der Kamera zurückzuführen.

8.4 Das GUI *FROG*

8.4.1 Zweck

Mit diesem GUI kann man das Modellauto fahren lassen und die Kamera kalibrieren.

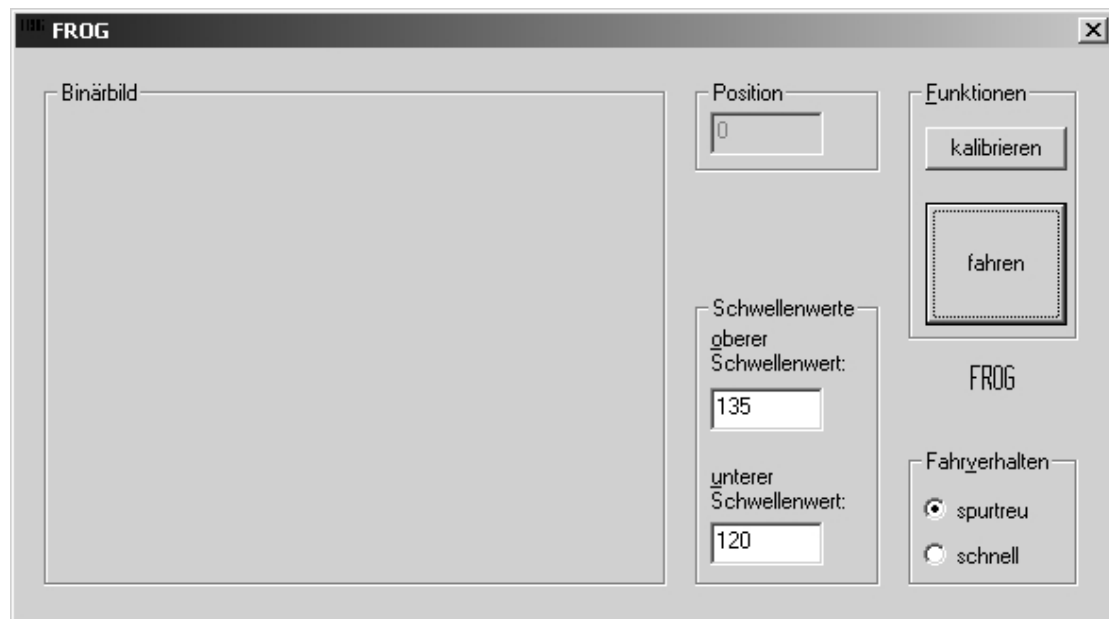


Abbildung 18 - Das GUI *FROG*

8.4.2 Funktionalität

Den Parametern *oberer Schwellenwert* und *unterer Schwellenwert* kommt die gleiche Bedeutung zu wie beim GUI *FrameGrabber*. Der Wert *Position* gibt den Abstand der Strassenmitte zum linken Bildrand an. Mit dem *Fahrverhalten* kann man festlegen, ob das Modellauto dem Strassenverlauf eher schnell oder eher genau folgen soll. Der Button *fahren* startet den Algorithmus, während der Button *kalibrieren* ein Dialogfenster zum Ausrichten der Kamera öffnet.

Für die korrekte Inbetriebnahme der Hardware und Software sei auf den Anhang A4 (Seite 32) verwiesen.

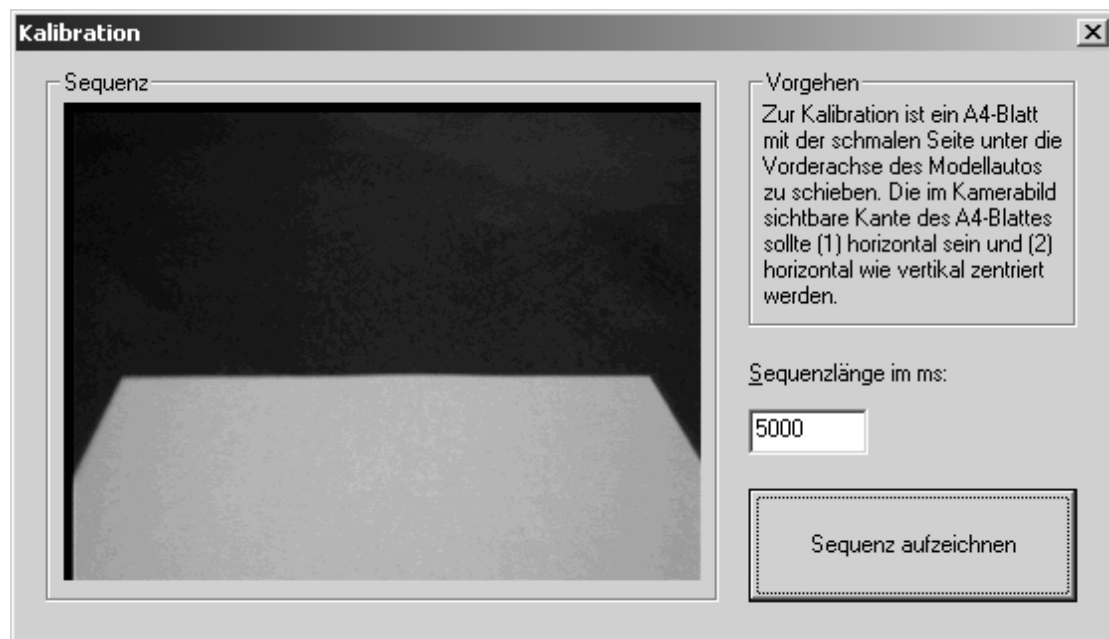


Abbildung 19 - Das Dialogfenster Kalibration

Bei *Sequenzlänge* kann eine Zeit eingestellt werden. Während dieser wird das Videobild angezeigt, um die Kamera auszurichten.

8.4.3 Umsetzung

Nebst den in den obigen GUIs genannten Ausführungen kommt hier noch hinzu, dass Abbruchmechanismen implementiert wurden. Des weiteren wurde eine Funktionalität hinzugefügt, um kurzzeitige Bildstörungen zu umgehen.

Der Radeinschlag wird anhand des Wertes *Position* bestimmt. Diese wird, wie bereits erwähnt, bei guten Lichtverhältnissen meist am oberen Bildrand gemessen. Da die Kamera einen begrenzten Blickwinkel hat, mussten wir den Radeinschlag auf 10° beschränken.

9 Zeitplan

9.1 Theoretische Vorarbeiten

Nachdem wir am 15.11.2001 mit unserer Arbeit gestartet sind, haben wir uns einige Grundlagen im Bereich der Bildverarbeitung, *Visual C++*-Programmierung und in der Dokumentation mit *Word* angeeignet. Weiter haben wir einen geeigneten Algorithmus ausgewählt.

9.2 Praktische Arbeiten

Der vorliegende Soll-Zeitplan ist aufgrund unserer Kenntnisse Ende Jahr erstellt worden.

Kalenderwoche	2	3	4	5	6	7	8	9
Bezug Laborarbeitsplatz	■							
Organisation der Hard- und Software	■	■	■					
Auto mittels DAC-Karte ansteuern		■	■	■	■			
Bild von Kamera in C++ einlesen			■	■	■	■		
Algorithmus und GUI implementieren				■	■	■	■	
Test und Optimierung						■	■	■
Dokumentation	■	■	■			■	■	■

Soll	■
Ist	■

Tabelle 3 - Zeitplan Soll und Ist

9.3 Kommentar zu den praktischen Arbeiten

9.3.1 Bezug Laborarbeitsplatz

Wir konnten unsere Arbeitsplätze wie geplant Ende Woche 2 beziehen.

9.3.2 Organisation der Hard- und Software

Die Verzögerungen in diesem Bereich sind darauf zurückzuführen, dass wir den Laptop und die *DAQCard* entgegen den Abmachungen erst anfangs der Woche 4 erhalten haben.

9.3.3 Auto mittels DAC-Karte ansteuern

Ende der Woche 4 waren wir in der Lage Spannungen am Laptop auszugeben. Die weiteren Verzögerungen sind aufgrund des in Messungen am Modell (Seite 13) beschriebenen Problemen entstanden.

9.3.4 Bild von Kamera in C++ einlesen

Da wir weder mit der *Windows*-Programmierung noch mit dem Einlesen von Bildern über die *USB*-Schnittstelle Erfahrung hatten, waren wir mit diesem Punkt überfordert. Auch konnten wir auf keinerlei Informationen von Vorgängern zurückgreifen. Die Kamera ist ausserdem als Anwendung und nicht als Entwicklungstool gedacht. Dementsprechend war in dieser Hinsicht keinerlei Dokumentation vorhanden. Dieses Problem gehört unserer Ansicht nach eher in den Bereich der Computertechnik oder Informatik.

9.3.5 Algorithms und GUI implementieren

Diesen Punkt konnten wir wesentlich schneller erledigen als geplant. Die Verzögerung hat sich aufgrund der Verspätung in den vorhergehenden Teilaufgaben ergeben.

9.3.6 Test und Optimierung

Ab Woche 8 funktionierte das Kamerasystem nicht mehr zuverlässig. Wir konnten daher nur eine Testfahrt absolvieren bevor das Kamerasystem komplett ausfiel. Ausreichende Tests und die geplante Optimierung konnten nicht mehr durchgeführt werden.

9.3.7 Dokumentation

Abgesehen vom zweiwöchigen Unterbruch bemühten wir uns die Dokumentation parallel zu den Arbeiten auf dem Laufenden zu halten.

10 Schlussfolgerungen

10.1 Was wir erreicht haben

Unsere Aufgabe bestand darin, dass ein Modellauto mit Hilfe einer Kamera selbständig einem Strassenverlauf folgt. Wir haben diese erfüllt. Entgegen anfänglicher Vermutungen haben wir sogar ein kontinuierliches Fahren erreicht.

10.2 Probleme

Da uns während den ersten Wochen noch kein Arbeitsplatz zur Verfügung stand, konnten diverse Probleme nicht gelöst bzw. analysiert werden. So fand sich zum Beispiel kein Rechner mit *USB*-Unterstützung, auf dem wir gleichzeitig die nötige Kamerasoftware hätten installieren können.

Aufgrund der Aufgabenstellung sind wir davon ausgegangen, dass die Signalübertragung vom Laptop zum Modellauto funktioniert. Wir rechneten nicht damit, dass die Verbindung des D/A-Wandlers mit der Fernbedienung Probleme bereiten würde (siehe Messungen am Modell, Seite 13).

Ein weiteres unerwartetes Problem stellte das Einlesen von Bildern über die *USB*-Schnittstelle dar. Wir haben hier überdurchschnittlich viel Zeit verloren, die uns danach anderweitig fehlte.

Da der Empfänger der Kamera ab Woche 8 nicht mehr zuverlässig funktionierte, konnten wir die Geschwindigkeit und den Radeinschlag nicht mehr optimieren.

10.3 Erkenntnisse

Als Vorteil hat sich das iterative Vorgehen erwiesen. Dadurch konnten wir uns rasch einen Überblick über die Problematik verschaffen, wodurch auch ungeahnte Probleme teilweise zum Vorschein traten.

Im Verlaufe der Arbeit haben wir folgendes festgestellt:

- Die wahren Probleme liegen nicht dort, wo man sie vermutet. So hätten wir zu Beginn nicht gedacht, dass wir bloss zwei Wochen mit dem Programmieren der Bildverarbeitung beschäftigt sein würden.
- Der Teufel steckt im Detail. Vermeintlich kleine Probleme haben sich als grosse entpuppt.

10.4 Vorschläge für zukünftige Arbeiten

- Stop-Button im GUI FROG implementieren. Für einen Stop-Button wären parallele Prozesse notwendig.
- Fahrverhalten optimieren: Die Geschwindigkeit soweit erhöhen, dass das Auto die Spur gerade noch nicht verliert. Den Radeinschlag nach einer Formel berechnen.
- Kantendetektion verbessern: Automatische Anpassung der Schwellenwerte an die Lichtverhältnisse.
- Erkennung von Kreuzungen, Verzweigungen und Verkehrssignalen; entsprechend reagieren.

- ❑ Wenn keine Strasse mehr detektiert werden kann, soll das Auto selbständig zum letzten definierten Punkt zurückfahren.
- ❑ Bei langen geraden Stücken soll die Geschwindigkeit automatisch erhöht werden.
- ❑ Um konstantere Lichtverhältnisse zu erreichen könnte man eine Lampe auf das Auto montieren.

10.5 Dank

Wir möchten es an dieser Stelle nicht unterlassen Herrn Schuster für die angenehme Zusammenarbeit zu danken. Um uns bei auftretenden Problemen zu helfen hat er auch mal selbst den LötKolben zur Hand genommen oder am Wochenende ein paar Zeilen Code geschrieben.

Auch hat es uns gefreut zu sehen, dass er ein persönliches Interesse an unserer Arbeit hatte.

11 Anhang

Nachfolgend sind einzelne Details der GUIs genauer beschrieben.

11.1 Anhang A1 – *Untitled FROG*

Für die Steuerung des Autos geben wir eine Spannung direkt auf die Fernbedienung. Diese kann über den in Unsere Hardware (Seite 6) beschriebenen Digital Analog Converter ausgegeben werden.

Damit die Karte aus Visual C++ angesteuert werden kann, müssen die entsprechenden Treiber installiert und die Bibliotheken geladen werden. Diese befinden sich auf der Doppel-CD *National Instruments, Data Acquisition Driver Software (NI-DAQ)*.

11.1.1 Ansteuerung der DAC-Karte in Visual C++

Die Karte stellt eine Reihe von Funktionen zur Verfügung. Für unsere Anwendung ist aber nur eine von Bedeutung:

`status = AO_VWrite(deviceNumber, chan, voltage)`

- `status` = Variable für einen Fehlercode
- `deviceNumber` = Nummer der Karte, in unserem Fall 1
- `chan` = Kanalnummer, bei uns 0 oder 1
- `voltage` = gewünschte Spannung.

Um in Visual C++ mit diesen Funktionen zu arbeiten, sind folgende Punkte zu beachten resp. Pfade anzugeben:

- Alle **.cpp*-Files müssen das File *nidaq.h* enthalten: `#include "nidaq.h"`
- Dem Projekt muss bekannt sein, wo die Headerfiles zu finden sind:
Project → Settings → C/C++ → Category: Preprocessor → Additional include directories: *.\National Instruments\NI-DAQ\Include*
- Weiter muss das Projekt die Library *nidaq32.lib* kennen:
Project → Settings → Link → Object/Library modules: *nidaq32.lib*

11.2 Anhang A2 - *FrameGrabber*

Wie schon in Wahl des Algorithmus zur Kantendetektion (Seite 10) beschrieben, benutzen wir für unsere Arbeit den Sobel-Operator in x-Richtung.

In diesem Kapitel möchten wir die Funktionsweise genau so beschreiben, wie wir sie bei unserem Problem angewendet haben. Die einzelnen Schritte können mit dem GUI *FrameGrabber* visualisiert werden.

Das Farbbild der Kamera erhalten wir in einem Array. Dabei sind jeweils der Rot, Grün und Blau-Wert eines Pixel nacheinander abgespeichert. Der Durchschnitt dieser drei Werte ergibt den Graustufenwert des betreffenden Pixels

Zur Vereinfachung der Erklärungen führen wir die Faltung wieder bei einem Schwarz-Weiss-Bild durch. Bei uns wird diese aber am Graustufenbild vollzogen.

Als Erstes wird das Bild in Abbildung 20 mit dem Sobel-Operator in x-Richtung gefaltet.

0	0	0	0	0	0	0	0	0	0
0	0	-1*0	0*0	1*0	0	0	0	0	0
0	0	-2*0	0*1	2*1	1	1	1	0	0
0	0	-1*0	0*1	1*1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Abbildung 20 - Graustufenbild der Kamera

Dadurch entsteht das Bild in Abbildung 21. Im Gegensatz zum vollständigen Sobel-Algorithmus erhalten wir hier positive wie auch negative Werte (Gradientbildung fällt weg). Damit erhält dieses Bild einen 3D-Charakter. Ausserdem können nur senkrechte Kanten detektiert werden.

Da die Randpixel durch die Faltung mit dem Sobel-Operator nicht bestimmt sind, weisen wir ihnen den neutralen Wert 0 (=keine Kante) zu. So können wir weiterhin mit einer gleichgrossen Matrix arbeiten.

0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	-1	-1	0
0	0	3	3	0	0	0	-3	-3	0
0	0	4	4	0	0	0	-4	-4	0
0	0	4	4	0	0	0	-4	-4	0
0	0	4	4	0	0	0	-4	-4	0
0	0	3	3	0	0	0	-3	-3	0
0	0	1	1	0	0	0	-1	-1	0
0	0	0	0	0	0	0	0	0	0

Abbildung 21 - Bild mit Sobel-Operator in x-Richtung gefaltet

Schlussendlich interessieren uns nur die Kanten eines Bildes. Wir entscheiden mit Hilfe zweier Schwellenwerte, was als Kante definiert wird und was nicht. In unserem Beispiel setzen wir den unteren Schwellenwert bei -3 und den oberen bei 3 fest. Das bedeutet, dass alle Werte $\leq (-3)$ und ≥ 3 als Kante definiert werden. In der Abbildung 22 sind die Kanten weiss dargestellt. Natürlich könnte auch nur ein Schwellenwert verwendet werden. Dementsprechend müssten die Beträge der Werte verglichen werden. Mit unserer Variante haben wir aber die Möglich die eine Kante stärker zu gewichten.

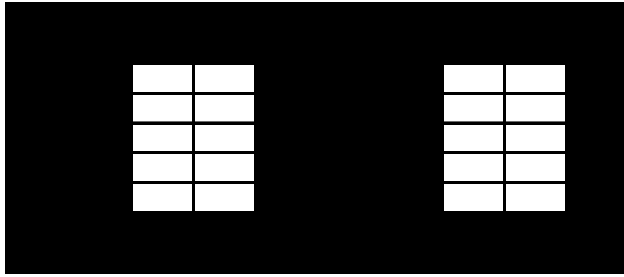


Abbildung 22 - Kantenbild mit den Schwellwerten (-3) resp. 3

11.3 Anhang A3 - Position

Dieser Anhang erläutert das genaue Vorgehen beim Bestimmen der Position der Strasse auf einem Binärbild.

Um den Algorithmus sicherer zu machen, haben wir fünf Kontrollmechanismen eingebaut, welche nachstehend beschrieben sind. In den Auszügen aus dem Programmcode ist die angesprochene Funktionalität jeweils fett gedruckt. Der ausführliche Code ist in den Listings zu finden.

11.3.1 Kontrollmechanismus I – *störende Pixel*

11.3.1.1 Beschreibung

Hier geht es im Wesentlichen darum, dass eine Kante aus mindestens zwei in x-Richtung aufeinanderfolgenden weissen Pixel bestehen muss, um als Strasse interpretiert zu werden. Damit werden einzelne Pixel, welche nicht zur Strasse gehören, ignoriert. Besonders eindrücklich zeigt dies die folgende Abbildung, auf welcher die Position der Strasse trotz der vielen Störpixel richtig detektiert wurde.

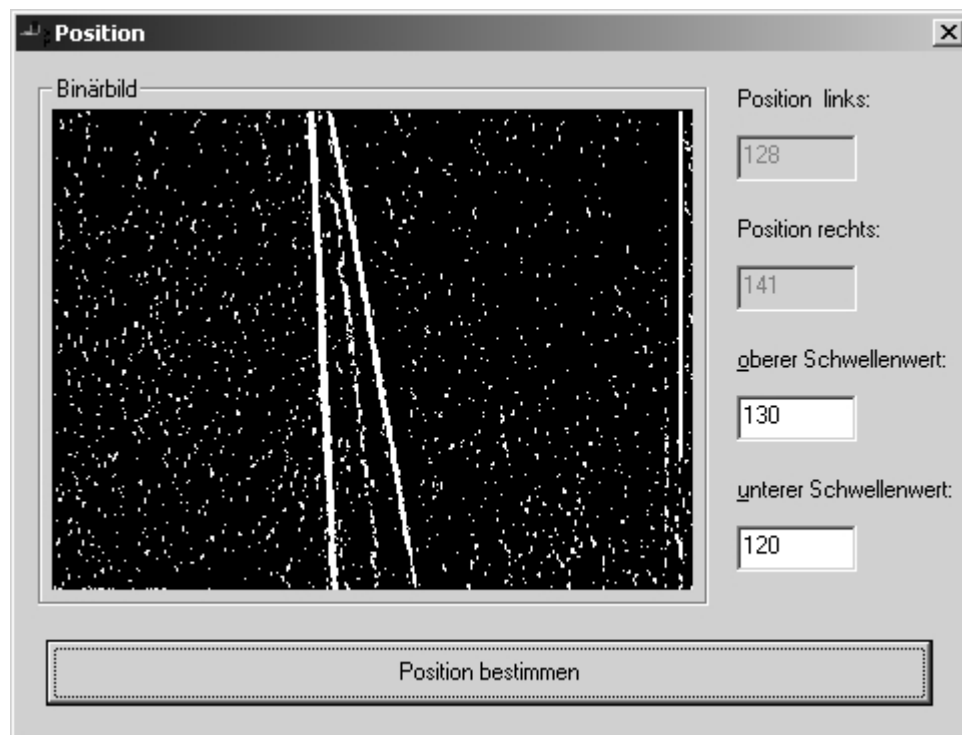


Abbildung 23 – *störende Pixel*

11.3.1.2 Auszug aus dem Programmcode

```
for(int i = Nr + x_Startwert; i < (Nr + width); i++)  
{  
    if(BinaerMap[i] == 255 && BinaerMap[i+1] == 255)  
    {  
        temp = i - y_Koordinate * width;  
        return temp;  
    }  
}
```

11.3.2 Kontrollmechanismus II – *Strassenbreite*

11.3.2.1 Beschreibung

Ein weiteres Vorgehen zum Feststellen ob die Strasse richtig detektiert wurde ist das Bestimmen der Strassenbreite. Diese muss sich in einem gewissen Wertebereich befinden, sonst wird das Resultat verworfen.

11.3.2.2 Auszug aus dem Programmcode

```
temp_links_neu = Position_links(Messort, x_Startwert_links);  
temp_rechts_neu = Position_rechts(Messort,  
x_Startwert_rechts);  
temp_diff = temp_rechts_neu - temp_links_neu;  
if((temp_diff < 60) && (temp_diff > 5)) {...}
```

11.3.3 Kontrollmechanismus III – *abgeschnittene Strasse*

11.3.3.1 Beschreibung

Sollte aus irgend einem Grund die Strasse nicht über die ganze Höhe des Bildes detektiert werden können – weil sie zum Beispiel verdeckt oder zu ende ist – so verwendet der Algorithmus den letzten noch gültigen Wert um die Position zu bestimmen.

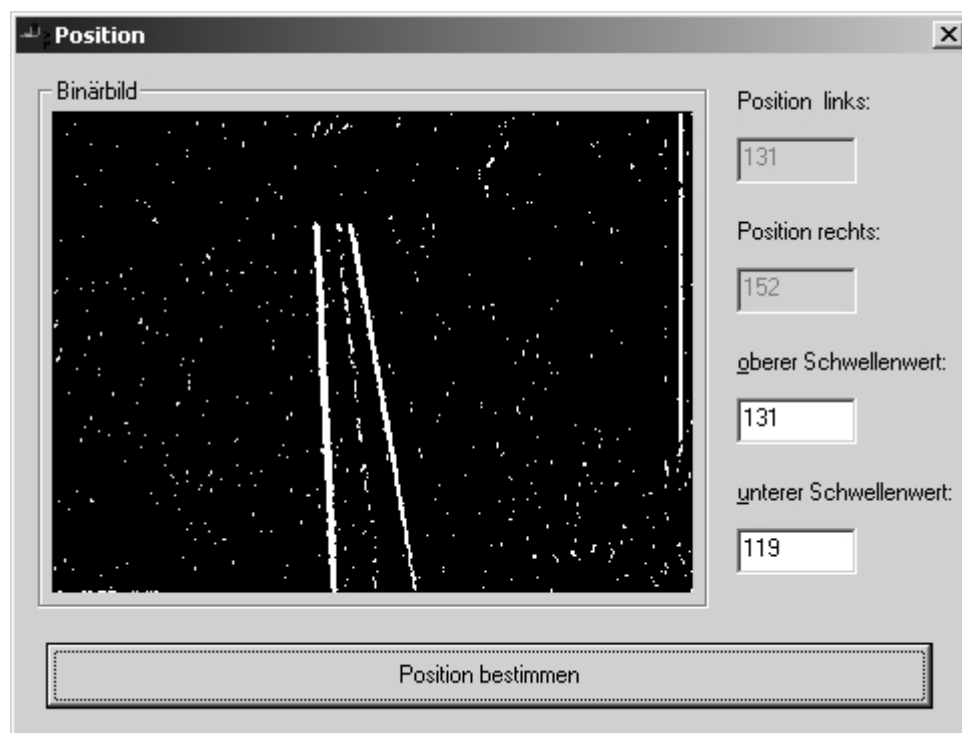


Abbildung 24 – abgeschnittene Strasse

11.3.3.2 Auszug aus dem Programmcode

```
if((temp_diff < 60) && (temp_diff > 5))
```

```
{  
    ...  
    temp_links = temp_links_neu;  
    temp_rechts = temp_rechts_neu;  
}  
else  
{  
    temp_links = temp_links_alt;  
    temp_rechts = temp_rechts_alt;  
    Abbruch = 1;  
}
```

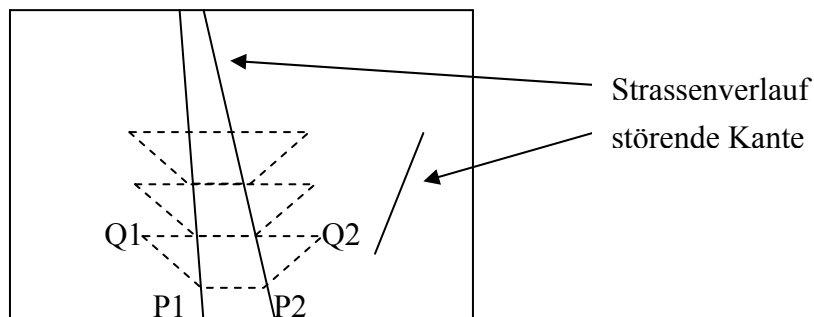
11.3.4 Kontrollmechanismus IV – *andere Kanten*

11.3.4.1 Beschreibung

Um die Kanten zu detektieren suchen wir das Bild vom linken respektive vom rechten Rand zur Mitte hin ab.

Wenn neben der Strasse noch andere ausgeprägte Kanten auf dem Binärbild zu sehen sind, sollen diese nicht als Strasse interpretiert werden. Dazu gehen wir wie folgt vor: Die Strasse wird als erstes am unteren Bildrand detektiert. Der Algorithmus findet die Punkte P1 und P2. Danach suchen wir die Strasse etwas weiter oben. Es ist dann jeweils nicht mehr notwendig, die gesamte Breite des Bildes nach der Strasse abzusuchen, weil der Strassenverlauf stetig ist. Es reicht daher, zwischen den Punkten Q1 und Q2 nach der Strasse zu suchen. Somit werden störende Kanten, welche weiter weg von der Strasse liegen, ignoriert.

In der Figur 1 ist das Verfahren schematisch dargestellt.



Figur 1 - Vorgehen zur sicheren Detektion der Strasse

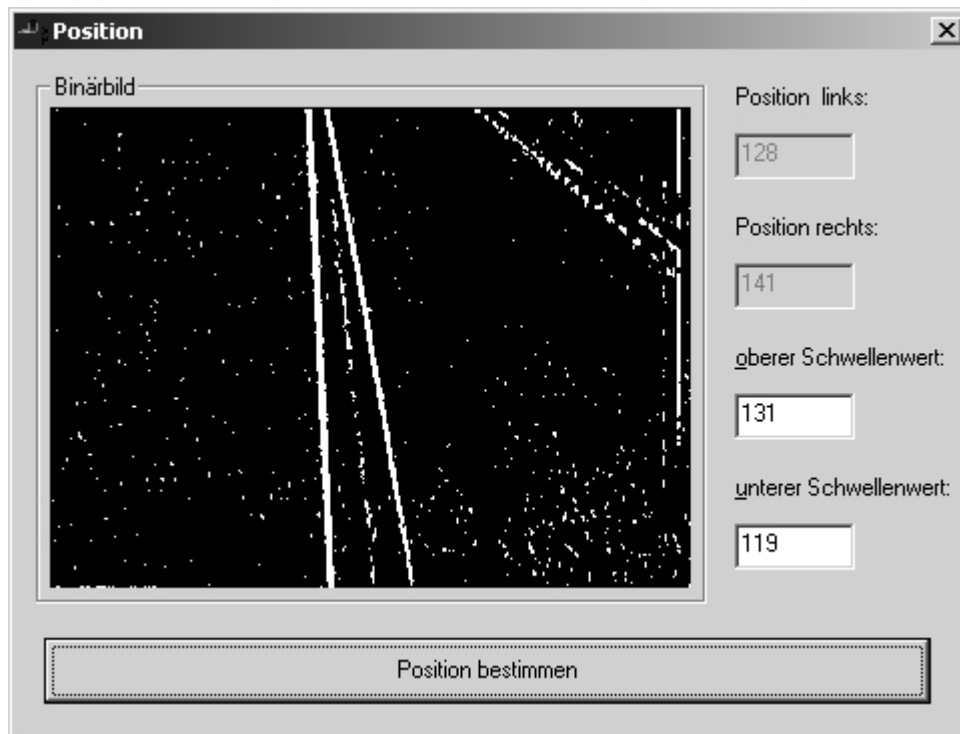


Abbildung 25 – andere Kanten

11.3.4.2 Auszug aus dem Programmcode

```

Messort -= 20;
if(Messort < 10) Abbruch = 1;
x_Startwert_links = temp_links_neu - 20;
if(x_Startwert_links < 0) Abbruch = 1;
x_Startwert_rechts = 320 - (temp_rechts_neu + 20);
if(x_Startwert_rechts < 10) Abbruch = 1;
  
```

11.3.5 Kontrollmechanismus V – Übertragungsstörung

11.3.5.1 Beschreibung

Damit Übertragungsstörungen der Wireless-Cam das Fahrverhalten nicht beeinflussen, wird bei nicht erfolgreicher Detektion der Strasse ein neues Bild von der Kamera eingelesen und ausgewertet. Dieser Vorgang wird insgesamt dreimal wiederholt. Wenn auch dann noch keine Strasse erkennbar ist wird eine Fehlermeldung ausgegeben und das Modellauto gestoppt.

11.3.5.2 Auszug aus dem Programmcode

```

do
{
    ...
    if((temp_diff < 60) && (temp_diff > 5))
    {
        ...
    }
}
  
```

```
    else
    {
        i++;
    }
}while(i < AnzVersuche);
if(i == AnzVersuche) {stop = 1;}
```

11.4 Anhang A4 - *FROG*

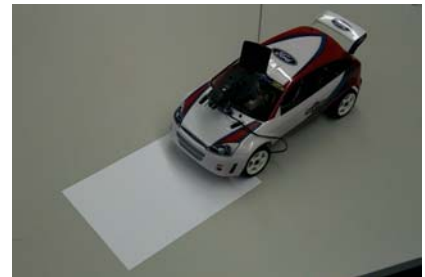
Dieser Anhang besteht aus einer Checkliste für die Inbetriebnahme von *FROG*.
Bei Störungen sei auf den Punkt 11.4.5 verwiesen.

11.4.1 Material Check

- Kamera mit 9V Anschlussclip und 9V-Block-Batterie
- Empfänger zu Kamera mit 12V Netzgerät, Grabber und Kabel
- DAQ-Card (installiert) mit Kabel (an einem Ende 34-poliger DIN-Stecker)
- Laptop mit Netzteil und Software
- Auto mit Akku
- Fernbedienung mit Batterien
- 2cm breites Malerклеbeband

11.4.2 Inbetriebnahme

- Strecke aufkleben, ACHTUNG: Kleinstmöglicher Radius = 2 bis 3m
- Für optimale Lichtverhältnisse sorgen
- Laptop einschalten
- frog* starten (gibt automatisch Standardspannungen an die Ausgänge)
- Fernbedienung einschalten
- Fernbedienung mit dem Laptop verbinden
- Auto einschalten
- Kamera-Empfänger einschalten
- 9V-Batterie an Kamera anhängen
- Kalibrieren*-Button klicken
- A4-Blatt gemäss Abbildung unter der Vorderachse des Modellautos platzieren
- Kamera so ausrichten, dass:
 - Die sichtbare Kante des Blattes horizontal erscheint.
 - Die sichtbare Kante horizontal wie auch vertikal in die Mitte des Bildes zu liegen kommt (siehe GUI).
- Start-Button klicken



11.4.3 Betrieb

- Auto stoppt, wenn die Linie zu Ende ist oder die Kamera keine solche mehr erkennen kann.

11.4.4 Stoppen und Ausschalten

- "Handstop": Speisung des Kamera-Empfängers ausschalten.
- Auto ausschalten
- Fernbedienung von Laptop trennen und ausschalten
- Laptop ausschalten

11.4.5 Störungsbehebung

Im folgenden Kapitel sind einige mögliche Probleme im Umgang mit FROG, sowie die Lösung dieser beschrieben. Überprüfen sie allgemein zuerst die Stromversorgung, sei es mit einem Netzteil oder die Batterien. Die Fernbedienung hat eine Spannungsüberwachung eingebaut. Bei genügend grosser Spannung leuchten die grüne und die rote LED.

11.4.5.1 Die Kamera zeigt kein Bild an

Kanaleinstellung bei Kamera und Empfänger überprüfen:

Kamera: Zuerst Gummi-Abdeckung wegnehmen. Schalter ganz bei der Linse entspricht Kanal A beim Empfänger Kalibration vergessen. Siehe auch Abbildung 26 und Abbildung 27.



Abbildung 26 - Kanal bei Empfänger



Abbildung 27 - Kanal bei Kamera

11.4.5.2 Das Binärbild ist verrauscht

Es sieht aus wie in Abbildung 28



Abbildung 28 - Verrauschtes Binärbild

Stellen sie die Farbtiefe im Menu Anzeige → Einstellungen auf "True Color (24bit)" einstellen.

11.4.5.3 Spur verloren

Folgende Fehlermeldung erscheint:

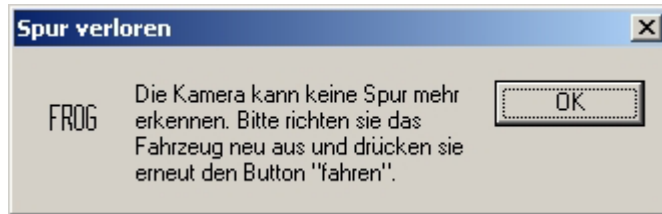


Abbildung 29 - Spur verloren

Entweder ist die Linie zu Ende oder die Lichtverhältnisse sind zu schlecht. Der Algorithmus muss wenigstens am unteren Bildrand beide Kanten des Klebebandes erkennen können.

Mögliche Lösungsmethoden:

- Schwellwerte ändern wie im Kapitel 8.2 (Seite 16) beschrieben
- Verdunkeln des Raumes
- Licht einschalten
- wechseln auf einen matten Boden

Als weiterer Fehler könnte das Klebeband zu schmal oder zu breit sein. Der Algorithmus ist so programmiert, dass er mit einer Klebebandbreite von 2cm funktioniert.

11.4.5.4 Das Auto fährt nicht

Fernbedienung muss auf PC anstatt manuell eingestellt sein (PC = Schalter rechts). Der Umschalter befindet sich im Batteriefach. Siehe Abbildung 30 und Abbildung 31.



Abbildung 30 – Fernbedienung

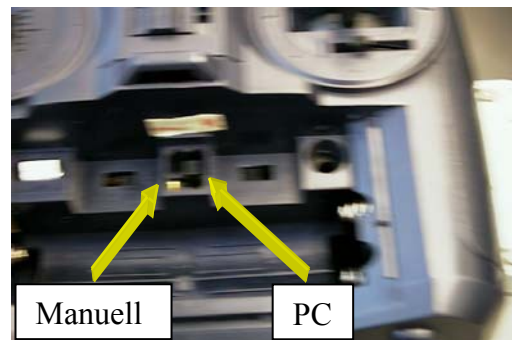


Abbildung 31 - Umschalter PC/Manuell

11.4.5.5 Kamera stürzt dauernd ab

Auf der Homepage von x10 nachschauen, ob ein neuerer Treiber verfügbar ist:

<http://www.x10.com/xrv/>

12 Listings

Nachfolgend ist ein Teil des Programmcodes aufgeführt. Namentlich die Files, die für das jeweilige GUI relevant sind. Alle anderen Files sind in elektronischer Form vorhanden.

Der selbstgeschriebene Code ist **fettgedruckt**.

12.1 Listing L1 – *Untitled FROG*

12.1.1 Headerfile *FROGView.h*

```
// FROGView.h : interface of the CFROGView class
//
////////////////////////////////////

#if !defined(AFX_FROGVIEW_H_B74875A9_BDB4_4B3B_A29C_EA3FE53FF1EA__INCLUDED_)
#define AFX_FROGVIEW_H_B74875A9_BDB4_4B3B_A29C_EA3FE53FF1EA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFROGView : public CFormView
{
protected: // create from serialization only
    CFROGView();
    DECLARE_DYNCREATE(CFROGView)

public:
    //{{AFX_DATA(CFROGView)
    enum { IDD = IDD_FROG_FORM };
    int         m_status;
    int         m_Radeinschlag;
    int         m_device;
    //}}AFX_DATA
};
```

```
int          m_channel0;
int          m_channel1;
float        m_Strecke;
int          m_Vorwaerts;
//}}AFX_DATA

// Attributes
public:
    CFROGDoc* GetDocument();

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CFROGView)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void OnInitialUpdate(); // called first time after construct
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CFROGView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CFROGView)
    afx_msg void OnGo();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
#ifndef _DEBUG // debug version in FROGView.cpp
inline CFROGDoc* CFROGView::GetDocument()
    { return (CFROGDoc*)m_pDocument; }
#endif

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FROGVIEW_H__B74875A9_BDB4_4B3B_A29C_EA3FE53FF1EA__INCLUDED_)
```

12.1.2 Implementationsfile *FROGView.cpp*

```
// FROGView.cpp : implementation of the CFROGView class
//

#include "stdafx.h"
#include "FROG.h"

#include "FROGDoc.h"
#include "FROGView.h"
#include "nidaq.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CFROGView

IMPLEMENT_DYNCREATE(CFROGView, CFormView)

BEGIN_MESSAGE_MAP(CFROGView, CFormView)
   //{{AFX_MSG_MAP(CFROGView)
    ON_BN_CLICKED(IDC_Go, OnGo)
   //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()

////////////////////////////////////
// CFROGView construction/destruction

CFROGView::CFROGView()
    : CFormView(CFROGView::IDD)
{
    //{{AFX_DATA_INIT(CFROGView)
    m_Radeinschlag = 0;
    m_Strecke = 15.0f;
    m_Vorwaerts = 0;
    //}}AFX_DATA_INIT

    m_status = AO_VWrite(1, 1, 2.64);           // Spannung initialisieren
    m_status = AO_VWrite(1, 0, 2.62);         // Spannung initialisieren
}

CFROGView::~~CFROGView()
{
}

void CFROGView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFROGView)
    DDX_Text(pDX, IDC_Radeinschlag, m_Radeinschlag);
    DDV_MinMaxInt(pDX, m_Radeinschlag, -20, 20);
    DDX_Text(pDX, IDC_Strecke, m_Strecke);
    DDV_MinMaxFloat(pDX, m_Strecke, 15.f, 50.f);
    DDX_Radio(pDX, IDC_Vorwaerts, m_Vorwaerts);
    //}}AFX_DATA_MAP
}

BOOL CFROGView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
}
```

```
        return CFormView::PreCreateWindow(cs);
    }

void CFROGView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit();
}

////////////////////////////////////
// CFROGView diagnostics

#ifdef _DEBUG
void CFROGView::AssertValid() const
{
    CFormView::AssertValid();
}

void CFROGView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CFROGDoc* CFROGView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFROGDoc)));
    return (CFROGDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
// CFROGView message handlers

void CFROGView::OnGo()
{
    m_device      = 1;
    m_channel0    = 0;
    m_channel1    = 1;
}
```

```
// lokale Variablen
int m_status;
int m_TimeToSleep;
float m_Geschwindigkeit = (float)0.05;
float m_voltageBewegung;
double m_voltageEinschlag;

// Funktionalität
UpdateData(TRUE);

// Einschlag: ausgegeben auf DAC1out
m_voltageEinschlag = 0.02 * m_Radeinschlag + 2.6;
m_status = AO_VWrite(m_device, m_channel1, m_voltageEinschlag);

// Bewegung: ausgegeben auf DAC0out
if(!m_Vorwaerts)
    m_voltageBewegung = (float)2.54;
else
    m_voltageBewegung = (float)2.69;

m_TimeToSleep = m_Strecke / m_Geschwindigkeit;
m_status = AO_VWrite(m_device, m_channel0, m_voltageBewegung);
Sleep(m_TimeToSleep);
m_voltageBewegung = (float)2.62;
m_status = AO_VWrite(m_device, m_channel0, m_voltageBewegung);
}

// Fahrzeit in ms
// Geschwindigkeit in cm/ms
// Spannung am AO_0
// Spannung am AO_1

// 2.54V zum Vorwärtsfahren
// 2.69V zum Rückwärtsfahren

// t= s/v
// fahren

// 2.62V zum Stillstehen
// halten
```

12.2 Listing L2 – *FrameGrabber*

12.2.1 Headerfile *FrameGrabberView.h*

```
// FrameGrabberView.h : interface of the CFrameGrabberView class
//
///////////////////////////////////////////////////////////////////

#if !defined(AFX_FRAMEGRABBERVIEW_H_A059FA99_6B43_4F65_AD74_FBB894E837E5__INCLUDED_)
#define AFX_FRAMEGRABBERVIEW_H_A059FA99_6B43_4F65_AD74_FBB894E837E5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFrameGrabberView : public CFormView
{
protected: // create from serialization only
    CFrameGrabberView();
    DECLARE_DYNCREATE(CFrameGrabberView)

public:
    //{{AFX_DATA(CFrameGrabberView)
    enum { IDD = IDD_FRAMEGRABBER_FORM };
    int             m_oberer_Schwellenwert;
    int             m_unterer_Schwellenwert;
    //}}AFX_DATA

// Attributes
public:
    CFrameGrabberDoc* GetDocument();
    HWND             hWndC;
    Long             width;
    Long             height;
    unsigned char*   RGBMap;
    unsigned char*   GrayMap;
```

```
    unsigned char*    TempMap;
    unsigned char*    KanteMap;
    unsigned char*    BinaerMap;

// Operations
public:
    void Faltung(int, int);
    unsigned char pixel_lesen(int, int);
    void pixel_schreiben(int, int, unsigned char);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFrameGrabberView)
    public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual void OnInitialUpdate(); // called first time after construct
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CFrameGrabberView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CFrameGrabberView)
    afx_msg void OnCapture();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in FrameGrabberView.cpp
```

```
inline CFrameGrabberDoc* CFrameGrabberView::GetDocument()  
{ return (CFrameGrabberDoc*)m_pDocument; }  
#endif  
  
////////////////////////////////////  
  
//{{AFX_INSERT_LOCATION}}  
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.  
  
#endif // !defined(AFX_FRAMEGRABBERVIEW_H__A059FA99_6B43_4F65_AD74_FBB894E837E5__INCLUDED_)
```

12.2.2 Implementationsfile *FrameGrabberView.cpp*

```
// FrameGrabberView.cpp : implementation of the CFrameGrabberView class  
//  
  
#include "stdafx.h"  
#include "FrameGrabber.h"  
#include "FrameGrabberDoc.h"  
#include "FrameGrabberView.h"  
  
#include "vfw.h"  
  
#ifdef _DEBUG  
#define new DEBUG_NEW  
#undef THIS_FILE  
static char THIS_FILE[] = __FILE__;  
#endif  
  
////////////////////////////////////  
// CFrameGrabberView  
  
IMPLEMENT_DYNCREATE(CFrameGrabberView, CFormView)  
  
BEGIN_MESSAGE_MAP(CFrameGrabberView, CFormView)  
    //{{AFX_MSG_MAP(CFrameGrabberView)  
        ON_BN_CLICKED(IDC_CAPTURE, OnCapture)  
    //}}AFX_MSG_MAP  
END_MESSAGE_MAP()
```

```
////////////////////////////////////  
// CFrameGrabberView construction/destruction  
  
CFrameGrabberView::CFrameGrabberView()  
    : CFormView(CFrameGrabberView::IDD)  
{  
    //{{AFX_DATA_INIT(CFrameGrabberView)  
    m_oberer_Schwellenwert = 135;  
    m_unterer_Schwellenwert = 115;  
    //}}AFX_DATA_INIT  
    // TODO: add construction code here  
  
}  
  
CFrameGrabberView::~CFrameGrabberView()  
{  
}  
  
void CFrameGrabberView::DoDataExchange(CDataExchange* pDX)  
{  
    CFormView::DoDataExchange(pDX);  
    //{{AFX_DATA_MAP(CFrameGrabberView)  
    DDX_Text(pDX, IDC_oberer_Schwellenwert, m_oberer_Schwellenwert);  
    DDV_MinMaxInt(pDX, m_oberer_Schwellenwert, 128, 255);  
    DDX_Text(pDX, IDC_unterer_Schwellenwert, m_unterer_Schwellenwert);  
    DDV_MinMaxInt(pDX, m_unterer_Schwellenwert, 0, 126);  
    //}}AFX_DATA_MAP  
}  
  
BOOL CFrameGrabberView::PreCreateWindow(CREATESTRUCT& cs)  
{  
    // TODO: Modify the Window class or styles here by modifying  
    // the CREATESTRUCT cs  
  
    return CFormView::PreCreateWindow(cs);  
}  
  
void CFrameGrabberView::OnInitialUpdate()  
{  
    CFormView::OnInitialUpdate();  
}
```

```
GetParentFrame()->RecalcLayout();
ResizeParentToFit();
// *** erzeugen eines Capture Window ***
hWndC = capCreateCaptureWindow("Capture Window", WS_CHILD | WS_VISIBLE, 20, 20, 0, 0, m_hWnd, 0);
capDriverConnect(hWndC, 0);

}

////////////////////////////////////
// CFrameGrabberView diagnostics

#ifdef _DEBUG
void CFrameGrabberView::AssertValid() const
{
    CFormView::AssertValid();
}

void CFrameGrabberView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CFrameGrabberDoc* CFrameGrabberView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFrameGrabberDoc)));
    return (CFrameGrabberDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
// CFrameGrabberView message handlers

void CFrameGrabberView::OnCapture ()
{
    UpdateData(TRUE);

    ::OpenClipboard(hWndC); // öffnet und reserviert das Clipboard
    EmptyClipboard(); // leert das Clipboard und gibt Benutzungsrecht
    capGrabFrame(hWndC); // holt ein Frame und zeigt es an
    capEditCopy(hWndC); // kopiert Inhalt des Buffers ins Clipboard
}
```

```

::CloseClipboard(); // schliesst das Clipboard und ermöglicht somit
// anderen Funktionen den Zugriff auf die Daten

::OpenClipboard(m_hWnd); // öffnet und reserviert das Clipboard
if(IsClipboardFormatAvailable(CF_BITMAP)) // prüft ob im Clipboard ein
{ // CF_BITMAP-Format vorhanden ist
    HBITMAP handle = (HBITMAP)GetClipboardData(CF_BITMAP); // holt Daten von einem bestimmten Format aus dem Clipboard
    CBitmap * p_bitmap = CBitmap::FromHandle(handle); // erzeugt einen (echten) Zeiger auf das Objekt

    CClientDC cdc(this); // erzeugt Objekt der Klasse CClientDC
    CDC dc; // erzeugt Objekt der Klasse CDC
    dc.CreateCompatibleDC(&cdc);
    dc.SelectObject(p_bitmap); // wählt ein Objekt in den entsprechenden device context
    BITMAP BMI; p_bitmap->GetBitmap(&BMI); // gibt Informationen über p_bitmap an BMI zurück
    width = BMI.bmWidth;
    height = BMI.bmHeight;

    // *** This works only if the desktop is set to 24 bit / pixel color ***
    RGBMap= (unsigned char*)malloc(width*height*3); // malloc: memory allocation, 24bpp
    GrayMap= (unsigned char*)malloc(width*height); // 8 bit per pixel
    TempMap= (unsigned char*)malloc(width*height*3); // 24 bit per pixel
    KanteMap= (unsigned char*)malloc(width*height); // 8 bit per pixel
    BinaerMap=(unsigned char*)malloc(width*height); // 8 bit per pixel

    p_bitmap->GetBitmapBits(width*height*3,RGBMap); // jedes bit aus p_bitmap wie nach RGBMap übertragen

    // *** KanteMap initialisieren ***
    for(int i = 0; i<width*height; i++)
    {
        KanteMap[i] = 127;
    }

    // *** Farbbild wird in Graustufenbild umgewandelt ***
    for(i = 0; i<width*height; i++)
    {
        GrayMap[i]=(RGBMap[3*i]+RGBMap[3*i+1]+RGBMap[3*i+2])/3; // Grauwert = (Rotwert+Grünwert+Blauwert)/3
    }

    // *** Graustufenbild wird mit Sobel-Operator gefaltet ***
    for(int y = 1; y < height; y++)

```

```
{
    for(int x = 1; x < width; x++)
    {
        Faltung(x, y);
    }
}

// *** Kantenbild wird im Binärbild umgewandelt ***
for(i = 0; i<width*height; i++)
{
    if((KanteMap[i] >= m_oberer_Schwellenwert) || (KanteMap[i] <= m_unterer_Schwellenwert))
        BinaerMap[i] = 255;
    else
        BinaerMap[i] = 0;
}

// *** zeichnet das Farbbild ***
cdc.BitBlt(25,32,width-100,height-115,&dc,0,0,SRCCOPY); // kopiert bitmap von dc zu cdc
// -> angezeigt im Capturewindow

// *** zeichnet das Graustufenbild ***
for(i = 0; i<width*height; i++) // kopiert Inhalt von GrayMap nach TempMap
{
    TempMap[3*i] =GrayMap[i];
    TempMap[3*i+1]=GrayMap[i];
    TempMap[3*i+2]=GrayMap[i];
}

p_bitmap->SetBitmapBits(width*height*3,TempMap); // kopiert Inhalt von TempMap nach p_bitmap
cdc.BitBlt(280,32,width-100,height-115,&dc,0,0,SRCCOPY); // kopiert bitmap von dc zu cdc
// -> angezeigt im Capturewindow

// *** zeichnet das Kantenbild ***
for(i = 0; i<width*height; i++) // kopiert Inhalt von KanteMap nach TempMap
{
    TempMap[3*i] =KanteMap[i];
    TempMap[3*i+1]=KanteMap[i];
    TempMap[3*i+2]=KanteMap[i];
}

p_bitmap->SetBitmapBits(width*height*3,TempMap); // kopiert Inhalt von TempMap nach p_bitmap
```

```
cdc.BitBlt(25,195,width-100,height-115,&dc,0,0,SRCCOPY); // kopiert bitmap von dc zu cdc
// -> angezeigt im Capturewindow

// *** zeichnet das Binärbild ***
for(i = 0; i<width*height; i++) // kopiert Inhalt von KanteMap nach TempMap
{
    TempMap[3*i] =BinaerMap[i];
    TempMap[3*i+1]=BinaerMap[i];
    TempMap[3*i+2]=BinaerMap[i];
}

p_bitmap->SetBitmapBits(width*height*3,TempMap); // kopiert Inhalt von TempMap nach p_bitmap
cdc.BitBlt(280,195,width-100,height-115,&dc,0,0,SRCCOPY); // kopiert bitmap von dc zu cdc
// -> angezeigt im Capturewindow

free(RGBMap); // allozierten speicher freigeben
free(GrayMap); // allozierten speicher freigeben
free(TempMap); // allozierten speicher freigeben
free(KanteMap); // allozierten speicher freigeben

}
::CloseClipboard(); // schliesst das Clipboard und ermöglicht somit
// anderen Funktionen den Zugriff auf die Daten

}

// *** Implementation der Funktion Faltung ***

void CFrameGrabberView::Faltung(int x, int y)
{
    signed char temp_signed = 0;
    unsigned char temp_unsigned = 0;

    temp_signed += (-1) * pixel_lesen(x-1,y-1)/8; // Division mit 8 um eine
    temp_signed += (-2) * pixel_lesen(x-1,y)/8; // Bereichsüberschreitung
    temp_signed += (-1) * pixel_lesen(x-1,y+1)/8; // zu verhindern.
// unsigned char: 0 bis 255
// signed char: -128 bis 127

    temp_signed += 1 * pixel_lesen(x+1,y-1)/8;
    temp_signed += 2 * pixel_lesen(x+1,y)/8;
    temp_signed += 1 * pixel_lesen(x+1,y+1)/8;

    temp_unsigned = temp_signed + 127; // signed char-Werte in unsigned char-Werte umwandeln
}
```

```
        pixel_schreiben(x, y, temp_unsigned);
    }

// *** Implementation der Funktion pixel_lesen ***

unsigned char CFrameGrabberView::pixel_lesen(int x, int y)
{
    int Nr;
    unsigned char Wert;

    Nr = y * width + x;
    Wert = GrayMap[Nr];

    return Wert;
}

// *** Implementation der Funktion pixel_schreiben ***

void CFrameGrabberView::pixel_schreiben(int x, int y, unsigned char Wert)
{
    int Nr;

    Nr = y * width + x;
    KanteMap[Nr] = Wert;
}

// Eingabe: Position in einer fiktiven
// zweidimensionalen Matrix
// Ausgabe: Wert an der entsprechenden Position eines
// Arrays

// Wert einer fiktiven zweidimensionalen
// Matrix an die entsprechende Position
// eines Arrays schreiben.
```

12.3 Listing L3 – Position

12.3.1 Headerfile *PositionDlg.h*

```
// PositionDlg.h : header file
//

#if !defined(AFX_POSITIONDLG_H__49B9A812_8BA8_43FD_B655_6469377C1854__INCLUDED_)
#define AFX_POSITIONDLG_H__49B9A812_8BA8_43FD_B655_6469377C1854__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

////////////////////////////////////
// CPositionDlg dialog

class CPositionDlg : public CDialog
{
// Construction
public:
    CPositionDlg(CWnd* pParent = NULL);    // standard constructor

    HWND hWndC;
    int Messort;
    long width;
    long height;
    unsigned char* RGBMap;
    unsigned char* GrayMap;
    unsigned char* TempMap;
    unsigned char* KanteMap;
    unsigned char* BinaerMap;

// Dialog Data
    //{{AFX_DATA(CPositionDlg)
    enum { IDD = IDD_POSITION_DIALOG };

```

```
int          m_Pos_links;
int          m_Pos_rechts;
int          m_oberer_Schwellenwert;
int          m_unterer_Schwellenwert;
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CPositionDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    void Faltung(int, int);
    unsigned char pixel_lesen(int, int);
    void pixel_schreiben(int, int, unsigned char);
    int Position_links(int, int);
    int Position_rechts(int, int);
    void erzeuge_Binaerbild();

    // Generated message map functions
    //{{AFX_MSG(CPositionDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnPosition_bestimmen();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_POSITIONDLG_H__49B9A812_8BA8_43FD_B655_6469377C1854__INCLUDED_)
```

12.3.2 Implementationsfile *PositionDlg.cpp*

```
// PositionDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Position.h"
#include "PositionDlg.h"

#include "vfw.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
//}}AFX_MSG
}
```

```
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
   //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
        // NO message handlers
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPositionDlg dialog

CPositionDlg::CPositionDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CPositionDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CPositionDlg)
    m_Pos_links = 0;
    m_Pos_rechts = 0;
    m_oberer_Schwellenwert = 135;
    m_unterer_Schwellenwert = 115;
   //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPositionDlg::DoDataExchange(CDataExchange* pDX)
{
```

```
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CPositionDlg)
DDX_Text(pDX, IDC_Pos_links, m_Pos_links);
DDX_Text(pDX, IDC_Pos_rechts, m_Pos_rechts);
DDX_Text(pDX, IDC_oberer_Schwellenwert, m_oberer_Schwellenwert);
DDV_MinMaxInt(pDX, m_oberer_Schwellenwert, 128, 255);
DDX_Text(pDX, IDC_unterer_Schwellenwert, m_unterer_Schwellenwert);
DDV_MinMaxInt(pDX, m_unterer_Schwellenwert, 0, 126);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPositionDlg, CDialog)
//{{AFX_MSG_MAP(CPositionDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDC_Position_bestimmen, OnPosition_bestimmen)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPositionDlg message handlers

BOOL CPositionDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
```

```
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
}

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);        // Set small icon

// *** erzeugen eines Capture Window ***
hWndC = capCreateCaptureWindow("Capture Window", WS_CHILD | WS_VISIBLE, 20, 20, 0, 0, m_hWnd, 0);
capDriverConnect(hWndC, 0);

return TRUE; // return TRUE unless you set the focus to a control
}

void CPositionDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CPositionDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting
    }
}
```

```
SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

// Center icon in client rectangle
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;

// Draw the icon
dc.DrawIcon(x, y, m_hIcon);
}
else
{
    CDialog::OnPaint();
}
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CPositionDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

// *** Implementation der Funktion OnPosition_bestimmen ***
// (bestimmt die Position für ein Bild im Format 320*240)

void CPositionDlg::OnPosition_bestimmen()
{
    int i = 0; // Laufvariable
    int Abbruch = 0;
    int x_Startwert_links;
    int x_Startwert_rechts;
    int temp_links;
    int temp_links_alt;
    int temp_links_neu;
    int temp_rechts;
```

```
int temp_rechts_alt;
int temp_rechts_neu;
int temp_diff;

do
{
    erzeuge_Binaerbild();
    Messort = 230;
    x_Startwert_links = 10;
    x_Startwert_rechts = 10;
    temp_links_neu = Position_links(Messort, x_Startwert_links);
    temp_rechts_neu = Position_rechts(Messort, x_Startwert_rechts);
    temp_diff = temp_rechts_neu - temp_links_neu;
    if((temp_diff < 60) && (temp_diff > 5))
    {
        Messort -= 20;
        if(Messort < 10) Abbruch = 1;
        x_Startwert_links = temp_links_neu - 20;
        if(x_Startwert_links < 0) Abbruch = 1;
        x_Startwert_rechts = 320 - (temp_rechts_neu + 20);
        if(x_Startwert_rechts < 10) Abbruch = 1; // wegen Bildstörung unserer Kamera
        temp_links = temp_links_neu;
        temp_rechts = temp_rechts_neu;

        while(Abbruch == 0)
        {
            temp_links_neu = Position_links(Messort, x_Startwert_links);
            temp_rechts_neu = Position_rechts(Messort, x_Startwert_rechts);
            temp_diff = temp_rechts_neu - temp_links_neu;
            if((temp_diff < 60) && (temp_diff > 5))
            {
                Messort -= 20;
                if(Messort < 10) Abbruch = 1;
                x_Startwert_links = temp_links_neu - 20;
                if(x_Startwert_links < 0) Abbruch = 1;
                x_Startwert_rechts = 320 - (temp_rechts_neu + 20);
                if(x_Startwert_rechts < 10) Abbruch = 1; // wegen Bildstörung unserer Kamera
                temp_links = temp_links_neu;
                temp_rechts = temp_rechts_neu;
            }
        }
    }
}
```

```
        else
        {
            temp_links = temp_links_alt;
            temp_rechts = temp_rechts_alt;
            Abbruch = 1;
        }
        temp_links_alt = temp_links_neu;
        temp_rechts_alt = temp_rechts_neu;
    }

    m_Pos_links = temp_links;
    m_Pos_rechts = temp_rechts;
    i = 3;
}
else
{
    i++;
}
}while(i < 3);
UpdateData(FALSE);
}

// *** Implementation der Funktion erzeuge_Binaerbild ***

void CPositionDlg::erzeuge_Binaerbild()
{
    UpdateData(TRUE);

    ::OpenClipboard(hWndC);
    EmptyClipboard();
    capGrabFrame(hWndC);
    capEditCopy(hWndC);
    ::CloseClipboard();

    ::OpenClipboard(m_hWnd);
    if(IsClipboardFormatAvailable(CF_BITMAP))
    {
        HBITMAP handle = (HBITMAP)GetClipboardData(CF_BITMAP); // holt Daten von einem bestimmten Format aus dem Clipboard
    }
    // öffnet und reserviert das Clipboard
    // leert das Clipboard und gibt Benutzungsrecht
    // holt ein Frame und zeigt es an
    // kopiert Inhalt des Buffers ins Clipboard
    // schliesst das Clipboard und ermöglicht somit
    // anderen Funktionen den Zugriff auf die Daten

    // öffnet und reserviert das Clipboard
    // prüft ob im Clipboard ein
    // CF_BITMAP-Format vorhanden ist
}
```

```
CBitmap * p_bitmap = CBitmap::FromHandle(handle);           // erzeugt einen (echten) Zeiger auf das Objekt

CClientDC cdc(this);                                       // erzeugt Objekt der Klasse CClientDC
CDC dc;                                                    // erzeugt Objekt der Klasse CDC
dc.CreateCompatibleDC(&cdc);
dc.SelectObject(p_bitmap);                                 // wählt ein Objekt in den entsprechenden device context
BITMAP BMI; p_bitmap->GetBitmap(&BMI);                     // gibt Informationen über p_bitmap an BMI zurück
width = BMI.bmWidth;
height = BMI.bmHeight;

// *** This works only if the desktop is set to 24 bit / pixel color ***
RGBMap= (unsigned char*)malloc(width*height*3);           // malloc: memory allocation, 24bpp
GrayMap= (unsigned char*)malloc(width*height);           // 8 bit per pixel
TempMap= (unsigned char*)malloc(width*height*3);         // 24 bit per pixel
KanteMap= (unsigned char*)malloc(width*height);           // 8 bit per pixel
BinaerMap=(unsigned char*)malloc(width*height);           // 8 bit per pixel

p_bitmap->GetBitmapBits(width*height*3,RGBMap);           // jedes bit aus p_bitmap wie nach RGBMap übertragen

// *** KanteMap initialisieren ***
for(int i = 0; i<width*height; i++)
{
    KanteMap[i] = 127;
}

// *** Farbbild wird in Graustufenbild umgewandelt ***
for(i = 0; i<width*height; i++)
{
    GrayMap[i]=(RGBMap[3*i]+RGBMap[3*i+1]+RGBMap[3*i+2])/3; // Grauwert = (Rotwert+Grünwert+Blauwert)/3
}

// *** Graustufenbild wird mit Sobel-Operator gefaltet ***
for(int y = 1; y < height; y++)
{
    for(int x = 1; x < width; x++)
    {
        Faltung(x, y);
    }
}
```

```
// *** Kantenbild wird im Binärbild umgewandelt ***
for(i = 0; i<width*height; i++)
{
    if((KanteMap[i] >= m_oberer_Schwellenwert) || (KanteMap[i] <= m_unterer_Schwellenwert))
        BinaerMap[i] = 255; // weiss
    else
        BinaerMap[i] = 0; // schwarz
}

// *** zeichnet das Binärbild ***
for(i = 0; i<width*height; i++) // kopiert Inhalt von KanteMap nach TempMap
{
    TempMap[3*i] =BinaerMap[i];
    TempMap[3*i+1]=BinaerMap[i];
    TempMap[3*i+2]=BinaerMap[i];
}

p_bitmap->SetBitmapBits (width*height*3,TempMap); // kopiert Inhalt von TempMap nach p_bitmap
cdc.BitBlt(18,28,width,height,&dc,0,0,SRCCOPY); // kopiert bitmap von dc zu cdc
// -> angezeigt im Capturewindow

free (RGBMap); // allozierten speicher freigeben
free (GrayMap); // allozierten speicher freigeben
free (TempMap); // allozierten speicher freigeben
free (KanteMap); // allozierten speicher freigeben

}
::CloseClipboard(); // schliesst das Clipboard und ermöglicht somit
// anderen Funktionen den Zugriff auf die Daten
}

// *** Implementation der Funktion Faltung ***

void CPositionDlg::Faltung(int x, int y)
{
    signed char temp_signed = 0;
    unsigned char temp_unsigned = 0;

    temp_signed += (-1) * pixel_lesen(x-1,y-1)/8;
    temp_signed += (-2) * pixel_lesen(x-1,y)/8;
```

```
temp_signed += (-1) * pixel_lesen(x-1,y+1)/8;

temp_signed += 1 * pixel_lesen(x+1,y-1)/8;
temp_signed += 2 * pixel_lesen(x+1,y)/8;
temp_signed += 1 * pixel_lesen(x+1,y+1)/8;

temp_unsigned = temp_signed + 127;
pixel_schreiben(x, y, temp_unsigned);
}

// *** Implementation der Funktion pixel_lesen ***

unsigned char CPositionDlg::pixel_lesen(int x, int y)
{
    int Nr;
    unsigned char Wert;

    Nr = y * width + x;
    Wert = GrayMap[Nr];

    return Wert;
}

// *** Implementation der Funktion pixel_schreiben ***

void CPositionDlg::pixel_schreiben(int x, int y, unsigned char Wert)
{
    int Nr;

    Nr = y * width + x;
    KanteMap[Nr] = Wert;
}

// *** Implementation der Funktion Position_links ***

int CPositionDlg::Position_links(int y_Koordinate, int x_Startwert)
{
    int Nr, temp;

    Nr = y_Koordinate * width;
```

```
for(int i = Nr + x_Startwert; i < (Nr + width); i++)
{
    if(BinaerMap[i] == 255 && BinaerMap[i+1] == 255)           // zur Sicherheit sollen hier
    {                                                         // ZWEI Pixel weiss sein
        temp = i - y_Koordinate * width;                     // (Wert = 255)
        return temp;
    }
}

// *** Implementation der Funktion Position_rechts ***

int CPositionDlg::Position_rechts(int y_Koordinate, int x_Startwert)
{
    int Nr, temp;

    Nr = (y_Koordinate + 1) * width - 1;
    for(int i = Nr - x_Startwert; i > (Nr - width); i--)
    {
        if(BinaerMap[i] == 255 && BinaerMap[i-1] == 255)       // zur Sicherheit sollen hier
        {                                                         // ZWEI Pixel weiss sein
            temp = i - y_Koordinate * width;                     // (Wert = 255)
            return temp;
        }
    }
}
```

12.4 Listing L4 – FROG

12.4.1 Headerfile *FROGDlg.h*

```
// FROGDlg.h : header file
//

#if !defined(AFX_FROGDLG_H_FC1E2D31_0F7E_4996_AAD4_EC8A397E7BDC__INCLUDED_)
#define AFX_FROGDLG_H_FC1E2D31_0F7E_4996_AAD4_EC8A397E7BDC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

////////////////////////////////////
// CFROGDlg dialog

class CFROGDlg : public CDialog
{
// Construction
public:
    CFROGDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(CFROGDlg)
    enum { IDD = IDD_FROG_DIALOG };
    int         m_oberer_Schwellenwert;
    int         m_Position;
    int         m_unterer_Schwellenwert;
    int         m_spurtreu;
    //}}AFX_DATA

// ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CFROGDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
};
```

```
// Implementation
protected:
    HICON m_hIcon;

    HWND hWndC;
    int stop;
    int m_status;
    long width;
    long height;
    unsigned char* RGBMap;
    unsigned char* GrayMap;
    unsigned char* TempMap;
    unsigned char* KanteMap;
    unsigned char* BinaerMap;

    void Position_bestimmen();
    void Steuerspannungen_generieren();
    void erzeuge_Binaerbild();
    void Faltung(int, int);
    unsigned char pixel_lesen(int, int);
    void pixel_schreiben(int, int, unsigned char);
    int Position_links(int, int);
    int Position_rechts(int, int);

    // Generated message map functions
   //{{AFX_MSG(CFROGDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void Onkalibrieren();
    afx_msg void OnStart();
    afx_msg void Onfahren();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_FROGDLG_H_FC1E2D31_0F7E_4996_AAD4_EC8A397E7BDC__INCLUDED_)
```

12.4.2 Implementationsfile *FROGDlg.cpp*

```
// FROGDlg.cpp : implementation file
//

#include "stdafx.h"
#include "FROG.h"
#include "FROGDlg.h"
#include "KalibrationDlg.h"
#include "SpurverlorenDlg.h"
#include "vfw.h"
#include "nidaq.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL
```

```
// Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
   //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CFROGDlg dialog

CFROGDlg::CFROGDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CFROGDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CFROGDlg)
    m_oberer_Schwellenwert = 135;
    m_Position = 0;
    m_unterer_Schwellenwert = 120;
    m_spurtreu = 0;
   //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32

```

```
m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

m_status = AO_VWrite(1, 1, 2.64); // Spannung initialisieren
m_status = AO_VWrite(1, 0, 2.62); // Spannung initialisieren
}

void CFROGDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFROGDlg)
    DDX_Text(pDX, IDC_oberer_Schwellenwert, m_oberer_Schwellenwert);
    DDV_MinMaxInt(pDX, m_oberer_Schwellenwert, 128, 255);
    DDX_Text(pDX, IDC_Position, m_Position);
    DDX_Text(pDX, IDC_unterer_Schwellenwert, m_unterer_Schwellenwert);
    DDV_MinMaxInt(pDX, m_unterer_Schwellenwert, 0, 126);
    DDX_Radio(pDX, IDC_spurtreu, m_spurtreu);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CFROGDlg, CDialog)
    //{{AFX_MSG_MAP(CFROGDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_kalibrieren, Onkalibrieren)
    ON_BN_CLICKED(IDC_fahren, Onfahren)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CFROGDlg message handlers

BOOL CFROGDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
}

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE); // Set big icon
SetIcon(m_hIcon, FALSE); // Set small icon

return TRUE; // return TRUE unless you set the focus to a control
}

void CFROGDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CFROGDlg::OnPaint()
{
    if (IsIconic())
```

```
{
    CPaintDC dc(this); // device context for painting

    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
}
else
{
    CDialog::OnPaint();
}
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CFROGDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

////////////////////////////////////
// CFROGDlg real stuff

void CFROGDlg::Onkalibrieren()
{
    CKalibrationDlg myKalibrationDlg;
    myKalibrationDlg.DoModal();
}

void CFROGDlg::Onfahren()
{

```

```
int m_device = 1; // für DAC - Karte
int m_channel1 = 1; // für DAC - Karte
int m_channel0 = 0; // für DAC - Karte
double m_voltageEinschlag_neutral = 2.64; // Spannung für Geradeausfahrt
double m_voltageGeschwindigkeit_stop = 2.62; // Spannung für Stillstand
double m_voltageGeschwindigkeit_fahren; // Spannung für Vorwärtsfahren

UpdateData(TRUE);
stop = 0;

if(m_spurtreu)
    m_voltageGeschwindigkeit_fahren = 2.44; // schnell fahren
else
    m_voltageGeschwindigkeit_fahren = 2.54; // langsam fahren

// *** erzeugen eines Capture Window ***
hWndC = capCreateCaptureWindow("Capture Window", WS_CHILD | WS_VISIBLE, 20, 20, 0, 0, m_hWnd, 0);
capDriverConnect(hWndC, 0);

do
{
    Position_bestimmen();
    if(!stop)
    {
        Steuerspannungen_generieren();
        m_status = AO_VWrite(m_device, m_channel0, m_voltageGeschwindigkeit_fahren);
    }
    else
    {
        m_status = AO_VWrite(m_device, m_channel1, m_voltageEinschlag_neutral);
        m_status = AO_VWrite(m_device, m_channel0, m_voltageGeschwindigkeit_stop);

        CSpurverlorenDlg myCSpurverlorenDlg; // Dialogbox mit
        myCSpurverlorenDlg.DoModal(); // Warnung öffnen
    }
}while(!stop);

capDriverDisconnect(hWndC);
}
```

```
// *** Implementation der Funktion Position_bestimmen ***

void CFROGDlg::Position_bestimmen()
{
    int i = 0; // Laufvariable
    int AnzVersuche = 3; // bestimmt die Anzahl Versuche
    int Messort;
    int Abbruch = 0;
    int x_Startwert_links;
    int x_Startwert_rechts;
    int temp_links;
    int temp_links_alt;
    int temp_links_neu;
    int temp_rechts;
    int temp_rechts_alt;
    int temp_rechts_neu;
    int temp_diff;

    do
    {
        erzeuge_Binaerbild();
        Messort = 230;
        x_Startwert_links = 10;
        x_Startwert_rechts = 10;
        temp_links_neu = Position_links(Messort, x_Startwert_links);
        temp_rechts_neu = Position_rechts(Messort, x_Startwert_rechts);
        temp_diff = temp_rechts_neu - temp_links_neu;
        if((temp_diff < 60) && (temp_diff > 5))
        {
            Messort -= 20;
            if(Messort < 10) Abbruch = 1;
            x_Startwert_links = temp_links_neu - 20;
            if(x_Startwert_links < 0) Abbruch = 1;
            x_Startwert_rechts = 320 - (temp_rechts_neu + 20);
            if(x_Startwert_rechts < 10) Abbruch = 1; // wegen Bildstörung unserer Kamera
            temp_links = temp_links_neu;
            temp_rechts = temp_rechts_neu;

            while(Abbruch == 0)
            {
```

```
temp_links_neu = Position_links(Messort, x_Startwert_links);
temp_rechts_neu = Position_rechts(Messort, x_Startwert_rechts);
temp_diff = temp_rechts_neu - temp_links_neu;
if((temp_diff < 60) && (temp_diff > 5))
{
    Messort -= 20;
    if(Messort < 10) Abbruch = 1;
    x_Startwert_links = temp_links_neu - 20;
    if(x_Startwert_links < 0) Abbruch = 1;
    x_Startwert_rechts = 320 - (temp_rechts_neu + 20);
    if(x_Startwert_rechts < 10) Abbruch = 1; // wegen Bildstörung unserer Kamera
    temp_links = temp_links_neu;
    temp_rechts = temp_rechts_neu;
}
else
{
    temp_links = temp_links_alt;
    temp_rechts = temp_rechts_alt;
    Abbruch = 1;
}
temp_links_alt = temp_links_neu;
temp_rechts_alt = temp_rechts_neu;
}

m_Position = (temp_rechts + temp_links)/2;
i = 999;
}
else
{
    i++;
}
}while(i < AnzVersuche);
if(i == AnzVersuche) {stop = 1;} // kompletter Stop
UpdateData(FALSE);
}

// *** Implementation der Funktion erzeuge_Binaerbild ***

void CFROGDlg::erzeuge_Binaerbild()
```

```

{
    UpdateData (TRUE);

    ::OpenClipboard (hWndC); // öffnet und reserviert das Clipboard
    EmptyClipboard (); // leert das Clipboard und gibt Benutzungsrecht
    capGrabFrame (hWndC); // holt ein Frame und zeigt es an
    capEditCopy (hWndC); // kopiert Inhalt des Buffers ins Clipboard
    ::CloseClipboard (); // schliesst das Clipboard und ermöglicht somit
                        // anderen Funktionen den Zugriff auf die Daten

    ::OpenClipboard (m_hWnd); // öffnet und reserviert das Clipboard
    if (IsClipboardFormatAvailable (CF_BITMAP)) // prüft ob im Clipboard ein
    { // CF_BITMAP-Format vorhanden ist
        HBITMAP handle = (HBITMAP)GetClipboardData (CF_BITMAP); // holt Daten von einem bestimmten Format aus dem Clipboard
        CBitmap * p_bitmap = CBitmap::FromHandle (handle); // erzeugt einen (echten) Zeiger auf das Objekt

        CClientDC cdc (this); // erzeugt Objekt der Klasse CClientDC
        CDC dc; // erzeugt Objekt der Klasse CDC
        dc.CreateCompatibleDC (&cdc);
        dc.SelectObject (p_bitmap); // wählt ein Objekt in den entsprechenden device context
        BITMAP BMI; p_bitmap->GetBitmap (&BMI); // gibt Informationen über p_bitmap an BMI zurück
        width = BMI.bmWidth;
        height = BMI.bmHeight;

        // *** This works only if the desktop is set to 24 bit / pixel color ***
        RGBMap= (unsigned char*)malloc (width*height*3); // malloc: memory allocation, 24bpp
        GrayMap= (unsigned char*)malloc (width*height); // 8 bit per pixel
        TempMap= (unsigned char*)malloc (width*height*3); // 24 bit per pixel
        KanteMap= (unsigned char*)malloc (width*height); // 8 bit per pixel
        BinaerMap= (unsigned char*)malloc (width*height); // 8 bit per pixel

        p_bitmap->GetBitmapBits (width*height*3, RGBMap); // jedes bit aus p_bitmap wie nach RGBMap übertragen

        // *** KanteMap initialisieren ***
        for (int i = 0; i < width*height; i++)
        {
            KanteMap [i] = 127;
        }

        // *** Farbbild wird in Graustufenbild umgewandelt ***
    }
}
    
```

```

for(i = 0; i<width*height; i++)
{
    GrayMap[i]=(RGBMap[3*i]+RGBMap[3*i+1]+RGBMap[3*i+2])/3; // Grauwert = (Rotwert+Grünwert+Blauwert)/3
}

// *** Graustufenbild wird mit Sobel-Operator gefaltet ***
for(int y = 1; y < height; y++)
{
    for(int x = 1; x < width; x++)
    {
        Faltung(x, y);
    }
}

// *** Kantenbild wird im Binärbild umgewandelt ***
for(i = 0; i<width*height; i++)
{
    if((KanteMap[i] >= m_oberer_Schwellenwert) || (KanteMap[i] <= m_unterer_Schwellenwert))
        BinaerMap[i] = 255; // weiss
    else
        BinaerMap[i] = 0; // schwarz
}

// *** zeichnet das Binärbild ***
for(i = 0; i<width*height; i++) // kopiert Inhalt von KanteMap nach TempMap
{
    TempMap[3*i] =BinaerMap[i];
    TempMap[3*i+1]=BinaerMap[i];
    TempMap[3*i+2]=BinaerMap[i];
}

p_bitmap->SetBitmapBits(width*height*3,TempMap); // kopiert Inhalt von TempMap nach p_bitmap
cdc.BitBlt(20,34,width,height,&dc,0,0,SRCCOPY); // kopiert bitmap von dc zu cdc
// -> angezeigt im Capturewindow

free (RGBMap); // allozierten speicher freigeben
free (GrayMap); // allozierten speicher freigeben
free (TempMap); // allozierten speicher freigeben
free (KanteMap); // allozierten speicher freigeben

```

```
    }  
    ::CloseClipboard();  
}  
  
// *** Implementation der Funktion Faltung ***  
  
void CFROGDlg::Faltung(int x, int y)  
{  
    signed char    temp_signed    = 0;  
    unsigned char  temp_unsigned  = 0;  
  
    temp_signed += (-1) * pixel_lesen(x-1,y-1)/8;  
    temp_signed += (-2) * pixel_lesen(x-1,y)/8;  
    temp_signed += (-1) * pixel_lesen(x-1,y+1)/8;  
  
    temp_signed += 1 * pixel_lesen(x+1,y-1)/8;  
    temp_signed += 2 * pixel_lesen(x+1,y)/8;  
    temp_signed += 1 * pixel_lesen(x+1,y+1)/8;  
  
    temp_unsigned = temp_signed + 127;  
    pixel_schreiben(x, y, temp_unsigned);  
}  
  
// *** Implementation der Funktion pixel_lesen ***  
  
unsigned char CFROGDlg::pixel_lesen(int x, int y)  
{  
    int Nr;  
    unsigned char Wert;  
  
    Nr = y * width + x;  
    Wert = GrayMap[Nr];  
  
    return Wert;  
}  
  
// *** Implementation der Funktion pixel_schreiben ***  
  
void CFROGDlg::pixel_schreiben(int x, int y, unsigned char Wert)
```

```
{
    int Nr;

    Nr = y * width + x;
    KanteMap[Nr] = Wert;
}

// *** Implementation der Funktion Position_links ***

int CFROGDlg::Position_links(int y_Koordinate, int x_Startwert)
{
    int Nr, temp;

    Nr = y_Koordinate * width;
    for(int i = Nr + x_Startwert; i < (Nr + width); i++)
    {
        if(BinaerMap[i] == 255 && BinaerMap[i+1] == 255)           // zur Sicherheit sollen hier
        {                                                           // ZWEI Pixel weiss sein
            temp = i - y_Koordinate * width;                       // (Wert = 255)
            return temp;
        }
    }
}

// *** Implementation der Funktion Position_rechts ***

int CFROGDlg::Position_rechts(int y_Koordinate, int x_Startwert)
{
    int Nr, temp;

    Nr = (y_Koordinate + 1) * width - 1;
    for(int i = Nr - x_Startwert; i > (Nr - width); i--)
    {
        if(BinaerMap[i] == 255 && BinaerMap[i-1] == 255)           // zur Sicherheit sollen hier
        {                                                           // ZWEI Pixel weiss sein
            temp = i - y_Koordinate * width;                       // (Wert = 255)
            return temp;
        }
    }
}
```

```
// *** Implementation der Funktion Steuerspannungen_generieren() ***  
  
void CFROGDlg::Steuerspannungen_generieren()  
{  
    int m_device = 1; // für DAC - Karte  
    int m_channel1 = 1; // für DAC - Karte  
    int m_channel0 = 0; // für DAC - Karte  
    double m_voltageEinschlag_gerade = 2.6; // Spannung (in V) für Geradeausfahrt  
    double m_voltageEinschlag_links = 2.4; // Spannung (in V) für 10°-Linkskurve  
    double m_voltageEinschlag_rechts = 2.8; // Spannung (in V) für 10°-Rechtskurve  
  
    if(m_Position > 206)  
    {  
        m_status = AO_VWrite(m_device, m_channel1, m_voltageEinschlag_rechts); // 10°-Rechtskurve  
    }  
    else  
    {  
        if(m_Position > 103)  
        {  
            m_status = AO_VWrite(m_device, m_channel1, m_voltageEinschlag_gerade); // Geradeausfahrt  
        }  
        else  
        {  
            m_status = AO_VWrite(m_device, m_channel1, m_voltageEinschlag_links); // 10°-Linkskurve  
        }  
    }  
}
```