

Inhaltsverzeichnis

1	Intro	3
2	Aufgabenstellung	3
3	Erste Entwürfe in Matlab.....	4
3.1	Kantendetektion.....	4
3.2	Autokorrelation	6
3.3	Folgerungen	7
4	Das Matlab-File test.m.....	9
4.1	Abtastung des Bildes	9
4.2	Kantendetektion.....	10
4.2.1	Gewöhnlicher Hochpass.....	10
4.2.2	Gauss'scher Hochpass.....	10
4.3	Streckenzug durch Spline ersetzen	11
4.3.1	Hermite Splines	11
4.3.2	Normalisieren der Bilder	12
4.3.3	Darstellung von Gleichungen in Matrizen	13
4.3.4	Regression	13
4.3.5	Legen einer Kurve in einem Bild.....	15
5	Rund um die Hardware	20
5.1	Einleitung.....	20
5.2	Aufgetauchte Störungen.....	20
5.3	Wechsel auf FM.....	22
5.4	Umbau der Steuerung	23
5.5	Lehrer-/Schülerbetrieb.....	23
5.6	Pulstelegramm der Servo	25
5.7	uP-Interface	27
5.7.1	Aufbau und Funktion der Schaltung	28
5.7.2	Erklärungen zur seriellen Kommunikation (RS232).....	29
5.7.3	Grundlegendes zum Microcontroller.....	30
5.7.4	Programm des Microcontrollers.....	30
5.7.5	Inbetriebnahme.....	32
5.7.6	Ablauf einer Datenübertragung.....	33
5.7.7	Visual C++-Beispielcode	34
5.8	Modifikationen am Gesamtsystem.....	36

6	Visual C++-Implementation	38
6.1	Einleitung.....	38
6.2	Neues Projekt.....	38
6.3	Video Capture.....	38
6.4	Steuerung über Interface.....	40
6.5	Matrix-Klassen in C++	41
6.6	MS Visual Development Kit.....	42
6.7	Geplanter Programmablauf	42
7	Persönliches Fazit.....	44
7.1	Ausgangslage.....	44
7.2	Was erstellt wurde	44
7.3	Erkenntnisse.....	44
7.4	Dankeschön	44

1 Intro

Ein ferngesteuertes Modellauto ist mit einer Wireless-Kamera ausgestattet und soll selbständig einem Strassenverlauf folgen.

Die Fahrtstrecke des Autos wird mittels einem hellen, schmalen Malerlebeband vorgegeben. Die Kamera sendet nun Bilder dieser Fahrtstrecke an einen Grabber, welcher mittels USB-Anschluss mit einem Notebook verbunden ist. Das heisst, die Bilder können vom Grabber in das Notebook eingelesen werden. Ziel ist es nun, die Fahrtstrecke, welcher das Auto zu folgen hat, mittels Bildverarbeitung zu erkennen. Wird die Strecke erkannt, so werden dem Fahrzeug vom Rechner über die Fernsteuerung die nötigen Steuersignale übermittelt. Auto, Kamera und Rechner bilden also einen eigenständigen Prozess.

2 Aufgabenstellung

Unser Team leistet bei dieser Semesterarbeit keine Pionierarbeit, sondern stützt sich auf bereits vorhandene Erkenntnisse früherer Arbeiten mit diesem Problem. Die bestehende Implementation ist im Bezug auf die Bildverarbeitung etwas zu unstabil. Dies bedeutet, dass bei schlechten Lichteinflüssen oder störenden Objekten im Bereiche der Fahrtstrecke keine Spur auf dem eingelesenen Bild der Fahrtstrecke mehr erkannt werden kann und deshalb dem Auto keine Steuersignale mehr übermittelt werden können, was unweigerlich zum Prozessabbruch führt.

Demzufolge lautete unser Auftrag, den Bildverarbeitungsprozess zu verbessern. Konkret heisst dies erstens, die Strecke auf dem Bild aufzuspüren und zweitens, eine mathematische Funktion zu „kreieren“, welche dem Streckenverlauf ähnlich sieht. Bisher stützten sich nämlich die errechneten Steuersignale direkt auf den Graustufen-Pixel des Bildes selber. Diese Lösung hat sich als nicht sehr robust herausgestellt. Hingegen die Steuersignale auf mathematischen Berechnungen basieren zu lassen und damit vom eingelesenen Bild abzukoppeln, erscheint elegant.

3 Erste Entwürfe in Matlab

Matlab ist ein numerisches Rechenprogramm, welches einfach zu handhaben ist. Es erlaubt mathematische Anwendungen mittels einfachen Befehlen auszudrücken und grafisch darzustellen. Ebenfalls erlaubt es Bilder in unkomprimierter Form einzulesen (z.B. Datentypen wie tif oder bmp) und zu bearbeiten. Da diese Anwendung unserem Tätigkeitsfeld entspricht, war es sinnvoll erste Ideen der Bildverarbeitung in dieser anwenderfreundlichen Programmiersprache zu implementieren.

3.1 Kantendetektion

Im Folgenden wird beschrieben, wie wir unser Problem angegangen haben. Zuerst werden einige Bilder von der Kamera mit dem Streckenzug in Matlab eingelesen (Abb. 1).



Abbildung 1: Originalbild

Was wollen wir mit dem Bild? Ziel ist es, die helle Fahrspur auf dem Bild zu finden. Hierzu nützen wir die Tatsache aus, dass sich die Fahrspur an ihren Kanten deutlich von der dunklen Umgebung abgrenzt. Wir versuchen also diese Kante auf dem Bild zu detektieren (Kantendetektion).

Mit dem Originalbild kann eigentlich noch nichts angefangen werden. Matlab interpretiert dieses Bild als dreidimensionale Zahlenmatrix, das heisst drei hintereinander liegenden Matrizen mit je 240x320 Bildwerten. Jede der drei Matrizen repräsentiert eine Farbebene (rot, grün und blau) wodurch ein Farbbild entsteht. Aus dem Farbbild entsteht ein Graustufenbild, wenn man als Bezug nur noch eine Ebene wählt (Abb. 2). Hierfür addiert man die drei Farbebenen punktweise und teilt die neuen Werte der einen Ebene durch drei. Die Matrizenwerte eines Graustufenbildes liegen zwischen 0 und 255, wobei 255 ein weisser Bildpunkt, 0 ein schwarzer Bildpunkt darstelle. Werte dazwischen sind Grautöne.



Abbildung 2: Graustufenbild

Im nächsten Schritt sind die beiden Kanten, welche die Spur von der Umgebung abgrenzen, zu finden. Hierfür verwenden wir einen sog. Sobel-Operator, wie diesen schon unsere Vorgänger eingesetzt hatten. Dazu wird eine 3x3-Matrize pixelweise vertikal und horizontal über das Graubild verschoben. Für jede Verschiebung wird einem Pixelwert ein neuer Wert zugeordnet. Diese Sobel Operation hat nun den Effekt, dass hohe Zahlenunterschiede zwischen zwei Nachbarpixel im neu entstandenen Bild deutlich zu sehen sind (Abb. 3). Die Sobel Operation entspricht also einem Hochpassfiltern des Bildes.

Damit sich feine Störungen im Graubild nicht stark auswirken, wird der Sobel Operation noch ein Tiefpassfilter vorgeschaltet. Das Tiefpass filtern eines Bildes ist so zu verstehen, dass ein Pixelwert der Werte seiner Nachbarpixel angeglichen wird

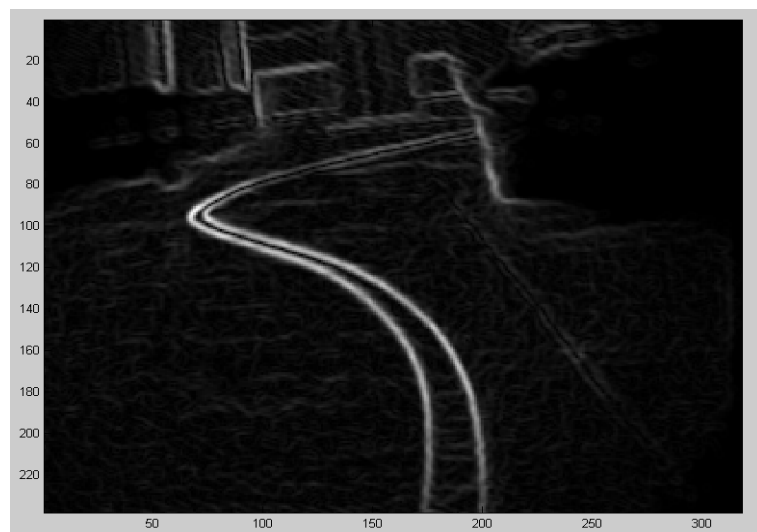


Abbildung 3: Bild nach Sobel-Operation

3.2 Autokorrelation

Nach Betrachtung von Abbildung 3 kam folgende Idee auf. Was entstünde, wenn dieses Bild autokorreliert würde? Das Sobelbild wird also pixelweise von links nach rechts über sein eigenes Bild gezogen bis sich die beiden Bilder decken. Bei jeder Verschiebung (Das Bild enthält in der Vertikalen 318 Pixel \rightarrow 318 Verschiebungen) werden alle Bildwerte elementweise multipliziert und addiert. Die so entstehende Zahl wird gespeichert. Damit dies möglich ist wird das Sobelbild zu seiner Linken noch mit einer ebenso grossen Nullermatrize ergänzt. Da bekannt ist, dass helle Bildpunkte einen grösseren Wert haben als dunkle, so ist es erdenklich, dass das Maximum der Summe zwar dort liegt, wo die Bilder aufeinanderliegen, es aber vorher noch einen Punkt geben muss, der ebenfalls eine grosse Zahl als Summe beinhaltet. Dieser Punkt ist dort, wo sich die beiden hellen Kanten überlappen.

Zur Veranschaulichung ein kleines Beispiel:

Es sei ein Vektor $a=[1 \ 1 \ 5 \ 0 \ 0 \ 1 \ 5 \ 2 \ 0]$, der stark vereinfacht unsere Bildmatrize darstelle. Die beiden höchsten Werte im Vektor sollen die Kanten der Spur darstellen, die anderen Werte deren Umgebung. Nun entsteht ein neuer Vektor b mit ebenfalls neun Werten, gebildet anhand der obigen Beschreibung.

Es entsteht $b=[0 \ 2 \ 7 \ 16 \ 26 \ 5 \ 7 \ 21 \ 57]$

Hier wird ersichtlich, dass der zweithöchste Wert von b an der Stelle liegt, an welcher sich die beiden Kanten überschneiden. Diese Stelle wird im weiteren als Zwischenmaximum bezeichnet.

Der Algorithmus dieser Korrelation wurde noch verbessert, indem nicht das ganze Bild auf einmal verschoben wird, sondern abschnittsweise. Also, zuerst beispielsweise Zeilen 1-20, dann 21-40 usw. Die Verbesserung deshalb, weil die Kanten auf dem Bild wegen der Kameraperspektive nicht parallel sind. Das Resultat sehen wir in Abbildung 4. Es ist zu sehen, dass im unteren Bildbereich eine schöne Linie zu sehen ist, welche unsere korrelierte Spur repräsentiert. Im oberen Bereich scheint der Algorithmus aber zu versagen. Dies deshalb, weil in diesem Bereich aufgrund der Perspektive die Kanten sehr nahe beieinander sind und kein Zwischenmaximum gefunden werden kann (oder es wird eine Summe an einem falschen Ort gefunden).

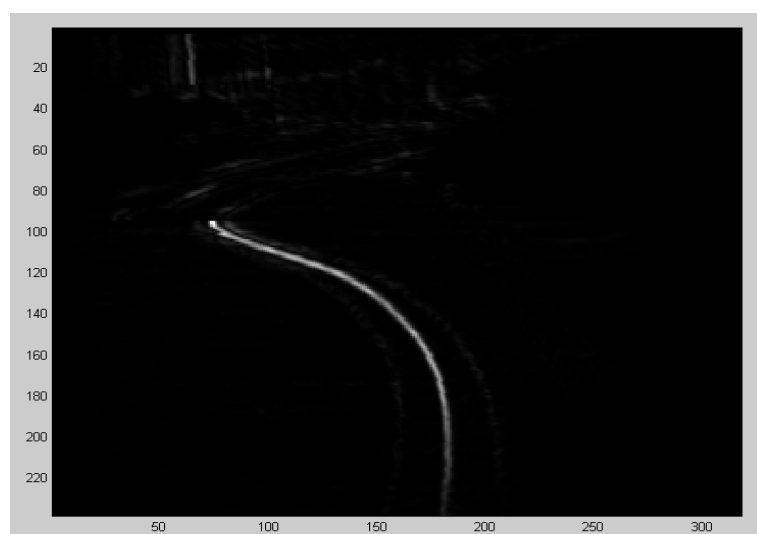


Abbildung 4: Bild nach Autokorrelation

3.3 Folgerungen

Eingangs wurde erwähnt, dass die Steuerung des Autos möglichst unabhängig von den Pixelwerten des Bildes arbeiten soll. Wir wollen und müssen zwar mit den Werten rechnen, sollten aber möglichst lange unabhängig von absoluten, einzelnen Werten bleiben. Die Suche des Zwischenmaximums in der Autokorrelationsfunktion ist nichts anderes als eine Schwellwertdetektion. Das heisst, wir suchen ein zweites Maximum in der Summe, welches nicht das Maximum der Funktion ist, wenn beide Bilder aufeinanderliegen. Diese Detektion verletzt die erwähnte Bedingung, da wir mit Maximas und Bedingungen rechnen. Daher gesehen stellt die Autokorrelation, wenngleich die Idee durchaus elegant erscheint, nicht die gesuchte Lösung dar, so dass künftig anderen Möglichkeiten nachgegangen werden müssen.

Eine andere Möglichkeit wäre beispielsweise eine robuste, lichtunempfindliche Kantendetektion. Verschiedene Arten wurden schon von unseren Vorgängern untersucht (Prewitt, Roberts). Wir unsererseits fanden im Canny-Verfahren eine passable Alternative. Dieses Verfahren ist schon in Matlab implementiert und scheint erstaunlich robust zu sein. Da Robustheit seinen Preis hat, wäre noch zu untersuchen, ob dieses Verfahren überhaupt in einer befriedigenden Laufzeit zu realisieren wäre.

Nichtsdestotrotz versuchten wir ein Programm zu erstellen, das an einem geeigneten Kurvenbild eine Hermite-Spline durchlegt. Das Programm sucht in jeder Zeile das Maximum und nimmt diese als Bezugspunkte um die geeigneten Koeffizienten für die Polynomfunktion der Art $ax^n + bx^{n-1} + \dots + yx + z$ zu finden. Da aber diese Funktion abhängig von x ist können 180° Kurven nicht dargestellt werden. Es ist jedoch zu sagen, dass solche Kurvengebilde enge Radien voraussetzen, welchen das Fahrzeug gar nicht folgen kann.

Dieses Programm ist aber erst ein Prototyp und verlangt eine klare Verbesserung, denn mit einer Spline kann man wichtige Punkte, beispielsweise Punkte unmittelbar auf der Strecke vor dem Auto, gewichten, indem dessen Gleichungen im zu lösenden Gleichungssystem mehrfach vorkommen.

Das Hermite-Spline Programm wurde auf das Korrelationsbild (Abb. 4) angewandt. In Abbildung 5 sind die Maxima dieser Funktion zu sehen. Gut ersichtlich wie die Maxima im oberen Bildbereich hin- und herspringen. Dies rührt, wie erwähnt, von der schlechten Korrelation, resp. vom nicht auffinden des Zwischenmaximums in der Korrelationsfunktion.

Abbildung 6 zeigt die entstandene Spline. Gewählt wurde eine Spline vierter Ordnung.

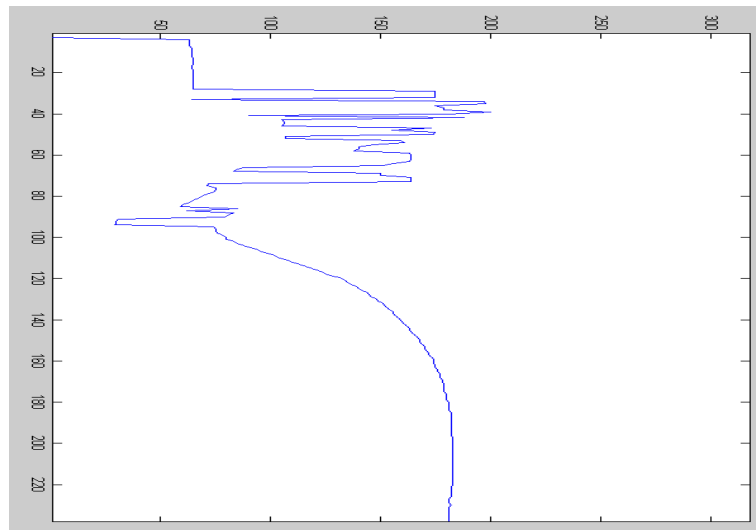


Abbildung 5: Maxima des Korrelationsbildes

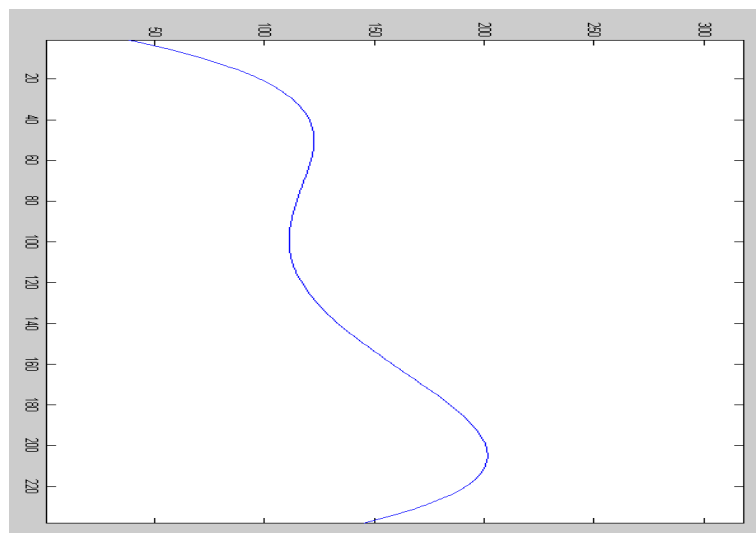


Abbildung 6: Hermite-Spline vierter Ordnung

4 Das Matlab-File test.m

Als wir einsahen, dass die Lösung mittels Autokorrelation nicht zum gewünschten Ergebnis führt, stellte unser Betreuer ein Matlab-File namens test.m zur Verfügung, welches unser Problem elegant löste. Einerseits legt dieses Programm eine Spline wünschbaren Grades durch die gefundene Spur, andererseits werden die Algorithmen in einer akzeptablen Zeit abgearbeitet. Sogar durch extreme künstliche Verrauschung dieses Bildes (keine Spur mehr sichtbar), wird die Spur detektiert und mathematisiert.

Nachfolgend sollen die wichtigsten Operationen dieses Files beschrieben werden.

4.1 Abtastung des Bildes

Ausgangssituation ist das eingelesene Kamerabild in Matlab. Danach wird es zur Bearbeitung bereitgestellt. Es wird beispielsweise in ein Graubild umgewandelt, wie in Kapitel 3 bereits beschrieben wurde. An dieser Stelle könnte man noch systematisches Rauschen dazuaddiert werden. Darauf soll aber nicht genauer eingegangen werden, da es keinen Einfluss auf die Abfolge des Bildverarbeitungsprozesses hat. Vielmehr ist es eine Grösse, von welcher man ausgehen sollte, um ein robustes System zu entwickeln. Rauschen auf Bildern kann durch Lichteinflüsse entstehen und äussert sich in vielen, zufällig verteilten, weissen Bildpunkten, vergleichbar mit einem Fernsehbild während eines Gewitters. Diese zufällige Verteilung nennt man systematisch. Auf solche Fehler reagiert das Programm robust. Es gibt aber auch unsystematische Fehler, z. B. Kratzer auf der Kameralinse. Solche Fehler hingegen haben einen starken Einfluss auf die Berechnung der Kurve.

Das Bild besteht wie erwähnt aus 240x320 Bildpunkten (Pixel). Um den Rechenaufwand zu dezimieren, ist es möglich beispielsweise nur jeden vierten Pixel in der Vertikalen und Horizontalen zu berücksichtigen. Dies nennt man Abtasten des Bildes und die Zahl vier heisst Abtastrate. Jedes Abtasten, sei es bei einem Zeitsignal oder etwas abstrakter bei unserem zugrundeliegenden Bild, setzt das einhalten des Abtastkriteriums voraus. Dies besagt, dass das Abtasten eines Signals nur erlaubt ist, wenn die Abtastfrequenz im Minimum doppelt so gross ist wie die höchste Frequenz des Signals.

Dies gilt auch für ein Bild. Deshalb wird dieses vorher mit einem Tiefpass gefiltert. Darunter versteht man, dass jeder Pixel mit seinen Nachbarn addiert wird. Dieses Resultat wird noch mit der Anzahl involvierter Pixel dividiert. Der quadratische Abtastwert (hier $4^2 = 16$) ergibt die Anzahl Nachbarn, welche zu berücksichtigen sind. Dem entstandenen Bild sagen wir Intensitätsbild. Dies nur, damit später bekannt ist wovon gesprochen wird.

Anmerkend sei an dieser Stelle noch gesagt, dass das filtern eines Bildes langsam ist. Die filter2-Funktion in Matlab, welche hierfür benutzt wird, prüft aber den Rang der Filtermatrix. Sind nämlich abhängige Vektoren in der Matrize vorhanden, so dezimiert sich der Rechenaufwand drastisch.

4.2 Kantendetektion

Das wichtigste wurde schon erwähnt. Es gibt viele Methoden, eine Kante in einem Bild zu detektieren. Es sind dementsprechend viele Algorithmen vorhanden. Viele von ihnen sind sich ähnlich. Sie funktionieren im Grunde wie der erwähnte Sobel-Operator, wenngleich die Filtermatrizen verschieden aufgebaut sind.

4.2.1 Gewöhnlicher Hochpass

Einfachste Implementation eines Hochpasses. Ruft wie der Tiefpass die filter2 Funktion auf. Die Filtermatrize ist jedoch so aufgebaut, dass Helligkeitsänderungen im Bild erkannt werden.

4.2.2 Gauss'scher Hochpass

Die Werte der Filtermatrize werden anhand einer Gaussglocken-Funktion (Abbildung 7) bestimmt. Die Idee dahinter besteht darin, dass primär die Differenz zwischen tiefen und hohen Werten in einer Bildmatrize grösser wird. Vom neuen Bild werden die Gradienten gebildet. Die Gradienten sind numerische Ableitungen in beiden Bildrichtungen. Grosse Werteunterschiede entsprechen hohen Werten in den Richtungsgradienten.

Das Kantenbild setzt sich also aus der Summe beider Richtungsgradienten zusammen, welche übrigens dieselbe Grösse wie die Bildmatrize haben. In der Regel werden die quadrierten Werte der Gradienten genommen, sodass das Kantenbild nur positive Werte enthält.

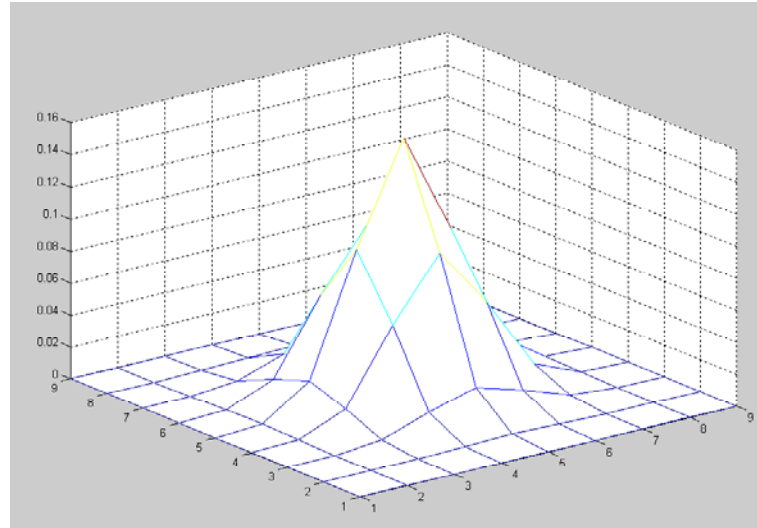


Abbildung 7: Gaussglocke

4.3 Streckenzug durch Spline ersetzen

4.3.1 Hermite Splines

Zu Beginn der Arbeit wurde uns nahegelegt, die Spur durch eine Hermite-Spline zu mathematisieren. Eine Hermite Spline besitzt einen Polynomial-Charakter und hat folgende Form:

$$h(s) = \begin{pmatrix} x(s) \\ y(s) \end{pmatrix} = \begin{pmatrix} a_x s^n + b_x s^{n-1} + \dots + y_x s + z_x \\ a_y s^n + b_y s^{n-1} + \dots + y_y s + z_y \end{pmatrix}$$

Es ist ersichtlich, dass die Spline aus zwei unabhängigen Funktionen besteht, welche die selbe Variable s gebrauchen. Nun wird die eine Funktion in einem Koordinatennetz in der Abszisse und die andere in der Ordinate eingetragen. Lässt man die Variable s nun einen beliebigen Bereich durchlaufen, z.B. $s=[0, 0.1, 0.2, \dots, 1]$ und interpretiert die entstehenden Funktionswerte als Koordinaten eines Vektors, so wird ersichtlich, dass man so beliebige Kurvenformen erzielen kann. Je höher der Grad der Polynome desto „verrückter“ wird die Spline. Abbildung 8 zeigt ein Beispiel einer Spline fünfter Ordnung.

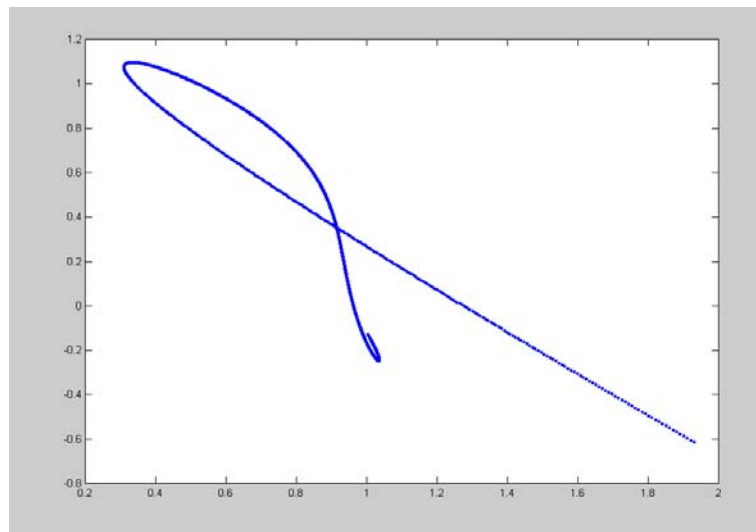


Abbildung 8: Hermite-Spline fünfter Ordnung

Der Loop in Abb. 8 ist nur möglich, weil die beiden Funktionen unabhängig voneinander sind. So liesse sich problemlos jedwede Kurve beschreiben. Nach genaueren Betrachten unseres Problems, mussten wir aber feststellen, dass wir dies gar nicht brauchten. In erster Linie deshalb nicht, weil so extreme Kurven gar nicht existieren, respektive von unserem Fahrzeug nicht gefahren werden können. Hauptsächlich brauchen wir derartige Approximationen nicht, weil uns nur der unmittelbar vorausliegende Streckenteil interessiert, und nicht was einige Meter weiter vorne passiert.

Aus diesen Gründen können wir auf die Hermite-Splines verzichten. Stattdessen kann das Problem mit einer einzigen Polynom-Funktion beschrieben werden. Da die Spur in vertikaler Richtung auf dem Bild erscheint, kann die obere linke Ecke als Nullpunkt, die Bildhöhe als Abszisse mit der Laufvariablen und die Bildbreite als Ordinate mit der Funktion der Laufvariablen interpretiert werden. Der Verzicht auf die Unabhängigkeit beider Achsen des Koordinatennetzes spart auch Rechenaufwand ein.

Das Polynom, welches wir legen wollen soll niederen Grades sein. Nach weiterer Betrachtung und Systemeinschränkung wird ersichtlich, dass sogar eine Gerade unser Problem durchaus zufriedenstellend löst.

4.3.2 Normalisieren der Bilder

Um das Polynom zu berechnen, werden zwei bereits erwähnte Bilder benutzt. Das Intensitätsbild und das Kantenbild. Wir wissen ebenfalls, dass wir eine Gerade suchen, höchstens aber ein Polynom zweiten Grades.

Zunächst werden die beiden Bilder aber noch normalisiert. Die beiden Bilder enthalten durch die verschiedenen Filtermethoden unterschiedlich proportionierte Bildwerte. Damit danach mit „gleichen Ellen“ gerechnet werden kann, müssen die beiden Bilder aufeinander abgestimmt oder eben normalisiert werden. Dies erreicht man durch Subtraktion des kleinsten Bildwertes an allen Bildwerten. In der Regel ist irgendwo im Bild ein schwarzer Bildpunkt vorhanden, sodass das Minimum also gleich null ist. Um die Proportionen der Bilder aufeinander abzugleichen, werden dessen Werte anschliessend durch die Standardabweichung derselben dividiert.

Standardabweichung einer Bildmatrize:
$$std = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$
 mit

- x_i als Abweichung des Einzelwertes vom Durchschnittswert
- \bar{x} als Durchschnittswert aller Bildwerte
- n als Anzahl Elemente der Bildmatrize

Ausserdem betrachten wir nur die unteren dreiviertel des Bildes, da uns den entfernten Streckenteil nicht interessiert.

4.3.3 Darstellung von Gleichungen in Matrizen

Um mathematische Aufgaben, in welchen beliebig viele Variablen vorkommen können, zu lösen, muss man Gleichungen aufstellen. Es ist bekannt, dass man ebenso viele, voneinander unabhängige Gleichungen benötigt wie Variablen. Bei vielen Unbekannten wird das Auflösen der Gleichungen von Hand bald einmal zu einem unlösbaren Problem. Deshalb greift man gerne einmal zu einem Rechner, obwohl auch dieser für n Gleichungen je nach Algorithmus von n^3 bis über $n \cdot (n+1)!$ Operationen benötigt.

Mit Matlab können Gleichungssysteme aufgelöst werden. Allerdings müssen Koeffizienten, Variablen und Resultate in Matrizen gefüllt werden. Einigen wir uns an dieser Stelle darauf, dass bekannte Werte in die A- und y-Matrize eingefüllt werden und die x-Matrize unsere gesuchten Zahlen beinhaltet. So kann ein Gleichungssystem

$$\begin{array}{l}
 a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = y_1 \\
 a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = y_2 \\
 a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = y_3
 \end{array}$$

geschrieben werden als

$$\begin{array}{c}
 A \cdot x = y \\
 \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}
 \end{array}$$

Der Matlab Befehl $x=A \setminus y$ löst das Gleichungssystem nach den drei Unbekannten auf. Nehmen wir nun an, im Gleichungssystem käme eine vierte Gleichung A_4 vor, welche gleich wäre wie die erste. Ihre Lösung unterschiede sich jedoch von der ersten und wäre y_4 . Das Auflösen des Systems würde die leere Menge ergeben, da die Lösung von A_4 nicht äquivalent den anderen Lösungen ist. Mit dem erwähnten Befehl von Matlab lässt sich x_1 bis x_3 aber trotz dieser vierten Gleichung bestimmen. Die Zahlen x_1 bis x_3 werden vom Programm so errechnet, dass die quadratische Fehlersumme der y -Zahlen minimal wird. Genau dieser Umstand ist bei unserer Bildanalyse äusserst hilfreich.

4.3.4 Regression

Eine Regression beschreibt eine optimale Annäherung einer Kurve oder Gerade an vorgegebene Messpunkte. Optimal soll bedeuten, dass die Differenz zwischen Messpunkt und Kurve möglichst klein bleibt (siehe auch 4.3.3).

Beispiel:

Die Messreihe in Abbildung 9 mit zehn Werten soll mit einem Polynom vierten Grades approximiert werden. Der erste Messwert sei bei $a=1$ der letzte bei $a=10$. Gesucht seien die Koeffizienten x_1 bis x_4 dieser Funktion.

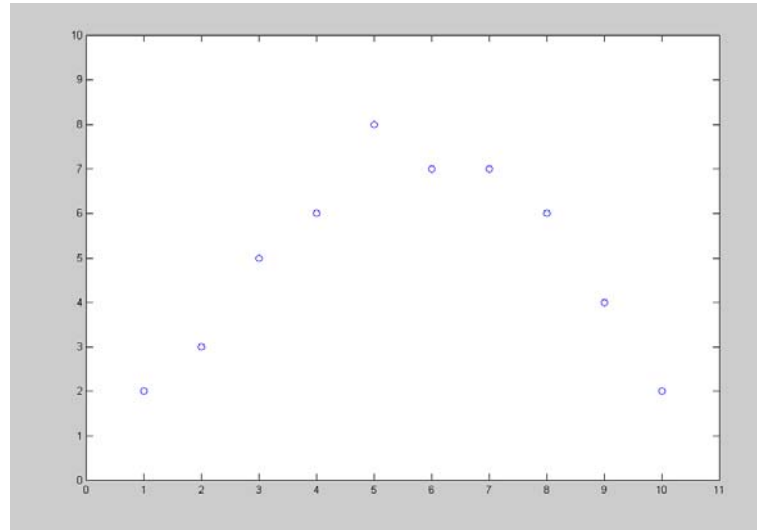


Abbildung 9: Messreihe

Lösung:

Da wir zehn Werte haben und das Polynom vierter Ordnung ist, besitzt unsere A-Matrix zehn Reihen und vier Spalten. Matrix y besteht dementsprechend aus zehn Reihen und einer Spalte:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \\ 64 & 16 & 4 & 1 \\ 125 & 25 & 5 & 1 \\ 216 & 36 & 6 & 1 \\ 343 & 49 & 7 & 1 \\ 512 & 64 & 8 & 1 \\ 729 & 81 & 9 & 1 \\ 1000 & 100 & 10 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 5 \\ 6 \\ 8 \\ 7 \\ 7 \\ 6 \\ 4 \\ 2 \end{bmatrix}$$

Grafik 1: Struktur der Matrix

Der mächtige Befehl in Matlab $x=A\backslash y$ liefert für die Koeffizienten x_1 bis x_4 :

$$\begin{aligned} x_1 &= -0.0132 \\ x_2 &= -0.0548 \\ x_3 &= 2.0796 \\ x_4 &= -0.3333 \end{aligned}$$

mit den optimalen y_1 bis y_{10}

$$\begin{aligned} y_1 &= 1.6783 & y_6 &= 7.3193 \\ y_2 &= 3.5012 & y_7 &= 7.0093 \\ y_3 &= 5.0559 & y_8 &= 6.0350 \\ y_4 &= 6.2634 & y_9 &= 4.3170 \\ y_5 &= 7.0443 & y_{10} &= 1.7762 \end{aligned}$$

Abbildung 10 zeigt das Resultatpolynom grafisch.

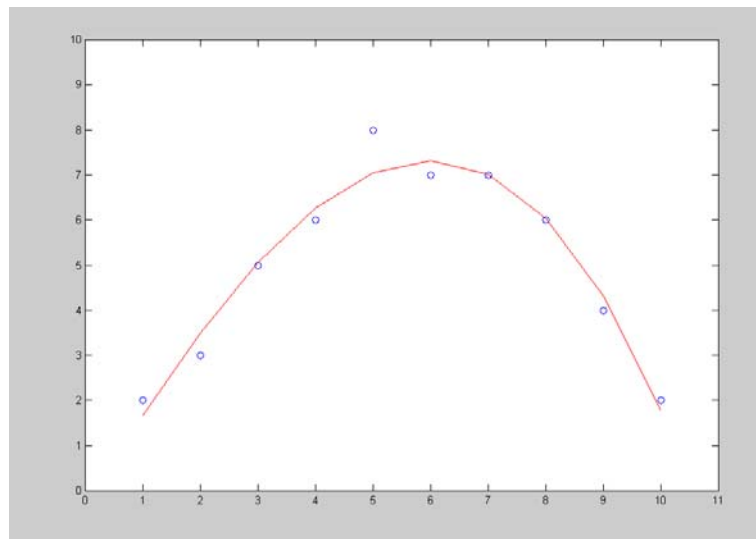


Abbildung 10: Resultatpolynom

Bemerkenswert erscheint, dass sogenannte Ausreisserwerte wie der Wert 8 einen kleinen Einfluss auf das Polynom haben. Auch dies kommt unserer Aufgabe entgegen. Sind beispielsweise helle Störobjekte neben der Spur auf dem Bild zu erkennen, so wirken sich diese nicht sonderlich schwerwiegend auf die errechnete Kurve aus.

4.3.5 Legen einer Kurve in einem Bild

Wie eine Kurve bestmöglichst verschiedenen Messwerten angeschmiegt wird, wurde eben erläutert. Die Frage ist nun wie man dies in einem Bilde realisiert. Dazu muss man sich fragen, welches die Unterschiede zwischen einer Messreihe aus Abb. 9 und einem Streckenbild, wie beispielsweise Abb. 1 darstellt, sind. Man kann sich hierfür folgendes vorstellen. Der Streckenzug stelle unsere Messreihe dar. Nun gibt es aber neben dem Streckenzug noch eine Umgebung, es handelt sich ja schliesslich um ein Bild. Wir wissen, dass ein Bild aus einer Zahlenmatrize aufgebaut ist. Da die Strecke hell ist, wird sie in der Matrize mit hohen Zahlen dargestellt. Die Umgebung erscheint dunkler, weswegen sie bedeutend tiefer bewertet wird als die Strecke. Und dies ist genau der Unterschied zur Wertereihe in Abb.9. Dort haben wir zum Beispiel bei $a=1$ genau einen einzigen Punkt $y=2$. Bei unserem Bild aber entspräche die Abszisse den Matrizenzeilen und die Ordinate den Matrizenzeilen. Unser Bild besteht aus 240 Reihen und 320 Spalten.

Dies heisst, dass zu $a=1$ also 320 Spaltenwerte zugeordnet sind, welche durch ihren Grauton gewichtet sind. Und genau diese Gewichtung haben wir bei der Messreihe nicht. Wozu auch, entspricht doch einem a immer auch einen Punkt auf y und nicht etwa mehreren Ypsilons. Folglich besitzt die A-Matrixe für das Bild nun auch nicht mehr nur a Zeilen sondern a mal y Zeilen.

Nachstehend wird anhand des Matlab Programms test erklärt, wie die A- resp. y-Matrizen aufgefüllt werden.

Anbei die Kürzel, welche im Programm-Segment vorkommen:

Pf	→	Kantenbild (als Matrix)
PP	→	Intensitätsbild (als Matrix)
order	→	Grad des Polynoms
rmax	→	Höhe des Bildes (Zeilen, Abszisse)
cmax	→	Breite des Bildes (Spalten, Ordinate)

```
A=zeros(rmax*cmax,order+1);  
y=zeros(rmax*cmax,1);
```

Grösse von A und y wird festgelegt. Zeilenanzahl ist $rmax*cmax$, Spaltenanzahl von A ist $order + 1$. Für eine Gerade wären es also zwei Spalten, für ein Polynom zweiten Grades deren drei. Matrix y besitzt nur eine Spalte.

```
for r=1:rmax  
  
    R=[1];  
    for o=1:order  
        R=[r^o, R];  
    end  
  
    for c = 1:cmax  
        index=(r-1)*cmax+c;  
        A(index,:)=r/rmax*PP(r,c)*Pf(r,c)*R;  
        y(index) =r/rmax*PP(r,c)*Pf(r,c)*c;  
    end  
  
end  
  
x=A\y;
```

Äussere for-Schleife:

Klein r geht von 1 bis r_{\max} .

Erste innere for-Schleife:

R baut für jedes r den Polynomvektor analog der Grafik 1 im letzten Kapitel auf. Dieser hängt vom Grad der Funktion ab und muss für jedes r neu gebildet werden. Beispielsweise für ein Polynom zweiten Grades:

$$\begin{aligned} r=1 & \rightarrow R=[1 \ 1 \ 1] \\ r=2 & \rightarrow R=[4 \ 2 \ 1] \\ r=3 & \rightarrow R=[9 \ 3 \ 1] \quad \text{usw.} \end{aligned}$$

Zweite innere for-Schleife:

Klein c geht von 1 bis c_{\max} . Matrizen A und y werden initialisiert. Eine Zeile von A wird gebildet aus dem Produkt eines Punktes (Wert) des Intensitätsbildes und eines Punktes des Kantenbildes und dem Polynomvektor R und eines weiteren Faktors. Der weitere Faktor heisst r/r_{\max} und ist maximal gleich eins. Die Punkte der Bilder werden also reihenweise in die A Matrize eingebaut (äussere for-Schleife), beginnend mit der ersten Zeile, welche den oberen Bildrand beschreibt. An jeder Zeile werden dann die dazugehörigen Spalten durchgegangen (zweite innere for-Schleife). Bei jeder Spalte wird nun das erwähnte Produkt gebildet und in A hineingeschrieben, wodurch r_{\max} mal c_{\max} Gleichungen entstehen.

Wenn man an der untersten Zeile des Bildes angelangt ist, dann ist $r=r_{\max}$ und der obige Faktor wird eins. Dieser Faktor ist nichts anderes als eine Gewichtung der Bildzeilen. Das heisst, je weiter unten die Zeile, desto mehr Einfluss hat sie auf das Resultat. Und das ist sinnvoll, denn uns interessieren die unteren Zeilen stärker als jene oben, da sie den unmittelbar bevorstehenden Streckenteil beschreiben.

Mit der y -Matrize wird genau gleich verfahren wie mit A . Zur Bildung von y wird R durch die aktuelle Spaltenzahl c ersetzt, denn es gilt die Beziehung $c=f(R)$.

Anmerkung:

Der Counter index geht von 1 bis r_{\max} mal c_{\max} und bezeichnet die aktuell Zeile von A , welche mit dem Produkt gefüllt werden soll. Die Bilder PP und Pf sind selbstverständlich gleich gross. $Pf(r,c)$ ist gleich dem Wert eines Bildpunktes in Zeile r und Spalte c des Bildes Pf .

Nachdem die Matrizen aufgefüllt wurden, kann wiederum der Befehl $x=A\backslash y$ angewandt werden. Die optimalen Koeffizienten werden errechnet. Die gefundene Funktion kann angepasst und über dem Graustufenbild aufgezeichnet werden. Abbildungen 11 und 12 zeigen Streckenapproximationen mit einer Gerade und einem Polynom zweiten Grades.

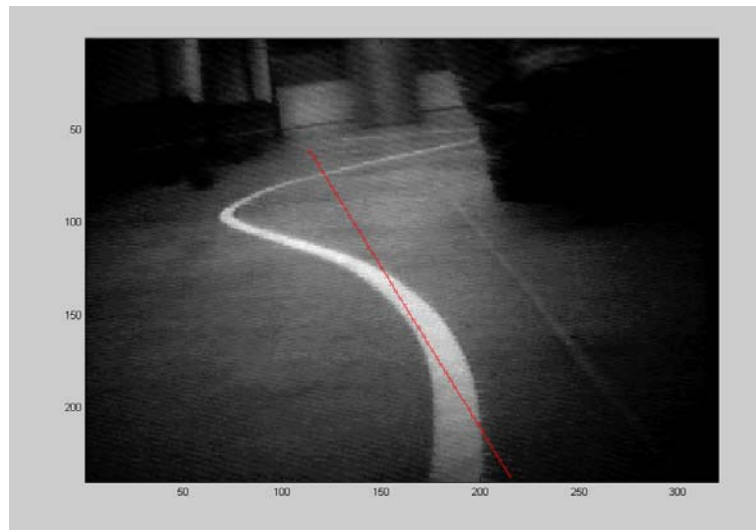


Abbildung 11: Gerade

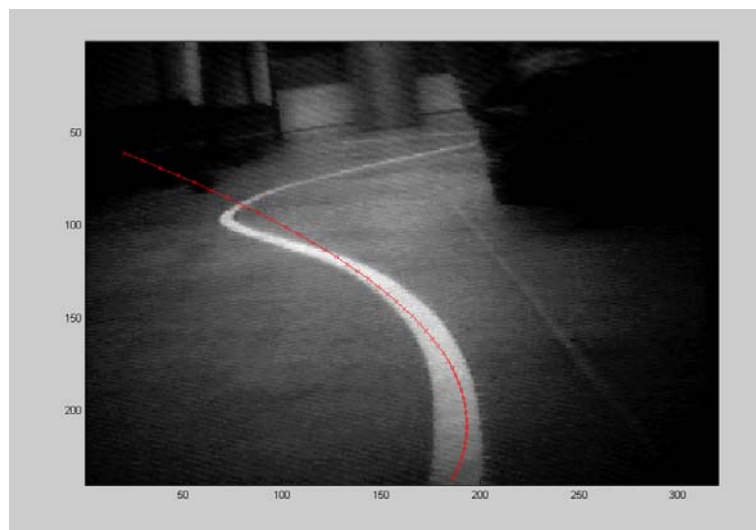


Abbildung 12: Polynom zweiter Ordnung

Aus Abbildung 12 ist zu erkennen wie das Polynom im unteren Bildbereich verzerrt ist. Daraus kann man schliessen, dass die Approximation an die Strecke nicht besser wird je höher die Ordnung des Polynoms ist. Das Gegenteil ist der Fall. Zwar sehen wir, dass auch die Gerade dem unteren Streckenteil nicht ganz zufriedenstellend folgen kann, wenn wir aber nur noch den unteren Viertel der Bildhälfte berücksichtigen, so wird die Annäherung besser (Abbildung 13). Natürlich wird so die Berechnung empfindlicher auf Störeinflüsse im unteren Bildteil.

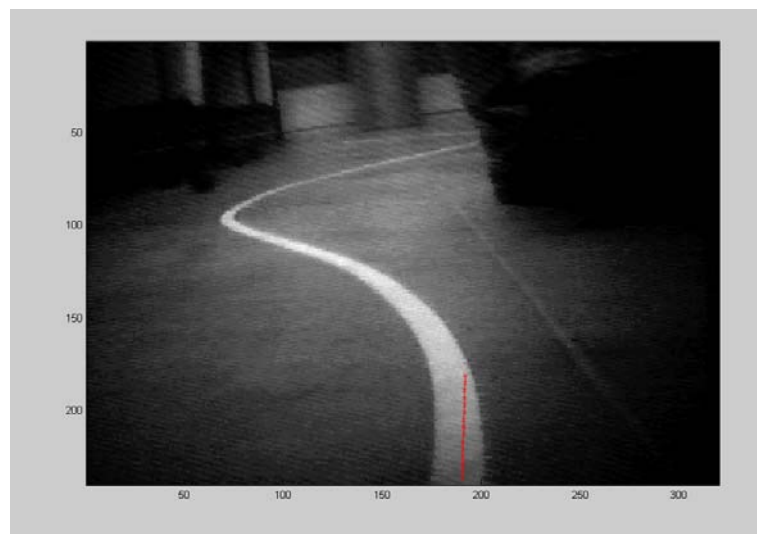


Abbildung 13: verbesserte Annäherung mittels Gerade

Die Abweichung der Geraden von der Vertikale ergibt einen bestimmten Winkel. Als ersten Versuch könnte dieser Winkel dem Einschlagwinkel des Autos entsprechen. Wenn genügend Bilder pro Sekunde bearbeitet werden könnten, wäre diese Möglichkeit eventuell schon die Lösung unseres Problems. Ansonsten müsste der Einschlagwinkel anders berechnet werden. Denn ein Einschlag an den Rädern mit einer Geschwindigkeit bedeutet einen integrierenden Prozess, das heisst bei Belassen des Einschlages drehte sich das Auto im Kreise. Bei langsamer Bildbearbeitung hiesse dies, dass das Auto sofort von der Spur geraten würde. Der Einsatz eines geeigneten Reglers müsste dann zur Lösung dienlich sein.

Bemerkung:

Das Matlab-File test.m sowie alle Entwurfsprogramme sind auf der Dokumentations-CD vorhanden.

5 Rund um die Hardware

5.1 Einleitung

Da diese Semesterarbeit auf vorhergehenden Arbeiten basiert, wurde zu Beginn angenommen, dass auf den bestehenden Lösungen aufgebaut werden kann. Diese Annahme stellte sich als Irrtum heraus, als die Anlage zum ersten Mal von uns in Betrieb genommen wurde. Aus unbekanntem Gründen zeigte die Funkübertragung zum Modellauto erhebliche Störungen beim Empfänger, was den ganzen Regelkreis zwischen Auto und Computer ausfallen liess. Das Auto führte Zitterbewegungen aus und reagierte nur noch unregelmässig auf Anweisungen des ablaufenden Programms auf dem Laptop. Grundsätzlich wurden diese Störungen dann sehr stark, wenn der Signal-/Rauschabstand beim Empfänger litt, weil die Fernbedienung zu weit weg stand oder leere Batterien die Senderfeldstärke verminderten.

5.2 Aufgetauchte Störungen

Die Störquelle wurden nach einigen sehr zeitaufwendigen Tests und Messungen im Labor der Kamera zugeschrieben. Im 27MHz-Band des RC-Trägers wurden auf dem Spektrumanalyzer Störsignale entdeckt, die von der Wireless Cam herrühren. Offenbar stammen diese von der Kameraelektronik, wobei z.B. Oberwellen der Bildwiederholungsfrequenz des CCD-Chips denkbar wären, denn das RC-Band ist vom Träger der Bilddaten auf 2.4GHz zu weit entfernt. Aber auch die AM-Anlage selber zeigte immer wieder sporadisch Störungen auf HF in der Umgebung, so dass es langsam schwierig wurde, die vielen Fehlerquellen zu isolieren und separat zu untersuchen.

Der Antenneneingang wurde mit einem Spektrum Analyser im Trägerfrequenzband untersucht. Der rechte der beiden dominanten Peaks bei der Center Frequenz von 27.145MHz gehört wahrscheinlich einem exakt eingestellten Lokaloszillator in der Empfängerschaltung, während die linke Frequenzspitze das eingestrahlte AM-Signal darstellt (siehe Abbildung 1).

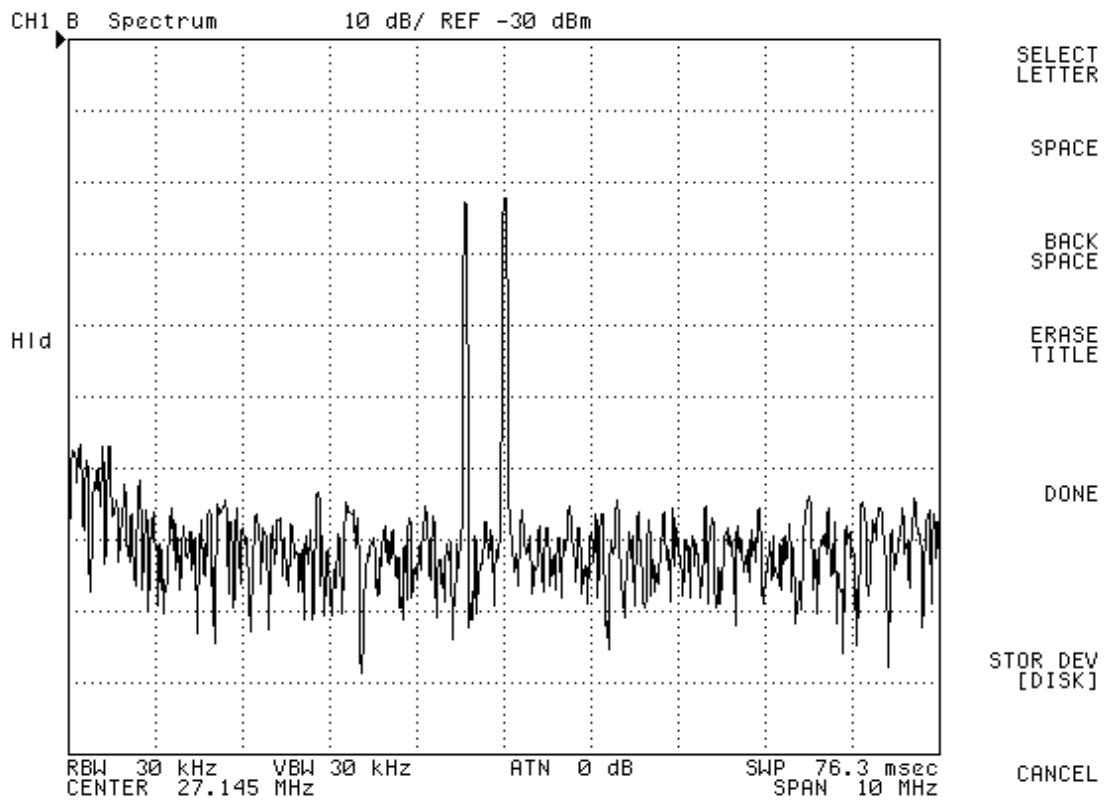


Abbildung 2

Wenn die Kamera eingeschaltet wurde, zeigten sich dort deutliche Interferenzen mit Leistungen, die sogar fast die Hälfte der maximal eingestellten Trägerschwingung erreichten (siehe Abbildung 2). Sie scheinen sich im oberen Band aufzulösen, stammen also entweder von irgendwelchen Intermodulationen oder sind Harmonische einer tieferen Grundschwingung.

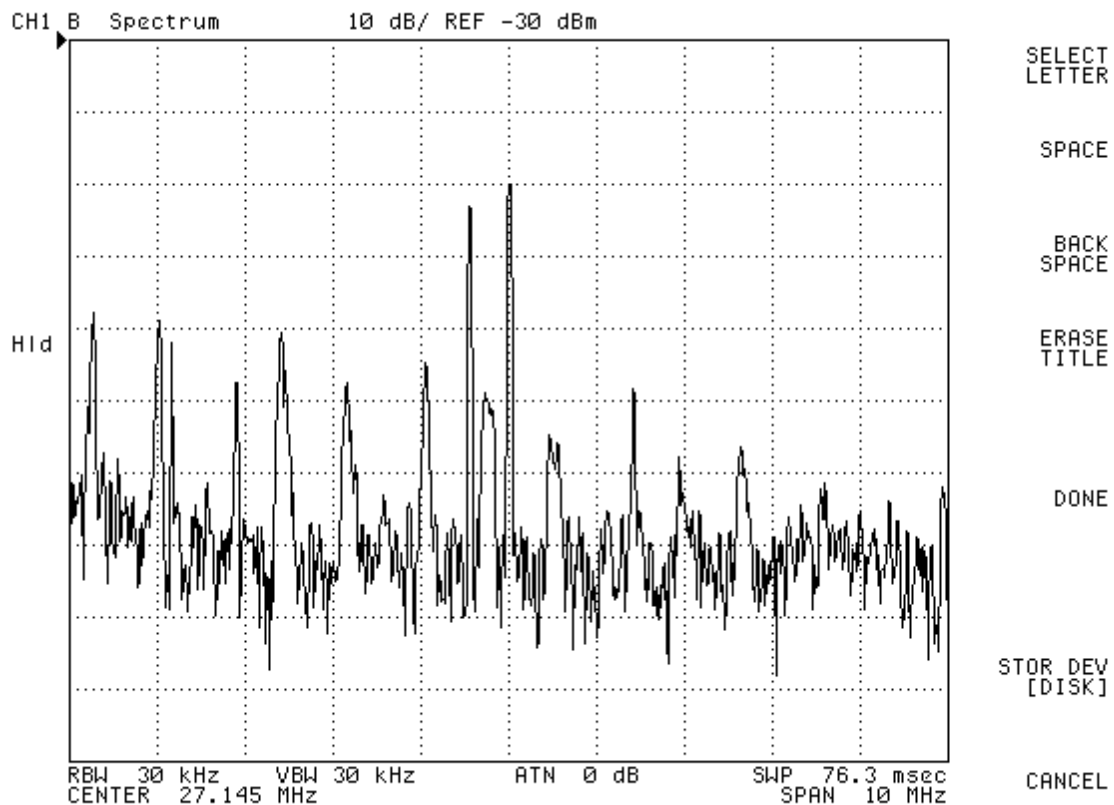


Abbildung 3

5.3 Wechsel auf FM

Schlussendlich wurde beschlossen, ganz auf AM zu verzichten und stattdessen eine FM-Fernbedienungsanlage anzuschaffen. Da die neue Sendefrequenz von 40MHz neben der Robustheit von FM viel weniger von der Kameraelektronik beeinflusst und ausserdem auch die Empfindlichkeit auf sonstige HF-Störungen in der Umgebung drastisch reduziert wird, begnügten wir uns vorläufig nur mit dieser Modifikation und verzichteten darauf, auch die Kamera gegen ein leistungsfähigeres Modell auszutauschen. Erste Tests im Modellbauladen mit der neuen RC-Anlage im gemeinsamen Betrieb mit der Kamera verliefen erfolgreich und bestätigten die vermuteten Probleme.

Nach dem Auswechseln des Empfängers und ersten Fahrversuchen mit eingeschalteter Kamera wurde jedoch klar, dass nach wie vor eine Störquelle vorhanden war, die jedoch durch die Robustheit von FM teilweise erheblich ignoriert werden konnte und nur einen Einfluss hatte, wenn sich das Sendersignal verlor. Nach erneuten Messungen und Versuchen wurde deutlich, dass das Problem ursprünglich dem falschen Objekt zugeschrieben worden war. Nicht die eigentliche Kameraelektronik war der Haken, sondern deren Stromversorgung. Die Kamera muss nämlich mit +12V gespeist werden und weil im Modellbau die 7,2V-Akkus gebräuchlich sind, hat die Vorgängergruppe einen DC/DC-Konverter eingebaut, der die Fahrspannung des RC-Cars verdoppelt und so den Einbau einer zusätzlichen Spannungsquelle ersparte.

Dabei wurde der Konverter schon früher ausgemessen, allerdings im stabilen Zustand und nur für Spannungen an Ein- und Ausgang mit dem KO. Der Ausgang verhält sich tatsächlich auch unter der Last der Kamera sehr stabil, sogar im Störfall. Eingangsseitig treten aber oft Schwingungen mit Spannungsamplituden von über 1V und nicht feststellbarer Grundfrequenz auf, wobei nicht klar ist, ob diese von der Servo- bzw. Empfängerelektronik oder vom Konverter angefacht werden. Es scheint, dass die Schaltkreise durch gegenseitige Mitkopplung in Schwingung geraten. Messungen der Ströme beim Konverter zeigten ausserdem, dass dieser Baustein fast dreimal soviel Leistung umsetzt wie die Kamera benötigt (ca. 70mA). Diese Verlustleistung wird wahrscheinlich grösstenteils über die angeschlossenen Kabel als HF abgestrahlt, denn als der Spannungswandler mit separater Quelle inklusiv sämtlichen Zuleitungen in Alufolie eingepackt wurde, verschwanden die Interferenzen. Fazit: Der extrem schlechte Wandler störte die übrige Elektronik durch HF-Immissionen dermassen stark, dass nicht einmal FM funktionierte. Aus diesem Grund, wegen der hohen Wandlungsverluste und der besseren Entkopplung vom unter Fahrbedingungen schwer belasteten Hauptstromkreis wird die Kamera jetzt mit einem 8x1,2V-NiMH-Akkupack betrieben, das dank seiner hohen Kapazität eine Mindestspannung von 9V aufrechterhalten kann. Diese Akkus können auch zusammen mit jenen für die Sendeanlage mit dem gleichen Ladegerät regeneriert werden.

Nach einigen Tests im Vollbetrieb und mit künstlich geschwächten Sendersignalen zeigte das modifizierte Modell definitiv keine Störungen mehr!

5.4 Umbau der Steuerung

Durch den Austausch der Fernbedienung musste natürlich auch das Interface mit dem Computer erneuert werden. Beim alten Gerät wurden die Steuersignale vom Laptop mit einer an der PCMCIA-Schnittstelle angeschlossenen D/A-Karte in analoge Spannungen umgewandelt und anstelle der Steuerknüppel-Potis direkt in die Schaltung eingespiesen. Dabei wurde klar, dass diese Methode auch recht störungsanfällig war und ausserdem häufig einen Abgleich benötigte. Mit der neuen FM-Fernbedienung entfallen diese Unannehmlichkeiten, denn das Gerät verfügt über eine Lehrer-/Schülerbuchse. Diese Schnittstelle macht es möglich, den manuellen Betrieb zu deaktivieren und die Servosignale dem FM-Modulator extern zukommen zu lassen.

Wegen der benötigten L-/S-Schnittstelle musste eine FM-Anlage im 40MHz-Band für Modellflugzeuge gewählt werden, die 7 Kanäle übertragen kann. Wir benötigten für das Modellauto nur 2 Kanäle, einer für die Drehzahl und die Lenkung. Das Lenkservo wurde beim RF-Empfänger auf Kanal 1, der Fahrtregler auf Kanal 2 festgelegt.

5.5 Lehrer-/Schülerbetrieb

Der Lehrer-/Schülerbetrieb funktioniert so, dass der Lehrer- und Schülergerät verbunden sind und der Schüler das Modell steuert. Die Stellsignale werden beim Schüler erzeugt und vom Lehrergerät moduliert und ausgestrahlt. Sobald der Lehrer den Trainer-Schalter auf seiner Fernbedienung loslässt, verliert der Schüler die Kontrolle und die Steuerknüppel beim Lehrergerät werden wieder wirksam.

Abbildung 3 zeigt eine solche Konfiguration aus der Sicht von hinten auf die Gehäuserückseite.

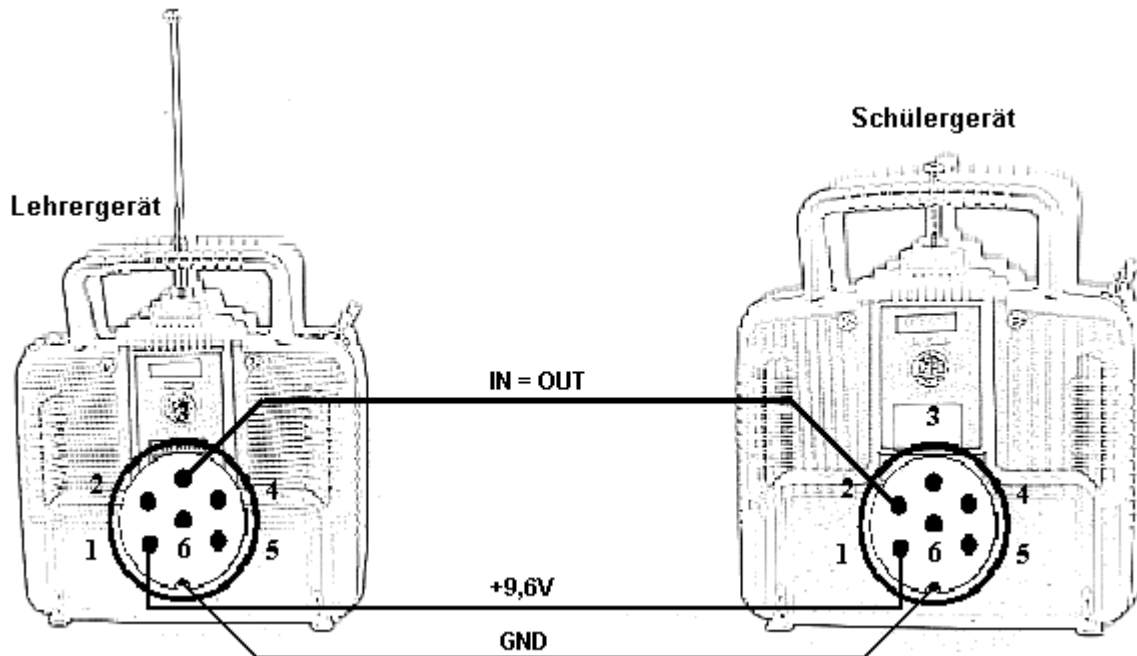
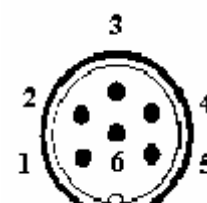


Abbildung 4

Die L-/S-Buchse hat 6 Anschlüsse, wobei wir nur 3 davon für unsere Anwendung wichtig sind einschliesslich Masse. Folgende Tabelle zeigt ihre Belegung:

L-/S-Buchse	Pin	Funktion
		Mantel/Schild
1		+9,6V (Supply Fernbedienung)
2		OUT
3		IN
4 & 5		Intern verbunden (N/A)
6		+5V (N/A)

Pin 1 liefert die Versorgungsspannung der Fernbedienung, in unserem Fall +9,6V und speist gleichzeitig unsere Interface-Schaltung. Pin 2 ist der Signalausgang mit dem Pulstelegramm der Knüppelstellung, aus der Sicht des Schülers betrachtet. Der wichtige Anschluss ist Pin 3, wo in der Regel die Schülersignale angelegt werden. Wir erzeugen diese Signale mit dem Interface selber. Pin 4+5 sind standardmässig fest verbunden und Pin 6, wo bei anderen Anlagen eine Spannung von +5V abgenommen werden kann, was für das Interface ideal wäre, in diesem Gerät nicht implementiert.

5.6 Pulstelegramm der Servo

Um herauszufinden in welcher Form die L-/S-Signale vorliegen wurde Pin 2 mit dem KO aufgezeichnet und die Auswirkungen der Knüppelbewegungen analysiert. Danach stand fest, welcher Puls zu welchem Kanal gehörte und in welchem Bereich sich die Rechtecklänge ändert. Zahlreiche Besuche bei Modellbauforen im Internet halfen dabei. Bei technischen Detailfragen zu Futaba-Geräten hat uns auch die Firma Spahr Elektronik gut beraten (Tel. 032 652 23 68).

In der Abbildung 4 ist ein Pulstelegramm vom Pin OUT aufgezeichnet. Gut ersichtlich ist periodische Form des Signals, das im Senderbetrieb noch mit der FM-Trägerwelle überlagert ist.

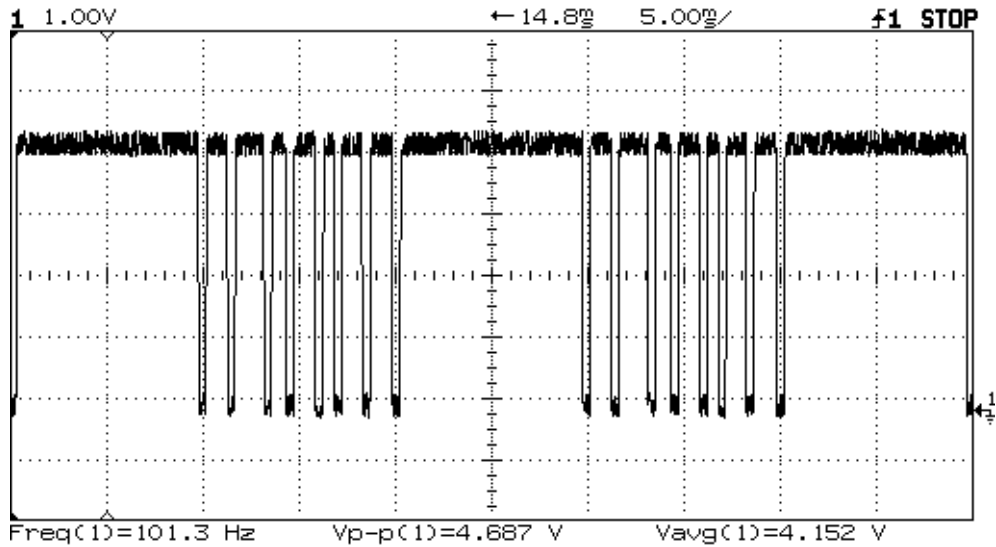


Abbildung 5

Abbildung 5 zeigt den Ausschnitt einer Pulsfolge aus Abbildung 4 etwas näher betrachtet. Klar erkennbar sind die konstanten Pulsabstände von 400µs.

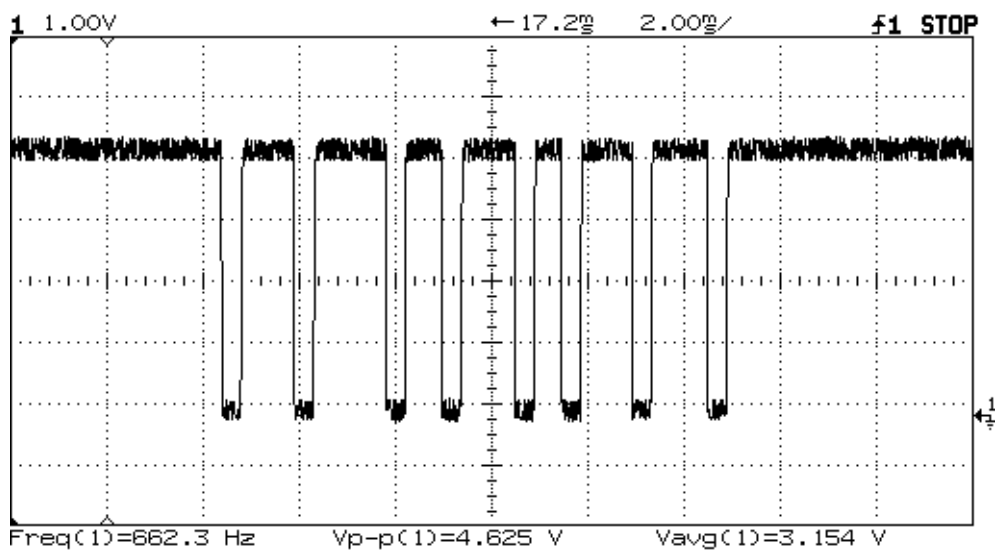


Abbildung 6

Die NF-Signale sind in TTL-Logik gehalten und PWM-moduliert (negative PPM). Die Länge eines Rechteckpulses (700us – 1500us), je einer pro Steuerkanal, stellt die Potistellung dar und bestimmt die Auslenkung des Servo, für das er empfängerseitig bestimmt ist. Die Pulslänge ist proportional zur Knüppelauslenkung und die Mittelstellung befindet sich demzufolge bei 1100us. Die Pulse werden in Form einer Pulsfolge multiplext periodisch übertragen, d.h. die Servo werden 50mal pro Sekunde mit dem Sender abgeglichen. Die Leitung steht während den Sendepausen auf hohem Potential. Eine anstehende Pulsfolge wird mit einer fallenden Flanke angekündigt und der Lowpegel während 400us gehalten. Dann werden der Reihe nach die Rechteckpulse für jeden Kanal in aufsteigender Ordnung und positiver Logik in Zeitabständen von 400us moduliert und die Leitung nachher für die Dauer einer Periode von 20ms wieder hochgesetzt.

Die Pulsabstände sind nicht genau festgelegt, da soweit bekannt mit Flanken getriggert wird. Wir haben uns aber an die gemessenen Werte der Fernsteuerung (Pin 2) gehalten, was immer gut funktioniert hat. Auch die Frequenz der Pulsfolge kann mehr oder weniger beliebig variiert werden. Sie bestimmt nur, wie oft die Servos beim Empfänger nachgesteuert werden. Die gebräuchlichen 50Hz reichen für unsere Anwendung vollkommen aus, da das Modell keine hohen dynamischen Anforderungen erfüllen muss.

In Abbildung 6 und 7 werden die ersten beiden Pulse (Kanal 1 und 2) dargestellt. Abbildung 6 ist wiederum ein Ausschnitt von Abbildung 5 bezieht sich auf das Originalsignal. Abbildung 7 wurde am Ausgang des Microcontrollers gemessen und zeigt hervorragende Eigenschaften.

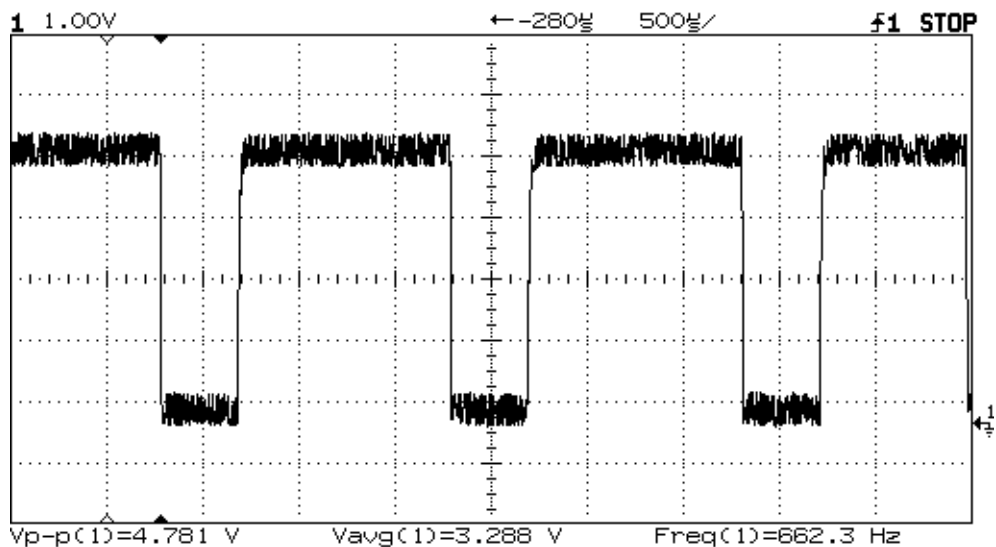


Abbildung 7

Ein Vergleich zwischen den echten Signalen der Fernsteuerung und den von der Interface-Schaltung künstlich erzeugten zeigt, dass die Rechtecke und Pulsabstände exakt gleich lang sind und sich nur geringfügig im Pegel unterscheiden, was aber für die digitale Signalverarbeitung keine Rolle spielt. Die Nachbildung unterscheidet sich aber darin, dass von jeder Pulsfolge nur die beiden ersten beiden Rechtecke (Kanäle) generiert werden. Im unteren Bild wird der Pegel also bis zur nächsten Periode hoch bleiben.

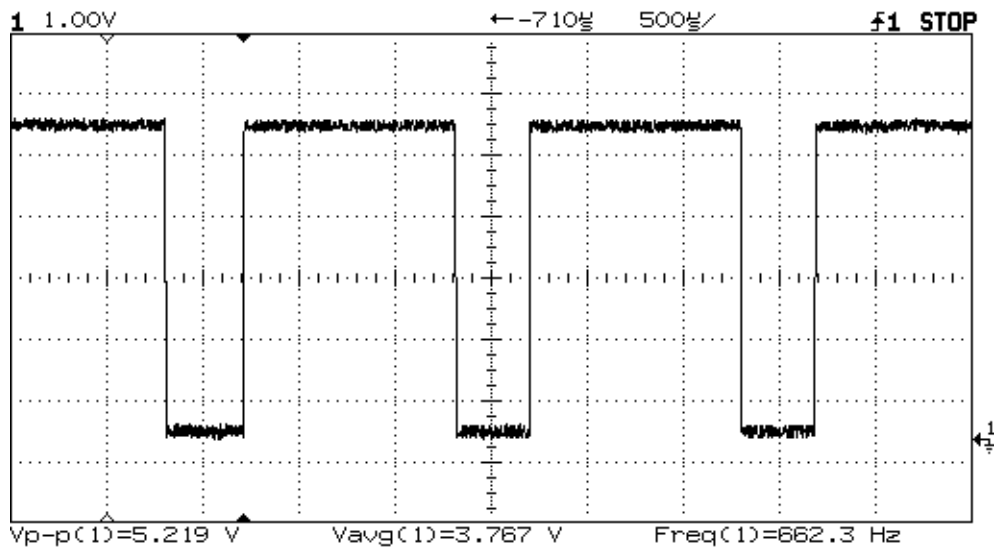


Abbildung 8

Zuerst wurde versucht, das Schüler-Pulstelegramm softwaremässig mit der seriellen Schnittstelle (COM) nachzubilden, indem die Strobe-Steuerleitung (RS232) mit einem Visual C++-Programm entsprechend der Pulsfolge ein- bzw. ausgeschaltet wurde, und zwar genau für eine bestimmte Dauer. Das funktionierte recht gut, solange der Computer nicht anderweitig beschäftigt war. Sobald aber die Rechenzeit knapp wurde und sich das Taskswitching vom Betriebssystem bemerkbar machte, wurden die Pulse nicht mehr genau. Diese Lösung war generell zeitkritisch, weil die Information in der Länge des Pulses steckte und diese Länge von einem Hochpräzisions-Timer kontrolliert wurde, der aber nur zur Verfügung stand, wenn der Prozess aktiv war. Der Prozess brauchte also kaum Rechenleistung, musste aber nach einer klar definierten Zeit wieder weiterlaufen – eine Forderung, die kaum an ein Multitaskingsystem gestellt werden kann. Hinzukommt, dass das Programm jeden Servo-Refresh im Abstand von 20ms selber generieren muss, also nicht nur die Kanaländerungen.

5.7 uP-Interface

Ein eleganterer Weg führt über eine Implementation in Hardware. Mit einem AT90S2313-Microcontroller in RISC-Technologie und einem RS232-Pegelwandler wurde eine Interface-Schaltung entworfen, die über die serielle Schnittstelle vom Programm die Steuerbewegungen in codierter Form (PCM) entgegennimmt und daraus die entsprechende Pulsfolge (PWM) formt. Das Interface dient also als unidirektionaler Konverter von der Modulationsart PCM in PWM. Das Pulstelegramm wird automatisch vom Controller wiederholt. Das Programm teilt diesem nur noch die neuen Kanalstellungen mit und kann sich dann wieder der Bildverarbeitung widmen. Ein grosser Vorteil bietet hier die asynchrone Kommunikation über die Schnittstelle, denn die Software muss die Datenübertragung grundsätzlich nicht kontrollieren und bekommt für die Übertragung auch eine hohe zeitliche Toleranz.

Eine Schaltung aus Timer- und Zählerbausteinen wurde deshalb verworfen, weil die Pulsfolge mit zwei modulierten Rechtecken schon recht kompliziert wird und gewisse Pulsabstände dazwischen gewünscht sind. Der Microcontroller übernimmt beinahe jede Aufgabe und nachträgliche Änderungen am Pulstelegramm können ohne jegliche Anpassung an der Beschaltung umprogrammiert werden, was das Ganze sehr skalierbar macht. Programmiert wurde er mit Assembler, wobei auch höhere Programmiersprachen wie C++ denkbar wären.

5.7.1 Aufbau und Funktion der Schaltung

Die Schaltung hängt an der gleichen Versorgungsspannung wie die Fernsteuerung (+9,6V) und regelt diese intern aber auf die für CMOS übliche Spannung von +5V hinunter. Die Pegelumwandlung zwischen der seriellen Schnittstelle (RS232) und der uP-Beschaltung (TTL) erledigt das DS232-IC, ein Verwandter des weit verbreiteten MAX232-Bausteins. Auch der eingebaute Spannungsregler LM7805 ist in solchen Schaltungen Standard. Der Microcontroller wird mit einem 4MHz-Quarz extern getaktet. Das einfache und übersichtliche Schaltschema ist im Anhang beigelegt.

Um die Interface-Elektronik vom starken Senderträgersignal abzuschirmen wurde die Kabelverbindung mit koaxialem Masseausenleiter versehen. Die Signalpegel von den Zuleitungen weisen natürlich trotzdem noch Schwingungen der Trägerfrequenz auf. Deren Amplituden sind jedoch recht klein und können vom Tiefpassfilter und dem nachgeschalteten Spannungsregler noch genügend reduziert werden. Die Diode schneidet deren negative Halbwellen raus.

Drei Leuchtdioden geben Auskunft über Zustände und Datenverkehr:

PB2/FE (gelb): Signalisiert einen Framing Error bei der seriellen Datenübertragung. Das bedeutet, dass der Controller beim letzten empfangenen Datenpaket kein gültiges Stopbit detektiert und das Paket deshalb verworfen hat. Da vom PC-Programm laufend Änderungen an den Fahrparametern übermittelt werden (19200 Baud) und Übertragungsfehler selten auftauchen, müsste die Anzeige im Fehlerfall sofort wieder von einer erfolgreichen Empfangsquittung gelöscht werden. Erlischt die LED nicht von selber wieder, sollte die Kabelverbindung zum PC und die Baudrate im Programm überprüft werden.

PB4/T0 (gelb): Ist jeweils für die Dauer einer generierten Pulsfolge aktiv. Dies geschieht jedes Mal, wenn der 8-Bit-Timer des Controllers einen Overflow Interrupt auslöst, was alle 20ms der Fall ist (Refresh der Servo). Die LED sollte also mit einer Frequenz von ungefähr 50Hz blinken.

PD0/RxD (rot): Zeigt die logischen Zustände der Empfangsdatenleitung vom Schnittstellentreiber (RS232→TTL) invertiert an, dort wo die empfangenen Pakete beim Controller eintreffen. In der Regel ist der logische Zustand der Datenleitung high und die LED leuchtet nur, wenn Bits durchgetaktet werden, sprich die Leitungspegel wechseln.

Mit dem Drucktaster am Rand der Platine kann ein Reset des Microcontrollers erzwungen werden. Kurzzeitige Unterbrüche der Betriebsspannung setzen den Chip ebenfalls zurück. Ein Reset bewirkt, dass die Servo auf Mittenstellung gebracht und solange dort gehalten werden, bis die ersten Steuersignale vom Computer kommen.

Auf der Platine werden nur die folgenden RS232-Leitungen zum Controller geführt:

RS232-Pin	Bezeichnung	Richtung (zu)	uP-Pin
3	RD Empfangsdaten	Controller	2
2	TD Sendedaten	Schnittstelle	3
5	CTS Sendebereitschaft	Schnittstelle	19
7	RTS Sendeanforderung	Controller	6
5	GND Schutz Erde	ohne	10

Ausser RD und natürlich GND wird im Moment keine andere Leitung auch wirklich benutzt, weder vom PC noch vom Chip. Anwendungen wären jedoch denkbar, wenn beispielsweise die Verbindung bidirektional erweitert werden soll. Der Microcontroller unterstützt externe Hardware Interrupts (Flankentrigger) auf Pin 6 (INT0), wo RTS schon angeschlossen ist. Damit könnte in der uP-Software eine Flusskontrolle implementiert werden.

5.7.2 Erklärungen zur seriellen Kommunikation (RS232)

Die Kanalinformationen werden von der Software auf dem PC über den Port COM1 seriell an die Interface-Schaltung gesendet. Es wird die asynchrone und unidirektionale Kommunikation benutzt. Die gesendeten Pakete werden nur einmal und ohne jegliche Flusskontrolle an die Schaltung geschickt. Fehler bei der Übertragung sind erfahrungsgemäss selten und haben keine gravierenden Auswirkungen. Eine Korrektur macht hier keinen Sinn, weil die Daten vom Programm häufig aktualisiert und alte Werte immer wieder durch neue ersetzt werden. Kritisch wäre nur der Fall, wenn das Auto beispielsweise lange Zeit einer Geraden folgen müsste und die Lenkung mit nur einem Befehl auf neutrale Stellung gebracht wird. Falls das betroffene Paket den Controller nicht erfolgreich erreichen würde, würde dieser die Servo nicht nachführen und das Modell würde seinen Kurvenkurs beibehalten.

Abbildung 8 zeigt die seriell übertragenen Pulse für das Datenwort 0xc040h (Kanal-Mittenstellung) - was binär dem Bitmuster 1100 0000 0100 0000b entspricht – nach der Pegelumwandlung in TTL-Logik. Die Bits werden beginnend beim niederwertigsten Bit #0 bis zum höchsten Bit #15 durchgetaktet, d.h. auf dem KO erscheint die Folge 0000 0010 0000 0011b über dem Zeitbereich von links nach rechts gelesen. Der lange Rechteck ganz links auf der Anzeige stammt grösstenteils noch vom vorherigen Paket, welche zur einfacheren Aufzeichnung wiederholt hintereinander gesendet wurden.

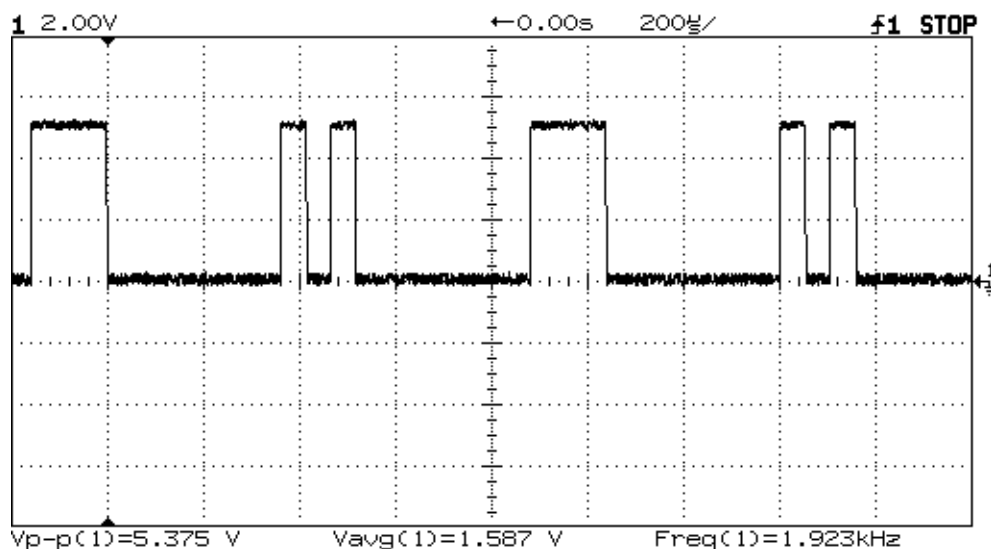


Abbildung 9

Ein einzelnes Bit hat hier die Dauer $1 / \text{Baudrate} = 1 / 19200 \text{ Bit/s} = 52,1\mu\text{s}$. Nach der fallenden Flanke des linken Rechtecks erscheint zuerst das Stopbit (Low-Pegel) und dann die 8 Datenbits gefolgt von einem Stopbit (High-Pegel). Das Paritätsbit wird nicht benötigt.

Die wichtigsten Parameter für die serielle Kommunikation zwischen Interface und PC lauten:

Port: COM1
Baudrate: 19200
Bytengröße: 8
Stopbit: 1
Parität: keine

Änderungen müssen sowohl in der Applikation sowie auch im Assemblerteil nachgeführt werden, damit sich beide UARTs von PC und Interface verständigen können.

5.7.3 Grundlegendes zum Microcontroller

Beim eingesetzten Microcontroller handelt es sich um den leistungsfähigen Baustein AT90S2313 der Serie AVR 8-Bit RISC von Atmel. Der Programmspeicher hat genug in-system-programmable Flash für eine leistungsfähige Applikation. Für uns war vor allem von Bedeutung, dass er Timer- und Interruptfunktionen unterstützt und das UART-Protokoll beherrscht. Der UART-Controller steuert die serielle Kommunikation über Status-, Kontroll- und Datenregister und ist auch in jedem PC eingebaut. Die serielle Schnittstelle unterstützt nur die asynchrone Datenübertragung, d.h. Taktsignale sind nicht vorhanden und es wird jeweils nur auf das Startbit synchronisiert, was eine Abtastung der Pegel erfordert. Die UART-Struktur im uP ermöglicht den Datenaustausch über die RxD- und TxD-Leitungen einer RS232-Schnittstelle durch Registerbefehle, ohne dass man sich um Abtasttheoreme auf dem untersten physikalischen Layer kümmern muss. Eine ausführliche Dokumentation zum Befehlssatz und der Architektur des Chips ist im PDF-Dokument des Herstellers enthalten.

Das Assemblerprogramm wurde mit der gratis erhältlichen Entwicklungsumgebung AVR Studio 4 von Atmel (<http://www.atmel.com>) geschrieben, welches in der Lage ist, die generierte .hex-Datei unmittelbar nach der Kompilation über die serielle Schnittstelle auf das Entwicklungssystem zu übertragen und den Flash-Speicher zu programmieren. Dafür stand das STK500 AVR® Flash MCU Starter Kit zur Verfügung. Mit diesem Board konnten auch gleich sämtliche Funktionen der Interface-Schaltung während der Programmlaufzeit simuliert und getestet werden, bevor die eigentliche Schaltung aufgebaut werden musste.

5.7.4 Programm des Microcontrollers

Der Maschinencode auf dem Microcontroller wurde so entworfen, dass er mehr oder weniger skalierbar ist. Änderungen der UART-Parameter sowie Taktfrequenzen, Perioden und Pulsabstände können grösstenteils an den Konstanten im Kopf des Assemblerprogramms vorgenommen werden. Der Precompiler berechnet daraus bei der Übersetzung die Registerwerte für das endgültige Programm. Polling-Verfahren konnten vermieden werden, stattdessen funktioniert praktisch die ganze Steuerung durch Interrupts. Der interne Datenaustausch läuft ausschliesslich mit Registeroperationen. Für einen sinnvollen Gebrauch des Stacks ist die Datenmenge nicht ausreichend. Nur das Statusregister wird falls nötig bei Interrupt-Aufrufen auf dem Stack gesichert. Die Routinen sind so aufeinander abgestimmt, dass sie sich nicht unerwünscht unterbrechen.

Die Assemblerbefehle sind in der .asm-Datei (Source) ausreichend dokumentiert und sollen an dieser Stelle nicht näher erläutert werden. Grob lässt sich das Programm in folgende Blöcke aufteilen, die eigentlich alle als Interruptroutinen eingesetzt werden:

RESET: Diese Interruptroutine initialisiert die Register mit Status- und Kontrolldaten sowie Startwerten bei jedem Reset des Controllers. Ein Reset wird durch den Tas-

ter auf der Schaltung oder beim Einschaltvorgang nach dem Anlegen der Speisepannung am uP verursacht. Während der Initialisierung werden auch die Servos in eine neutrale Lage gebracht und dort gehalten, solange die Schnittstelle noch nichts sendet.

- TIM_OVF0:** Overflow Interruptroutine des 8-Bit-Timers (Timer 0), die alle 20ms ausgelöst wird und die Generierung einer Pulsfolge bewirkt. Dazu startet er wiederum einen präzisen 16-Bit-Timer, der das Pulstelegramm nachbildet und danach die Kontrolle wieder zurückgibt. Dieser Programmabschnitt ist zeitkritisch und darf nicht unterbrochen werden, damit die Pulse nicht verzerrt werden. Die Routine startet den 8-Bit-Timer vor dem Ablauf automatisch wieder neu.
- TIM_COMP1:** Compare Match Interruptroutine des 16-Bit-Timers (Timer 1) wird bei jeder Flankenänderung aufgerufen und generiert die Rechteckpulse, indem sie den Ausgang toggelt, den Vergleichswert des Präzisionszählers auf die neue Pulslänge setzt und den 16-Bit-Timer erneut startet, solange noch Pulse anliegen.
- UART_RXC:** Receive Interruptroutine wird angesprungen, sobald der UART ein Byte empfangen hat und im Empfangsregister bereitgestellt hat. Dieser Interrupt kann jederzeit aufgerufen werden, weil die Daten vom PC nicht periodisch eintreffen. Timeroutinen haben Priorität, sodass diese Routine allenfalls warten muss, bis die Pulsfolge abgearbeitet ist.

Am Dateianfang sind die Konstanten definiert, die gegebenenfalls ändern können. Zahlen, die den Wertebereich eines Bytes überschreiten, können unter Umständen Probleme bei den Precompiler-Berechnungen machen, deshalb muss hier sehr vorsichtig vorgegangen werden. Vermutlich verarbeiten auch nicht alle Compiler diesen Codeteil gleich.

Das Zustandsdiagramm des Programmablaufs mit den relevanten Zuständen und Aktionen ist in Abbildung 9 ersichtlich. Zu beachten ist der Status des Ausgangspins, wo die Pulse erscheinen und die Interrupt-Steuerung. Durch einen Reset können die Zustände jederzeit vorzeitig verlassen werden.

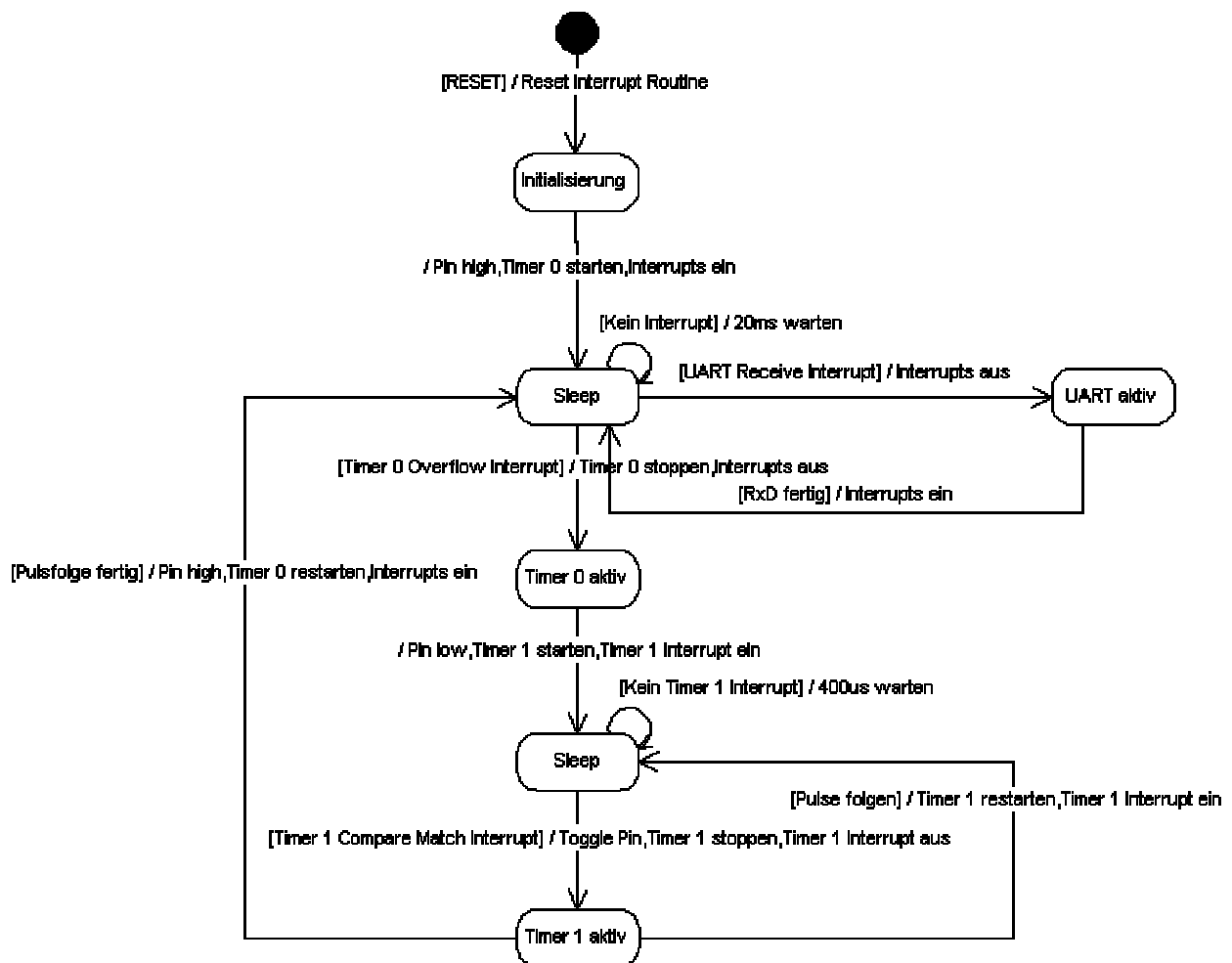


Abbildung 10

5.7.5 Inbetriebnahme

Dieser Abschnitt widmet sich nur der Inbetriebnahme der grundlegenden Systeme Fernbedienung und Interface. Da die Implementation des neuen Matlab-Codes in Visual C++ nicht fertig geworden ist, können noch keine Kalibrierungsvorgänge (z.B. für Kamera) im Programm beschrieben werden. Im Moment wird nur gerade die Schnittstelle initialisiert.

Folgendes Vorgehen wird empfohlen, ist aber nicht unbedingt in genau dieser Reihenfolge notwendig:

1. Fernbedienung mit der Interface-Schaltung verbinden.
2. Interface-Schaltung mit der seriellen Schnittstelle (COM1) verbinden. Das Steuerprogramm noch nicht starten.
3. Kippschalter oben links auf dem Senderpult auf die untere Position Trainer stellen.
4. Fernbedienung einschalten (Power-Schieber nach oben). In der Schaltung blinkt die gelbe LED PB4/T0 und PB2/FE leuchtet in der Regel, wenn noch keine gültigen Daten empfangen wurden.

5. Modellauto einschalten.
Die Räder sollten sich automatisch in die Mittenstellung bewegen und der Motor ruhig sein. Bewegen sich die Antriebsräder ein wenig oder befinden sich die Räder nicht genau in der Mitte, muss entweder in der Software des Controllers oder im Computerprogramm ein Offset eingebaut werden. Die Potis für die Verschiebung der Mittenstellung auf der Fernbedienung sind im Schülerbetrieb leider wirkungslos.
6. Das Computerprogramm für die Steuerung kann jetzt aufgerufen werden und die Kontrolle des Autos übernehmen.

Die Interface-Schaltung sollte die Fernbedienung auch ohne Verbindung zum Computer mit den letzten gültigen Signalen „füttern“.

5.7.6 Ablauf einer Datenübertragung

Die beiden Kanäle für die Lenkung und die Drehzahl werden vom Programm binär kodiert und auf 2 Datenbytes (Word) verteilt. Ein solches Byte ist nach einem vorgegebenen Muster aufgebaut, welches die Software auf dem Microcontroller verarbeiten kann. Wir haben folgende Lösung gewählt und implementiert:

- Bit# 0-6: Stellt den Kanalwert aufgelöst in Schritten von 0-127 dar. Der Wert 0 stellt bei der Lenkung (Kanal 1) den maximalen Ausschlag nach links dar, für die Drehzahl (Kanal 2) bedeutet er volle Rückwärtsfahrt. Die Neutralstellung liegt in der Mitte bei 64.
- Bit# 7: Hat den Wert 1, wenn das Byte dem Kanal 2 zugeordnet wird, andernfalls den Wert 0. Für den Microcontroller ist diese Unterscheidung wichtig und muss genauso eingehalten werden, sonst werden die Kanäle vertauscht.

Eine Funktion in der Visual C++-Applikation, die bei jeder Änderung von Geschwindigkeit und Richtung des Modells aufgerufen wird, schreibt ein Datenwort mit den geänderten Kanalwerten an die serielle Schnittstelle (COM1). Das Highbyte bildet dabei Kanal 2, das Lowbyte gehört Kanal 1.

Da bei der seriellen Datenübertragung die Paketbits in umgekehrter Reihenfolge durchgetaktet werden (Bit# 0,1,2,...,15), kommt zuerst das LSB (Kanal 1) gefolgt vom MSB (Kanal 2) ans Interface durch.

Der UART im Microcontroller beginnt bei der Detektion eines Startbits (fallende Flanke) mit der Pegelabtastung und löst nach dem Empfang des Stopbits einen Interrupt aus. Die dafür zuständige Interrupt-Routine liest das empfangene Byte im Register, verarbeitet es und ordnet den Wert seinem Kanal zu. In die Verarbeitung fällt das Maskieren der Kanalbits #0-7 (eigentliche Information), die Invertierung des Kanals 2 für die Drehzahl (üblich bei Futaba-Anlagen) sowie die Umrechnung der Codes (PCM) in einen Timervergleichswert für die Bildung der Rechteckimpulse.

Ein hochpräziser 16-Bit-Timer formt nun aus den berechneten Pulslängen die Pulsfolge für die Fernsteuerung (PWM). Bei jedem Compare Interrupt ergibt sich eine Pegeländerung des Pins am Controller, der mit dem Anschluss IN der L-/S-Buchse verbunden ist. Es werden nur 2 Rechteckpulse à durchschnittlich 2200us + 3 Pulsabstände à 400us = 2600us periodisch erzeugt, weil die restlichen 5 Kanäle nicht belegt sind und deshalb ignoriert werden können.

5.7.7 Visual C++-Beispielcode

An dieser Stelle soll der wichtigste Codeabschnitt des Testprogramms PPModulator.exe kurz beschrieben werden. Mit dieser Applikation kann man durch Schieberegler in einer Dialogbox über das Interface Steuersignale an das Modellauto durchgeben. Das Programm wurde nur zum Austesten der Steuerung aus der Software heraus entwickelt, ist also nicht ohne Mängel und ist im Projekt auch nicht gross dokumentiert. Die MSDN Library im Visual Studio beschreibt alle benutzten Befehle und Strukturen recht ausführlich.

Etwas problematisch ist vielleicht der Umstand, dass ab Windows 2000 keine direkten Portzugriffe mehr möglich sind. Anwendungsprogramme laufen nämlich im User Mode und dürfen dort nur bestimmte Speicherbereiche und Befehle nutzen, die das Betriebssystem zulässt. Sie übergeben ihre Daten normalerweise einem Treiber, der dazu berechtigt ist. Es gibt verschiedene Lösungen dafür, wie man trotzdem für seine Applikation diesen Adressbereich freischalten kann. Sofern man aber nicht selber Zugriff auf Status-, Steuer- und Datenregister des UART braucht um gezielt einzelne Leitungen der RS232-Verbindung an- oder auszuschalten, sondern nur gewöhnliche serielle Datenübermittlung über Rx/Tx wünscht, kann man die API von Visual C++ nutzen. Diese tauscht über herkömmliche Dateischreib- und Lesebefehle Daten mit dem Schnittstellentreiber aus. Das funktioniert auf allen Windowssystemen, weil der installierte Treiber die hardwarenahe Umsetzung übernimmt.

Die Kommunikation über die serielle Schnittstelle verläuft in drei Schritten:

1. Initialisierung der Schnittstelle:

```
HANDLE hCom;
DCB dcb;
DWORD dwError;
BOOL fSuccess;

hCom = CreateFile( "COM1",          // pointer to name of the port
                  GENERIC_READ | GENERIC_WRITE, // access mode
                  0, // comm devices must be w/exclusive-access
                  NULL, // no security attrs
                  OPEN_EXISTING, // comm devices use OPEN_EXISTING
                  0, // not overlapped I/O
                  NULL); // hTemplate must be NULL for comm devices

if (hCom == INVALID_HANDLE_VALUE)
{
    /* handle error */
    dwError = GetLastError();
}

/* Omit the call to SetupComm to use the default queue sizes.
   Get the current configuration. */
fSuccess = GetCommState(hCom, &dcb);
if (!fSuccess)
{
    /* handle error */
    MessageBox("GetCommState failed!", "Port Error", MB_ICONERROR);
}
```

```
/* Fill in the DCB: baud=19200, 8 data bits, no parity, 1 stop bit. */
dcb.BaudRate = CBR_19200;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
dcb.fDtrControl = DTR_CONTROL_ENABLE;          // enable DTR line
dcb.fRtsControl = RTS_CONTROL_TOGGLE;         // toggle the RTS line
dcb.fOutxCtsFlow = FALSE; // no CTS output flow control
dcb.fOutxDsrFlow = FALSE; // no DSR output flow control
dcb.fOutX = FALSE; // no XON/XOFF out flow control

fSuccess = SetCommState(hCom, &dcb);
if (!fSuccess)
{
    /* handle error */
    MessageBox("SetCommState failed!", "Port Error", MB_ICONERROR);
}
```

Bei der Methode `CreateFile` sind die Bedingungen zu beachten, die für einen seriellen Port als Handle gelten. Die Fehlerbehandlung wird hier nicht sonderlich genau genommen und strenggenommen müsste man noch Timeouts einführen, damit das Programm nicht hängen bleibt, wenn die Verbindung nicht geöffnet werden kann.

Die Struktur DCB enthält die ganzen Kontrolleinstellungen für den UART. Alle Parameter werden hier gesetzt. Ursprünglich war mal noch die Idee, auf Seite des Interfaces für mehr Sicherheit beim Empfang der Pakete zu sorgen, indem die RTS-Leitung jede Datenübertragung signalisiert (`dcb.fRtsControl = RTS_CONTROL_TOGGLE`). So könnte das Interface damit die Authentizität der eintreffenden Daten überprüfen, bevor daraus fehlerhafte Pulse entstehen. Diese Einstellung blieb jedoch aus unbekanntem Gründen wirkungslos. Es gab jedoch ohnehin nie solche Störungen, sodass die Kommunikation mit den eruierten Parametern auch so tadellos funktioniert hat und lieber nicht geändert werden sollten.

2. Schreiben der Daten:

```
DWORD dwData; // Datenwort
DWORD dwBytesWritten;

/* Schiebereglerstellungen werden im Datenwort codiert */
dwData = ( (UINT)m_sl_ch1.GetPos() << 8) // MSB = Kanal 1
         + (UINT)m_sl_ch2.GetPos() + 0x80; // LSB = Kanal 2

/* Sende die Daten */
WriteFile( hCom, // handle to output port
           &dwData, // pointer to input buffer
           2, // number of bytes to write
           &dwBytesWritten, // number of bytes written
           NULL ); // pointer to structure for overlapped I/O
```

Die Funktion `WriteFile` schreibt die Daten verpackt in einem Wort (16 Bit) an die Schnittstelle. Hier könnte man anschliessend mit der Variablen `dwBytesWritten` noch nachprüfen, ob auch wirklich die gewünschte Anzahl Bytes gesendet werden konnte. Andernfalls müsste das Paket nochmals übertragen werden.

Es macht wegen der Dynamik des Programms wahrscheinlich keinen Sinn, hier lange zu verweilen. Eine elegante Erweiterung wäre vielleicht eine bidirektionale Kommunikation mit dem Interface: Der Controller bestätigt jedes erhaltene Wort, indem er es unverändert zurück an die Schnittstelle schickt. Das Programm liest dieses Paket mit der Funktion `ReadFile` ein und vergleicht es mit dem gesendeten. Bei Ungleichheit wird die Übertragung einmal wiederholt, sofern die Daten in der Zwischenzeit nicht schon wieder aktualisiert worden sind. Treten mehrmals derartige Fehler auf, sollte das dem Benutzer mitgeteilt werden.

3. Schliessen der Datei:

```
CloseHandle(hCom); // handle to object to close
```

Dieser Befehl darf nicht fehlen, weil sonst die Verbindung zur seriellen Schnittstelle für andere Teilnehmer gesperrt bleibt.

5.8 Modifikationen am Gesamtsystem

Die Gerätschaften, die am Regelkreis zur Steuerung des Modellautos beteiligt sind, haben durch den Umbau etwas geändert, wobei neue Teile hinzugekommen sind und einige nicht mehr gebraucht werden.

Die Modifikationen lassen sich aufteilen in die Gruppen:

Modellauto

Das ferngesteuerte Modellauto wurde in seinem Innenleben erheblich umgebaut. Änderungen wurden an folgenden Komponenten vorgenommen:

Empfänger: Das RF-Modul (27MHz-AM) wurde durch den neuen FM-Empfänger (40MHz) ausgetauscht. Die Kanalbelegung wurde so gewählt, dass die Lenkung auf Kanal 1 und der Fahrtregler auf Kanal 2 hört.

Spannungskonverter: Die Schaltung für die Bereitstellung einer geeigneten Versorgungsspannung für die Kamera wurde wegen erheblichen HF-Störungen des DC/DC-Konverters ausgebaut.

Akkupack: Anstelle des Wandlers wird die Kamera aus einem 8x1,2V-Akkupack gespeisen, der mit Klettband an der Innenseite des Autodachs befestigt ist. Der neue Schalter für den Kamerabetrieb wurde gut zugänglich auf dem Autodach angebracht.

Fernsteuerung

Die alte AM-Fernbedienung wurde durch den neuen FM-Sender Futaba Skysport 6 ausgetauscht. An der neuen Fernsteuerung wurden auch noch folgendes verbessert:

Trainer-Schalter: Der eingebaute monostabile Kippschalter für den Lehrer-/Schülerbetrieb wurde mit dem bistabilen Schalter von Kanal 5 (Gear) ausgetauscht, damit im Betrieb mit der Interface-Schaltung nicht ständig der Schalter gedrückt werden muss. Die untere Schalterstellung mit der Bezeichnung Trainer auf dem Sendepult bedeutet die Übergabe der Kontrolle an den PC. Das Modellauto lässt sich im manuellen Betrieb mit dem rechten Steuerknüppel fahren.

Akkugehäuse: Um jederzeit eine sichere Senderausgangsleistung garantieren zu können, wurde anstatt der Akkus ein geregeltes Netzgerät benutzt. Am Stecker für den Anschluss der Versorgungsspannung auf der Platine kann beides angeschlossen werden, bei speziellen Steckverbindungen ist aber unbedingt auf die richtige Polarität zu achten.

Interface

Die Interface-Schaltung ist neu hinzugekommen und verbindet Fernbedienung und Laptop bzw. PC. Sie braucht grundsätzlich keine externe Stromversorgung, da sie die Spannung des Senders abnimmt.

Laptop

Am Laptop, der für dieses Projekt verwendet wurde, gibt es nur zwei Modifikationen:

A/D-Wandler (PCMCIA): Entfällt mit der neu eingeführten digitalen Steuerung und wird vom Programm nicht mehr benutzt.

Serielle Schnittstelle: Ersetzt den A/D-Wandler durch das serielle Verbindungskabel zum Interface.

6 Visual C++-Implementation

6.1 Einleitung

Das Visual C++-Programm der Vorgängergruppe (FROG) musste aufgrund völlig neuer Vorgaben fallengelassen werden. Mit der Einführung eines digitalen Interface entfiel die Ansteuerung des A/D-Wandlers und der Bildverarbeitungsalgorithmus musste als wesentlicher Bestandteil der Semesterarbeit sowieso völlig neu überarbeitet werden. Wegen zahlreichen unerwarteten Störungen bei der Hardware konnte jedoch die im Matlab gefundene Lösung nicht mehr im Visual C++ implementiert werden, die Zeit reichte einfach nicht mehr. Es wurde zwar ein Visual C++-Projekt angefangen, doch ist das Programm noch nicht voll funktionsfähig und hängt quasi in einem undefinierten Zustand.

6.2 Neues Projekt

Zahlreiche Änderungen am Visual C++-Teil führten schliesslich zu einem neuen Projekt mit dem Namen SmartFrog. Die GUI von FROG mit der dialogbasierten Oberfläche wurde durch eine SDI-Anwendung mit Menüunterstützung ersetzt.

Ein wesentliches Problem war schon immer das Einlesen der Bilddaten vom Grabber in das Programm. Video für Windows stellt zwar diverse Schnittstellen und Funktionen für die Steuerung und Verarbeitung von Kameraaufnahmen zur Verfügung, aber diese beschränken sich mehrheitlich auf die GUI-Anbindung oder das Erstellen von Videodateien aus Streams (z.B. .avi-Dateien). Es war jedoch nicht klar, wie eine Visual C++-Applikation direkt auf diese Bilddaten zugreifen konnte, und zwar bis auf Pixelebene. Dateizugriffe würde zu lange dauern, ein Buffer wurde gebraucht.

6.3 Video Capture

Die Methode, die dann in FROG das Problem vorübergehend gelöst hat, war ein Umweg über die Zwischenablage. Sie funktioniert im Wesentlichen so, dass über die Funktion capGrabFrame ein Frame vom Grabber in die Zwischenablage kopiert wird, wo es das Programm in einem unkomprimierten Bitmapformat einlesen und in einem Feld von unsigned char speichern kann. Der entsprechende Code dazu lautet:

```
::OpenClipboard(hWndC); // öffnet und reserviert das Clipboard
EmptyClipboard(); // leert das Clipboard und gibt Benutzungs-
recht
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
{
// holt Daten von einem bestimmten Format aus dem Clipboard
HBITMAP handle = (HBITMAP)GetClipboardData(CF_BITMAP);
// erzeugt einen (echten) Zeiger auf das Objekt
CBitmap * p_bitmap = CBitmap::FromHandle(handle);
```

```

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

```

Das Problem tauchte aber gleich beim erstmaligen Ausführen der FROG-Anwendung auf einem projektfremden Laptop auf: Dort waren in der Windowsumgebung vom Grafiktreiber entweder 16-Bit oder 32-Bit Farbtiefe vorgegeben. Die Kamerabilder erschienen mit der besseren Auflösung von 32-Bit total verrauscht und unbrauchbar. Der Fehler steckte im Programm, wo fest mit 3 Bytes pro Pixel für den Rot-, Grün und Blauanteil (24-Bit RGB) gearbeitet wurde. Im Betrieb mit 32-Bit jedoch stellt Windows jedes Pixel noch mit einem zusätzlichen Byte - dem sogenannten Alpha - dar. Das Ganze war nur deshalb ein Problem, weil der Inhalt der Zwischenablage in der Auflösung der Grafikkarte geliefert wird, im Programm aber von festen 24-Bit in Graustufe umgerechnet wird.

```

// *** Farbbild wird in Graustufenbild umgewandelt ***
for(i = 0; i<width*height; i++)
{
    // Grauwert = (Rotwert+Grünwert+Blauwert)/3
    GrayMap[i]=(RGBMap[3*i]+RGBMap[3*i+1]+RGBMap[3*i+2])/3;
}
// *** This works only if the desktop is set to 24 bit pixel color

```

Wir brauchten die Rohdaten vom Grabber oder zumindest ein Format, dass sich mit jeder Bildeinstellung verträgt. Natürlich könnte man diese Einstellung jedesmal im Programm abfragen und entsprechend berücksichtigen, nur wird es recht kompliziert, wenn z.B. mit 16-Bit Farbtiefe gearbeitet wird. Viel einfacher, unabhängiger und sicher auch schneller wäre ein Zeiger auf ein Feld, wo der Grabbertreiber die RGB-Daten reinschreibt und das Programm benachrichtigt.

Nach einigem Suchen wurde das Makro `capSetCallbackOnFrame` gefunden, die eine Callback-Funktion in der Applikation setzt. `AVICap` ruft diese Funktion im Anschluss an `capGrabFrame` auf und liefert als Übergabeparameter eine Struktur mit den Bilddaten. Mit `capSetUserData` wird noch ein Zeiger auf die aktuelle Klasse mitgegeben, damit die Callback-Funktion auch auf deren Membervariablen und -funktionen zugreifen kann.

```

HWND hWndC = capCreateCaptureWindow( "Capture Window",
                                     WS_CHILD|WS_VISIBLE|WS_BORDER,
                                     0,0,
                                     320,240,
                                     m_hWnd,0 );
capSetUserData(hWndC, this);
capSetCallbackOnFrame(hWndC, FrameCallbackProc);

LRESULT CALLBACK FrameCallbackProc(HWND hWnd, LPVIDEOHDR lpVHdr)

```

Der Parameter `lpVHdr` ist eine `VIDEOHDR` structure mit der Membervariablen `lpData`, welche ein Zeiger auf ein Feld von Bytes ist. Das ist genau der Eingangsbuffer, den wir im Programm einlesen. In der Funktion `FrameCallbackProc` werden diese Bytes – jeweils 3 pro Pixel hintereinander – linear in Graustufe umgerechnet. Vor dem Return übergibt sie das Feld mit den Graustufenwerten an eine Memberfunktion des Hauptprogramms, wo die weitere Bildverarbeitung stattfindet.

Ein Debug dieses Programmteils machte klar, dass dieser Ablauf war klappt, aber die zurückgegebenen Graustufenbilder unbrauchbar sind. Bei Abgabe des Berichts war noch nicht klar, ob der Fehler bei der Wiedergabe des Bildes im Applikationsfenster liegt oder auf einen fehlerhaften Bufferzugriff zurückzuführen ist. Jegliche Bildkompression wurde zudem in der Dialogbox, die mit dem Befehl `capDlgVideoCompression(hWndC)` erscheint, ausgeschaltet und nachkontrolliert. Man sollte vielleicht mal den Bufferinhalt detailliert untersuchen, nachdem ein einfaches Bild mit markanten Merkmalen geschossen wurde, und nach Zusammenhängen suchen.



Abbildung 1

Abbildung 1 zeigt die Oberfläche von SmartFrog, wie sie momentan existiert. Der linke Fensterausschnitt ist für das Capture Window der Kamera reserviert. Der Befehl `capGrabFrame` zeigt dort die Bilder vom Grabber autonom an. Auf der rechten Seite wird das von der Callback-Funktion berechnete Graustufenbild dargestellt. Es macht wirklich den Eindruck, dass der Buffer falsch gelesen wird, denn die Kanten können jeweils mit der Aufnahme in Verbindung gebracht werden, abgesehen davon, dass sie scheinbar an der horizontalen Achse gespiegelt sind.

6.4 Steuerung über Interface

Der Sender für das Modellauto wird über die Interface-Schaltung vom Programm ferngesteuert. Auf die Kommunikation mit dem Interface über die serielle Schnittstelle soll hier nicht mehr eingegangen werden, da in der Hardware-Beschreibung schon ein Beispiel vorgestellt wird. Es wird empfohlen, dieses Beispiel im Projekt als Memberfunktion aufzunehmen. Diese Funktion kann dann nach der Bildverarbeitung die berechnete Richtung und Geschwindigkeit des Modells an die Hardware weitergeben. Im Moment wird in SmartFrog nur gerade im Initialisierungsteil mit dem Interface „gesprochen“.

6.5 Matrix-Klassen in C++

Matlab arbeitet hauptsächlich mit Matrizen und Vektoren, während C++ nur Zeiger und Basistypen kennt. Weil im Matlab-Programm einige mehrdimensionale Matrizen und Spezialoperatoren benutzt werden, wurde nach C++-Klassen gesucht, die Matrizen auf objektorientierter Basis umsetzen. Zusammen mit überladenen Operatoren wäre der C++-Code sehr übersichtlich und fast so gut lesbar wie in Matlab. Es gibt relativ viele Anwendungen in diesem Gebiet und jede Klasse bietet andere Möglichkeiten. Optimal wäre eine einzige Klasse im Projekt, die alle benötigten Matrizenmanipulationen beherrscht, doch viele sind so umfangreich, dass das Projekt schlussendlich mehr Headerfiles für die Einbindung dieser Klassen enthält als eigentliche Projektdateien.

Dabei werden bei weitem nicht alle möglichen Matrizenoperationen ausgeschöpft. Im Wesentlichen braucht es nur eine Implementation der folgenden Befehle und Strukturen:

- Grundarithmetik wie Addition, Subtraktion, etc. mit überladenen Operatoren
- Zuweisungs- und Kopieroperatoren
- Submatrizen, Vektoren
- Lineare Faltung einer Matrix mit einer Submatrix (Filter)
- (Quadrat)Wurzel
- Potenz
- Minimum, Maximum einer Matrix
- Betragsbildung
- Standardabweichung
- Solver für lineare Gleichungssysteme

Letzteres ist am anspruchvollsten, fehlt aber meist in keiner leistungsfähigen Matrix-Klasse. Das lineare Gleichungssystem reicht deshalb, weil die Fahrspur mit einer Geraden angenähert werden soll, was für den Regler einfacher ist und kostbare Rechenzeit spart. Der Datentyp der Matrizen ist überall `double` oder sonst ein ausreichend genauer Fließkommazahltyp. Zu beachten ist auch, dass in C++ die Indexierung von Arrays bei Null beginnt, während Matlab ab Eins nummeriert.

Viele Matrix-Klassen sind im Internet frei erhältlich. Auf der Semesterarbeit-CD sind einige davon abgelegt. Meist handelt es sich um `.h`-Dateien mit den Definitionen der Klassen und `.c`-Dateien mit deren Deklarationen. Die Matrix TPL `matrix11` schien besonders interessant und brauchbar. Die auf der CD verfügbare Professional Version beherrscht einen Grossteil der oben aufgelisteten Funktionen, ist objektorientiert programmiert und gut dokumentiert. Die Matrix-Klassen sind Schablonen und können mit einem beliebigen Datentyp arbeiten. Für die Einbindung der Klassen ins Projekt ist das dazugehörige Beispielprogramm hilfreich.

Jene Befehle, die in der eingesetzten Klasse nicht implementiert wurden, müssen halt nachträglich durch eigenen Code ergänzt werden. Die Algorithmen sind allgemein bekannt und gebräuchliche Lösungen können entweder von anderen Matrix-Klassen (z.B. `matrix2d`) abgeschaut oder im Internet recherchiert werden. Zuletzt wurde im Projekt von SmartFrog jedoch der MS Visual Development Kit eingebunden und auf weitere Matrix-Klassen verzichtet.

6.6 MS Visual Development Kit

Weil die Darstellung und Verarbeitung von Bitmaps mit der MFC-Umgebung für Laien recht kompliziert ist und viel Overhead an Programmcode benötigt, wurde der MS Visual Development Kit (VDK) heruntergeladen und installiert. Die Installation muss übrigens auf jedem Arbeitsplatz zuerst durchgeführt werden, bevor diese Klassen im Projekt auch vorhanden sind. Zur unterstützten Bildverarbeitung gehören auch Matrizenberechnungen, die für die Umsetzung des Matlab-Algorithmus benötigt werden. Es handelt sich hier also um eine all-in-one-Lösung. Die Membervariablen und -funktionen sind in der VSDK Help ausführlich beschrieben und brauchen nicht vorgestellt zu werden. Ein Graustufenbitmap, das zuvor aus einem Feld von Bytes `pGrayMap` entstanden ist, wird mit einigen wenigen Codezeilen auf den Bildschirm gebracht.

```
CVisGrayByteImage temp(320,240,1,evisimoptDefault,pGrayMap);
CVisPane pane( 320,
               240,
               "Display Window",
               evispaneVisible,
               FromHandle(m_hWnd),
               false );
pane.Display( evisnormalizeCopyBytesSameType,
             temp,
             evispanedispDefault,
             false,
             true);
```

Auch das MS Visual Development Kit befriedigt nicht alle Wünsche, da er sich mehr auf die GUI und konventionelle Bildverarbeitung konzentriert als auf streng mathematische Matrizenberechnungen. Ein Grossteil der benötigten Befehle sind ohnehin einfach zu implementieren.

6.7 Geplanter Programmablauf

Das Programm in Visual C++ konnte zwar nicht fertiggestellt werden, aber eine Vorstellung vom Ablauf und den beteiligten Prozessen ist vorhanden. Der Prozess vom Schiessen eines Bildes bis zur Nachsteuerung des Modellautos ist mit einem Sequenzdiagramm in Abbildung 2 nachvollzogen, wobei nur die wichtigsten Objekte bzw. Klassen und Aktionen enthalten sind. Die Initialisierung der seriellen Schnittstelle und das Eröffnen eines Capture Window werden hier weggelassen.

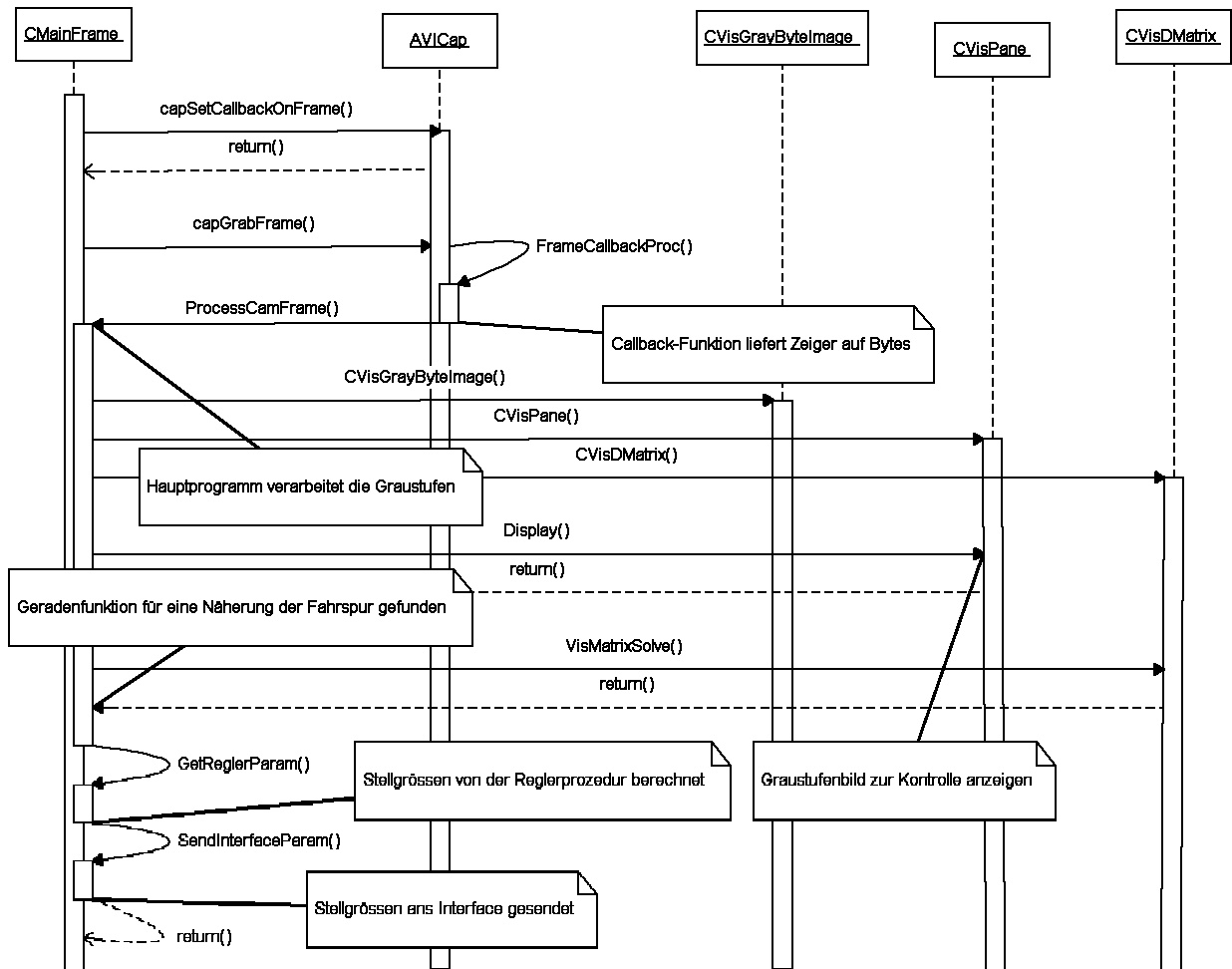


Abbildung 2

Die letzte Funktion muss `capGrabFrame` erneut aufrufen, um den Prozess wieder zu starten. Diese Endlos-Schleife soll solange laufen, bis das Programm entweder gravierende Fehler in der Verarbeitung entdeckt oder die Fahrspur nicht mehr eindeutig detektiert werden kann. Um dem Programm noch maximale Rechenleistung zur Verfügung zu stellen, ist es vielleicht noch sinnvoll, dass es seinen eigenen Thread beim Betriebssystem mit höchster Priorität deklariert.

7 Persönliches Fazit

7.1 Ausgangslage

Wie eingangs erwähnt, wäre es unsere Aufgabe gewesen die Bildverarbeitung robuster gegen störende Einflüsse zu gestalten. Wir waren davon ausgegangen, dass die Hardware funktions-tüchtig sein würde. Als wir jedoch den Versuchsaufbau gemäss unserer Vorgängergruppe testen wollten, wurde festgestellt, dass das Fahrzeug massiv ruckelte und an einen lauffähigen Prozess nicht zu denken war. Zuerst galt es also, die Ursachen dieses Ruckelns zu ergründen und zu beheben. Es kristallisierte sich heraus, dass das Auffinden dieser Störungen den grössten Teil unserer Zeit in Anspruch genommen hatte, da sie sich als besonders hartnäckig herausstellten. Deswegen blieb dann nicht mehr allzu viel Zeit übrig, um uns unserem eigentlichen Auftrag zu widmen.

7.2 Was erstellt wurde

Die Störungen konnten eliminiert werden, sodass das Fahrzeug nun in absolut ruhigem Zustand verharrt. Eine Übersetzung des Matlab-Programms in ein Visual-C++ Projekt blieb aber erfolglos. Es wurde versucht einen Matlab/C++ Compiler von MathWorks dafür einzusetzen, dieser erzeugte aber so viel unübersichtlichen Schnittstellencode, dass auch diese, zunächst elegant erscheinende Möglichkeit verworfen werden musste. Das Compilieren einer mathematischen C-Klasse, welche die für unser Problem benötigten Algorithmen (u. A. Matrizenoperationen) enthalten hätte, scheiterte an den überaus zahlreichen Reklamationen des Compilers.

7.3 Erkenntnisse

Wir wählten die Fachrichtung digitale Signalverarbeitung für unsere erste Studienarbeit. Dieses Fach war uns vorher unbekannt und baut auf den Grundlagen der Nachrichtentechnik aus dem zweiten Studienjahr auf. Der Bereich Bildverarbeitung steht nicht im Lehrplan. Eine solche Aufgabe stellt eine spannende Herausforderung dar, lernt man doch auch die mathematischen Beziehungen von der praktischen Seite her kennen, welche einem in der Theorie schon nahegelegt worden sind. Durch die Störungen des Autos, verliessen wir dann leider bald wieder den Bereich der digitalen Signalverarbeitung, um uns vielmehr mit Computertechnik und Informatik auseinander zu setzen. Dies sollte eigentlich nicht der Sinn einer DS-Arbeit sein. Dieser Sachverhalt zeigt aber auch auf, dass man eben auch vor negativen Überraschungen nicht gefeit ist. Ein scheinbar kleines Problem kann sich auf ein Problemfeld ausweiten, welches dann viel Zeit in Anspruch nimmt.

7.4 Dankeschön

Zu guter Letzt möchten wir uns für die freundschaftliche Zusammenarbeit mit Herrn Schuster bedanken. Er diskutierte mit uns wichtige Sachverhalte um das Thema der Bildverarbeitung. Auch war er sich nicht zu schade, die sturen Störungen des Fahrzeuges einmal selber mit Alufolie zu bekämpfen. Herzlichen Dank.