

Automatische Autofernsteuerung mittels 802.11b und Bildverarbeitung auf einem PC

Semesterarbeit SS03

R. Guldener T. Balczarczyk

Hochschule Rapperswil, 11. Juli 2003

Inhaltsverzeichnis

1 Zusammenfassung	2
1.1 Aufgabenstellung	2
1.2 Abstract	3
1.3 Summary	4
2 Bildverarbeitung	6
2.1 Prinzip und Ablauf	6
2.1.1 RGB \rightarrow Gray	6
2.1.2 Antialiasing Filter und Downsampling	7
2.1.3 Kantendetektion	7
2.1.4 Selektion	7
2.1.5 Normalisierung	8
2.1.6 Polynomapproximation 1. Grades	9
2.2 Line Locking	12
2.2.1 Lineare Gewichtung	12
2.2.2 Hyperbolische Gewichtung	13
2.2.3 Gauss'sche Gewichtung	14
2.2.4 Tests	15
2.2.5 Line Locking ausgeschaltet	18
2.3 Kurvenapproximation	19
2.3.1 1. Algorithmus	19
2.3.2 2. Algorithmus	19
2.3.3 3. Algorithmus	19
2.3.4 Matlab Code (3. Algorithmus)	20
2.4 Bestandteile des Hauptprogramms	23
2.4.1 Initialisierung	23
2.4.2 Senden von UDP Paketen	23
2.4.3 Anzeigen des Kantenbildes	23
2.4.4 Extrahieren des Bildes aus dem Bildschirm	24
3 PIC Board für die Cardbus-Wirelesskarte	26
3.1 Zusammenfassung der Änderungen	26
3.1.1 Baudrate	26
3.1.2 Verzögerung	27
3.1.3 UDP to SERIAL	27
4 Ansteuerung der Servos	28
4.1 Taktfrequenz und Pulsweitenmodulation	28
4.2 RS232 Schnittstelle	29
4.3 Hauptprogramm	30
4.4 Automatische Abschaltung mittels Watchdog-Timer	31
4.5 Die Verbindung der zwei PIC's	31

5	Bedienungsanleitung	32
5.1	Vorbereitung	32
5.1.1	Kurs abkleben	32
5.1.2	ActiveX-Komponente XPLUG.OCX	32
5.1.3	Installation der Wireless-Karte	32
5.1.4	Aufladen der Akkus	32
5.2	Starten des Autos	33
5.3	Abschalten des Autos	33
5.4	Fehlerbehebung	33
5.5	Software Kompatibilität	34
A	ChipWeb 2.17 mit MPLAB 6.x und HI-TECH Compiler 8.20PL4	36
A.1	Änderungen MPLAB/HI-TECH	36
A.2	Änderungen ChipWeb	36
A.3	Änderungen MPLAB Compiler-Einstellungen	37
B	CVM Class Library	38
B.1	Warum CVM?	38
B.2	Installation	38
B.2.1	Voraussetzungen	39
B.2.2	Kopieren der Dateien	39
B.3	Arbeiten mit CVM	39
C	Entwicklungsumgebung MPLAB	40
C.1	MPLAB Configuration for MPC	41
C.1.1	MPC configuration	41
C.1.2	MPLAB configuration	41
C.1.3	DOS Box problems	43
D	Listings der Bildverarbeitung	45

Abbildungsverzeichnis

2.1	Bildverarbeitung mit Line Locking	12
2.2	Lineare Gewichtung	13
2.3	Hyperbolische Kurven für die Gewichtung	14
2.4	Gauss'sche Kurven für die Gewichtung	15
2.5	Gewichtungsfunktion gemäss Formel (2.2.1) mit $\alpha = 0.0875$	16
2.6	1. Durchgang ohne Line Locking, die Linie muss zuerst gefunden werden	16
2.7	2. Durchgang mit Line Locking und einer kleinen Störung	16
2.8	3. Durchgang mit Line Locking und durch eine Störung verkürzter Linie	17
2.9	4. Durchgang mit Line Locking und mehreren Störungen	17
2.10	5. Durchgang mit Line Locking und noch mehr Störungen	17
2.11	1. Durchgang mit Line Locking = OFF	18
2.12	2. Durchgang mit Line Locking = OFF und einer Störung	18
2.13	Kurvenapproximation, 2. Algorithmus	19
2.14	Kurvenapproximation, 3. Algorithmus	20
4.1	Ansteuerung der Servomotoren	28
5.1	Starten des Autos	33
B.1	Runtime Error	38
C.1	PIC Programmierung mit ICD2	40
C.2	Programmierung des PIC16F876 mit PICSTART	41
C.3	Install Language Tool Dialog	42
C.4	New Project Dialog	42
C.5	Node Properties Dialog	43

Listings

2.1	Das Bitmap wird in eine Matrix überführt und gleichzeitig wird die Graustufenumwandlung durchgeführt	6
2.2	Anwendung des Antialiasing Filters und Downsampling werden in einem Schritt durchgeführt. Der Downsampling Faktor beträgt $ds = 4$	7
2.3	Das Helligkeitsbild wird durch einen Hochpassfilter hindurch gelassen, dadurch werden Helligkeitsunterschiede (hell↔dunkel) detektiert	7
2.4	Für die Polynomapproximation wird nicht das ganze Helligkeits- oder Kantenbild benötigt	7
2.5	Normalisierung des Helligkeits- und Kantenbildes	8
2.6	Aufbau der Matrizen für die Polynomapproximation in MATLAB	9
2.7	Lösen des überbestimmten Gleichungssystems in MATLAB ohne die Benutzung der Links-Division $x = A \backslash y$	10
2.8	C-Code für die Polynomapproximation	10
2.9	Code-Ausschnitt aus der Member-Funktion <code>OnInitDialog()</code>	23
2.10	Member-Funktion <code>displayEdgePicture(...)</code>	23
2.11	Member-Funktion <code>extractPicture(...)</code>	24
3.1	Code-Ausschnitt <code>p18_ser.c</code>	27
3.2	Code-Ausschnitt <code>p18_udp.c</code>	27
D.1	<code>imageProcessing.h</code>	45
D.2	<code>imageProcessing.cpp</code>	45
D.3	<code>80211CarDlg.h</code>	54
D.4	<code>80211CarDlg.cpp</code>	55

Danksagung

An dieser Stelle möchten wir allen danken, die uns während dieser Semesterarbeit mit Rat und Tat zur Seite standen.

Besonderer Dank geht an unseren Betreuer, Prof. Dr. Guido M. Schuster, welcher während der ganzen Semesterarbeit uns mit sehr guten Tipps und Ratschlägen zur Seite stand und uns erfolgreich durch diese Semesterarbeit führte.

Weiterer Dank geht an die Laborassistenten des Medialabs, Peter Roffler, Daniel Ostojic und Matthias Engelhardt, die in kniffligen Hard- und Software-Fragen immer eine gute Hilfe leisteten und ohne ihre Hilfe hätte es sicher länger gedauert, das Auto fertig zu stellen.

1 Zusammenfassung

1.1 Aufgabenstellung

Das Ziel dieser Semesterarbeit ist es, ein RC Auto über IEEE 802.11b fernzusteuern, wobei die Fernsteuerung mittels einer Bildauswertung im PC autonom ist. Um UDP Pakete vom PC zum Auto zu schicken, wird das ER21 Wireless Development Board von Iosoft und die dazugehörige Software benutzt. Ausserdem wird eine 802.11b Wireless Cam DCS-1000W von D-Link benutzt, um Bilder vom Auto zum PC zu schicken. Die Bilder werden auf dem PC ausgewertet und die Fahrstrecke gefunden, die mit einem weissen Klebeband markiert ist. Nachdem der PC die Strecke gefunden hat, wird das Auto mittels der UDP Übermittlung auf der Strecke gehalten.

1.2 Abstract

Wie schon in der Aufgabenstellung erwähnt, ist es das Ziel, ein ferngesteuertes Auto so umzubauen, dass es selbstständig einer Linie auf dem Boden nachfahren kann. Dabei werden die Bilder von einer digitalen Kamera per ActiveX dem PC gesendet, welcher die Bilder nach einer Linie auswertet. Die ActiveX wirkt nur vom Auto bis zum Bildschirm des PC. Es ergab sich keine Möglichkeit, die Bilder in einem geeigneten Format direkt zu bekommen, z.B. über eine ActiveX-Funktion. Somit werden die Bilder vom Bildschirm ins RAM kopiert und danach in ein BITMAP gewandelt.

Um das Bild auf eine Linie hin untersuchen zu können, wird das Bild in einen Array (Matrix) gebracht, dann kann auf jedes Pixel des Bildes zugegriffen werden; dies wurde mit der *CVM Class Library*¹ realisiert. Die Bildverarbeitung enthält ein zusätzliches Feature mit der Bezeichnung *Line Locking*. Nachdem die Bildverarbeitungssoftware die Fahrspur gefunden hat, wird das nächste Bild mit der aktuellen Linie gewichtet. Die Gewichtungsfunktion ist eine Gauss'sche Funktion, damit wird der Bereich, der nahe der Linie ist relativ stark gewichtet (0.7...1); weit entfernte Bereiche werden ausgeblendet, weil die Funktion dort schon gegen Null konvergiert hat. Der Nutzen von Line Locking besteht darin, dass wenn man die richtige Linie einmal gefunden hat, sie sozusagen fixiert und nicht relevante Teile des Bildes damit ausblendet.

Das Halten des Autos auf der Linie während es fährt, wird mit einer modifizierten Bezier-Kurvenapproximation realisiert. Dabei wird die ungefähre Lage des Autos anhand der letzten drei berechneten Fahrwinkeln und der aktuellen Geschwindigkeit geschätzt.

Nach der Berechnung wird die Information von 4 Bytes in ein UDP Paket verpackt und an das Auto gesendet. Auf dem Auto sind zwei PICs installiert. Der eine PIC ist mit einer Cardbus (WLAN) ausgestattet; dieser nimmt das UDP Paket entgegen, extrahiert den Inhalt und schickt die Daten auf der seriellen Leitung dem zweiten PIC. Der zweite PIC verifiziert die empfangenen Daten und steuert dann die Servos oder den Fahrtenregler an.

¹<http://www.cvmlib.com>

1.3 Summary

Ausgangslage

Dieses Auto war vor dieser Semesterarbeit schon von zwei anderen Teams bearbeitet worden. Dabei wurde schrittweise die Hardware modifiziert; die AM-Fernsteuerung wurde durch eine FM-Steuerung ersetzt. Dieser Wechsel brachte nicht viel, da Störungen immer noch einen starken Einfluss auf die damals analoge Kamera und die Fahrzeugelektronik hatten. Darum war es Ziel dieser Semesterarbeit, die Fernsteuerung sowie die Kamera durch eine digitale Übertragungstechnik zu ersetzen.

Vorgehen und Ergebnisse der Bildverarbeitung

Als erstes wurden die Semesterarbeiten der letzten beiden Gruppen durchgelesen, um sich einen groben Überblick über die Fahrzeugelektronik und die Bildauswertung zu verschaffen. Da die erste Gruppe keine echte Bildauswertung implementiert hatte, hat die zweite Gruppe einen Code in MATLAB geschrieben, welche für unsere Semesterarbeit gute Ergebnisse zeigte. Dieser Code wurde in einer ersten Version als Stand-Alone-One-Picture-At-A-Time-Analyzer in C++ geschrieben. Die Ergebnisse des MATLAB-Skriptes und des C++-Programms waren in allen analysierten Bildern in der gleicher Grössenordnung (± 0.1). Danach wurde dieses Programm für die Hauptapplikation übernommen. Die Hauptapplikation liest das vom Auto erhaltene Bild vom Bildschirm und stellt es der Bildverarbeitungssoftware in einer Matrix zur Verfügung. Die grössten Probleme in der Erstellung der Hauptapplikation machten sich mit der ActiveX-Komponente und den Threads bemerkbar. Als diese Probleme gelöst waren, wurde das Line Locking implementiert und verschiedene Gewichtungsfunktionen wurden live am Auto ausprobiert. Gegen Schluss der Arbeit wurde die Bildverarbeitung an diversen Stellen ein wenig modifiziert um die Anzahl Code-Zeilen zu verringern und die Bildverarbeitung schneller zu machen; die Durchschnittszeit zwischen 2 UDP Paketen liegt bei 115ms. Unter 100ms bekommen die PICs Probleme, mit der Anzahl ankommenden Paketen Schritt zu halten.

Vorgehen und Ergebnisse rund um die Fahrzeugelektronik

Am Anfang bekamen wir die drei Komponenten Auto, PIC mit Wireless-Karte und Kamera. Am Anfang überlegten wir uns, wie wir die Daten dem Auto senden und wie diese aussehen sollten. Es war von Anfang an klar, dass wir UDP Pakete senden, weil wir Pakete verlieren dürfen und das nächste Paket relativ schnell gesendet wird. Als nächstes kam die Aufgabe, den Servo und den Fahrtenregler anzusteuern. Beide verlangen eine Pulsweitenmodulation. Da auf dem PIC mit der Wirelesskarte nur ein Pulsweitenmodulationsausgang vorhanden war, wurde ein zweiter PIC eingesetzt. Die beiden kommunizieren über die serielle Schnittstelle, da dies einfach mit einem RS232 Stecker realisiert werden konnte. Doch wie kommen die Daten vom UDP Paket zur seriellen Schnittstelle? Sobald nach etlichen Stunden der Compiler lief, konnte der Code des PICs studiert werden, der mitgeliefert wurde. Viele Funktionen waren schon implementiert. So war es nicht schwierig die UDP Pakete, die auf den Port 3000 kommen, auf die serielle Schnittstelle umzuleiten. Der zweite PIC verursachte manchmal Kopferbrechen. Manchmal lief das Programm, manchmal nicht. Mit dem Compiler wurden Beispielprogramme mitgeliefert, die bei der Problemlösung sehr behilflich waren. Zuerst wurde mal ganz einfach ein Servomotor angesteuert. Die Schwierigkeit dabei war, dass die PWM Periode von der Taktfrequenz abhängig ist. Bei der Quarzwahl machten wir nun einen Fehler, der später bei der seriellen Schnittstelle verherrend war. Wir wählten einen Uhrenquarz mit einer Taktfrequenz von 32768 Hertz. Dieser war viel zu langsam. Für unsere Anwendung reichte es mit einem Trick aus. Für die serielle Schnittstelle gibt es genau eine Bauderate, die für die zwei Taktfrequenzen einigermaßen übereinstimmt. Die PICs funktionierten nun. Jetzt wurden alle Komponenten auf das Auto montiert und die ersten Testfahrten unternommen. Es wurde schnell festgestellt, dass der Motor den Akku sehr belastet und die Spannung zusammenbrach. Ein zweiter Akku für die Elektronik schaffte da Abhilfe. Wir glaubten am Ziel angekommen, als wir feststellen mussten, dass das Auto die Geschwindigkeit nicht konstant halten konnte. Für den letzten Algorithmus ist dieses Verhalten eine Katastrophe. Ein neuer Motor, der für Tracks (tiefe Drehzahlen) gebaut wurde, brachte da einen

neuen Wind ins Spiel. Ein Getriebe passt nicht, da der Motor dann an den Reifen anstehen würde. Somit war dann auch die letzte Woche vorbei.

Probleme

Nicht unbedingt ein Problem, aber eine Umgewöhnung stellte das Programmieren in Microsoft Visual C++ dar. Besonders das Arbeiten mit den MFC-Klassen stellte sich am Anfang alles andere als leicht heraus. Eine grosse Hilfeleistung erbrachte in dieser Beziehung das Internet mit all seinen Programmier-Foren².

Das Programmieren der beiden PICs erfolgte mit den von den Hersteller gelieferten Beispielprogrammen. Dies funktionierte die meiste Zeit, aber manchmal zeigten die PICs bei gleicher Programmierung unterschiedliches Verhalten. Besonders langsam verlief die Semesterarbeit zu Beginn, da für die PICs nach einem geeignetem Compiler gesucht werden musste und auch hier manchmal die Software streikte (ohne einen vernünftigen Grund).

Ausblick Bildverarbeitung

Nächstes Ziel in der Bildverarbeitung wäre, diese adaptiv zu gestalten. Es wäre zum Beispiel möglich, die Geschwindigkeit des Autos an die Kurvenradien anzupassen. Dazu müsste die Bezier-Kurvenapproximation neu entworfen werden, da dort eine Standortschätzung des Autos vorgenommen wird. Es ist klar, dass die Standortschätzung je nach Geschwindigkeit andere Ergebnisse liefert; die Geschwindigkeit ist von der Stromversorgung abhängig und nimmt ab, wenn die Spannung abnimmt.

Auf der anderen Seite könnte noch am Line Locking weitergearbeitet werden, auch da könnte etwas Adaptives entworfen werden. Hier wird es allerdings schwieriger, etwas Neues und Besseres zu erstellen, da wir uns in dieser Semesterarbeit viel mit Line Locking befasst haben.

Die Bildverarbeitung könnte auch in Bezug auf die Anzahl zu analysierender Bilder optimiert werden. Konkret würde das heissen, man schätzt die Position des Autos und wenn man z.B. feststellt, die nächsten 2m geht es nur gerade aus, dann reduziert man die Anzahl der zu analysierender Bilder auf ein Drittel oder die Hälfte und schickt nur ein UDP Paket mit der Geschwindigkeitsangabe.

Ausblick Fahrzeugelektronik

Der Motor läuft zur Zeit im unteren Drehzahlbereich. Um das Drehmoment zu Erhöhen kann man noch ein Getriebe 3:1 einsetzen. Maxon oder auch der Modellbauladen besitzt passende. Das Getriebe kommt aber seitlich zu weit hinaus und streift die Räder. Vollgummiräder könnten auf der Drehbank halbiert werden um so den nötigen Platz für das Getriebe zu schaffen.

Am Schluss fällt immer das Stichwort "konstante Geschwindigkeit". Um dies wirklich zu erreichen, muss man einen Schrittmotor einfügen und diesen dann mit dem PIC ansteuern. Das bedingt, dass ein 1MHz Quarz für den PIC eingesetzt wird um die nötige Rechenleistung bereitzustellen.

IOsoft hat auf seiner Homepage eine Variante, um die Bilder einer Kamera zu grabben. Dies könnte die Verzögerung und den Stromverbrauch reduzieren.

Um die verschiedenen Geschwindigkeiten in den Griff zu bekommen, wäre es von Vorteil, wenn ein Sensor montiert würde und diese Daten dem Computer gesendet würden.

Das ganze Problem könnte man auch auf einem PC104, welcher auf dem Auto montiert würde, in Real-Time lösen. Die nicht geeignete Kamera von D-Link wäre leicht durch eine USB- oder FireWire-Kamera ersetzbar. Dann könnte die Bildverarbeitung weiter optimiert werden (s. Oben) oder auch die Ansteuerung der Servos und des Fahrtenreglers könnte durch eine Software-PWM-Emulation realisiert werden.

²<http://www.experts-exchange.com>
<http://www.codeproject.com>

2 Bildverarbeitung

In diesem Kapitel wird das Prinzip der Bildverarbeitung erläutert und welche Informationen dem Auto geschickt werden müssen, um das Auto der Linie entlang fahren zu lassen. Die einzelnen Verarbeitungsschritte in der Bildverarbeitung werden mit dem dazugehörigen Source-Code kommentiert.

2.1 Prinzip und Ablauf

1. RGB \rightarrow Gray
2. Antialiasing Filter und Downsampling
3. Kantendetektion
4. Selektion
5. Normalisierung
6. Polynomapproximation 1. Grades

2.1.1 RGB \rightarrow Gray

Für die Bildverarbeitung ist ein Graubild nötig. Dies erreicht man durch eine $RGB \rightarrow YC_rC_b$ Farbtransformation. Y steht für die Helligkeit (engl. luminance, brightness) und repräsentiert das Graubild. Die C Werte stehen für den Farbton (engl. chrominance) Rot und Blau, werden für die vorliegende Bildverarbeitung nicht gebraucht. Die Helligkeit lässt sich mittels $Y = 0.299R + 0.587G + 0.114B$ berechnen. Einen kurzen Überblick über die Darstellung von Farben liefert [1] S.90.

Die Umwandlung in Graustufen erfolgt bereits in der Funktion `BMP2MATRIX()` (Listing 2.1). Diese Funktion schreibt ein Bitmap in eine Matrix¹ und führt gleichzeitig die Graustufenumwandlung durch.

```
void BMP2MATRIX( BYTE* pBuffer, int pWidth, int pHeight, rmatrix* pPicture ) {
    // Find the number of padding bytes
    int padding = 0;
    int scanlinebytes = pWidth * 3;
    while( ((scanlinebytes + padding) % 4) != 0 ){
        padding++;
    }

    // Get the padded scanline width
    int psw = scanlinebytes + padding;

    long bufpos = 0;
    long tempX = 0;

    for( int y=0; y<pHeight; y++){
        for( int x=0; x<3*pWidth; x+=3, tempX++){
            bufpos = (pHeight - y - 1) * psw + x;
            // We read the pixel data from the buffer and write it to our matrix,
            // which holds our gray-scaled picture
            (*pPicture)(y+1,((tempX%320)+1)) = 0.299*pBuffer[bufpos] + 0.587*pBuffer[bufpos + 1]
                + 0.114*pBuffer[bufpos + 2];
        }
    }
}
```

Listing 2.1: Das Bitmap wird in eine Matrix überführt und gleichzeitig wird die Graustufenumwandlung durchgeführt

¹Unter Benutzung der CVM Library

2.1.2 Antialiasing Filter und Downsampling

Downsampling mit dem Faktor n bedeutet, dass nur jedes n -te Pixel verwendet wird. Das entspricht in der Signaltheorie einer Abtastung. Das bedeutet aber auch, dass die hohen Frequenzen im Bild (scharfe Kanten und Übergänge) entfernt werden müssen, was durch das Antialiasing Filter erreicht wird. In diesem Fall ist das Antialiasing Filter ein einfacher Tiefpass, dessen Implementierung eine einfache Durchschnittsbildung² durchführt. Das entstandene Bild wird im folgenden Helligkeitsbild genannt.

Wie in Listing 2.2 zu erkennen ist, wird das Tiefpassfilter und Abtasten in einem Schritt durchgeführt.

```
// Antialiasing Filter and Downsampling in one step
//-----
rmatrix intensityPictureB( SMALLHEIGHT, SMALLWIDTH );

for( i=1, m=1; i<=HEIGHT-ds; i+=ds, m++ ){
  for( j=1, n=1; j<=WIDTH-ds; j+=ds, n++ ){
    intensityPictureB(m,n) = (myPixell(i,j) + myPixell(i,j+1) + myPixell(i,j+2)
      + myPixell(i,j+3) + myPixell(i+1,j) + myPixell(i+1,j+1)
      + myPixell(i+1,j+2) + myPixell(i+1,j+3) + myPixell(i+2,j)
      + myPixell(i+2,j+1) + myPixell(i+2,j+2) + myPixell(i+2,j+3)
      + myPixell(i+3,j) + myPixell(i+3,j+1) + myPixell(i+3,j+2)
      + myPixell(i+3,j+3)) / (ds*ds);
  }
}
//-----
```

Listing 2.2: Anwendung des Antialiasing Filters und Downsampling werden in einem Schritt durchgeführt. Der Downsampling Faktor beträgt $ds = 4$.

2.1.3 Kantendetektion

Das Helligkeitsbild wird nun durch einen Hochpassfilter hindurch gelassen, Bildänderungen (dunkel \leftrightarrow hell) werden dadurch detektiert. Dieses Bild wird Kantenbild genannt.

Listing 2.3 illustriert die Filterung des Helligkeitsbildes mit einem einfachen Hochpass Filter.

```
// Filter the picture with a highpass to detect edges
//-----
rmatrix edgePictureB( SMALLHEIGHT-2, SMALLWIDTH-2 );

double k = 1/sqrt(8);
double s = 0.5;

for( i=2, m=1; i<=SMALLHEIGHT-1; i++, m++ ){
  for( j=2, n=1; j<=SMALLWIDTH-1; j++, n++ ){
    edgePictureB(m,n) = abs(-k*intensityPictureB(i-1,j-1) - s*intensityPictureB(i-1,j)
      + k*intensityPictureB(i-1,j+1) - s*intensityPictureB(i,j-1)
      + s*intensityPictureB(i,j+1) - k*intensityPictureB(i+1,j-1)
      + s*intensityPictureB(i+1,j) + k*intensityPictureB(i+1,j+1));
  }
}
//-----
```

Listing 2.3: Das Helligkeitsbild wird durch einen Hochpassfilter hindurch gelassen, dadurch werden Helligkeitsunterschiede (hell \leftrightarrow dunkel) detektiert

2.1.4 Selektion

Das oberste viertel vom Helligkeits- und Kantenbild wird für die Liniensuche nicht berücksichtigt.

```
// Grab lower part of the picture
//-----
rmatrix intensityPicture( FINALHEIGHT, FINALWIDTH );
rmatrix edgePicture( FINALHEIGHT, FINALWIDTH );
```

²Das aktuelle Pixel $p(i, j)$ wird mit seinen unmittelbar angrenzenden Pixeln zu einem neuen Pixel (= Durchschnittswert) berechnet.

```

for( i=14, m=1; m<=FINALHEIGHT; i++, m++ ){
    for( j=1, n=1; j<=FINALWIDTH; j++, n++ ){
        intensityPicture(m,n) = intensityPictureB(i,j);
        edgePicture(m,n) = edgePictureB(i,j);
    }
}
//-----

```

Listing 2.4: Für die Polynomapproximation wird nicht das ganze Helligkeits- oder Kantenbild benötigt

2.1.5 Normalisierung

Das Helligkeits- und Kantenbild werden in der Polynomapproximation miteinander verrechnet. Dazu müssen die beiden Bilder aufeinander abgestimmt (= normalisiert) werden.

Vorgängig muss noch der kleinste Bildwert an allen Pixeln abgezogen werden. Der Mittelwert \bar{x}_k eines Bildes $g_k(z, s)$ (z =Zeile, s =Spalte) errechnet sich nach

$$\bar{x}_k = \frac{1}{NM} \sum_{z=1}^N \sum_{s=1}^M g_k(z, s) \quad (2.1.1)$$

mit N =Anzahl Zeilen und M =Anzahl Spalten des Bildes. Um die Standardabweichung s_k zu bekommen, setzt man Gleichung (2.1.1) in Gleichung (2.1.2) ein:

$$s_k = \sqrt{\frac{1}{NM-1} \sum_{z=1}^N \sum_{s=1}^M [g_k(z, s) - \bar{x}_k]^2} \quad (2.1.2)$$

In Listing 2.5 ist die Normalisierung des Helligkeits- und Kantenbildes dargestellt:

```

// Normalize both pictures
//-----
double avgEdge=0, avgIntensity=0, tempEdge=0, tempIntensity=0, stdEdge=0, stdIntensity=0;
double minIntensity=255, minEdge=255, tempIntensityMin=0, tempEdgeMin=0;

int minIntensityPOS = intensityPicture.indofmin();
int minEdgePOS = edgePicture.indofmin();

int rowI = minIntensityPOS % FINALHEIGHT;
int colI = (minIntensityPOS-(minIntensityPOS % FINALHEIGHT))/FINALHEIGHT+1;
int rowE = minEdgePOS % FINALHEIGHT;
int colE = (minEdgePOS-(minEdgePOS % FINALHEIGHT))/FINALHEIGHT+1;

if( rowI == 0 ){
    rowI = FINALHEIGHT;
    colI--;
}

if( rowE == 0 ){
    rowE = FINALHEIGHT;
    colE--;
}

minIntensity = intensityPicture(rowI,colI);
minEdge = edgePicture(rowE,colE);

for( i=1; i<=FINALHEIGHT; i++ ){
    for( j=1; j<=FINALWIDTH; j++ ){
        // tempIntensity & tempEdge hold the sum of all pixels
        tempIntensity += abs(intensityPicture(i,j)-minIntensity);
        tempEdge += abs(edgePicture(i,j)-minEdge);
    }
}

avgIntensity = tempIntensity / (FINALHEIGHT*FINALWIDTH);
avgEdge = tempEdge / (FINALHEIGHT*FINALWIDTH);

tempIntensity=0;
tempEdge=0;

for( i=1; i<=FINALHEIGHT; i++ ){

```

```

    for( j=1; j<=FINALWIDTH; j++){
        tempIntensity += ((intensityPicture(i,j) - avgIntensity)
            * (intensityPicture(i,j) - avgIntensity));
        tempEdge += ((edgePicture(i,j) - avgEdge) * (edgePicture(i,j) - avgEdge));
    }
}

stdIntensity = sqrt(tempIntensity/(FINALHEIGHT*FINALWIDTH-1));
stdEdge = sqrt(tempEdge/(FINALHEIGHT*FINALWIDTH-1));

for( i=1; i<=FINALHEIGHT; i++){
    for( j=1; j<=FINALWIDTH; j++){
        intensityPicture(i,j) = intensityPicture(i,j) / stdIntensity;
        edgePicture(i,j) = edgePicture(i,j) / stdEdge;
    }
}
//-----

```

Listing 2.5: Normalisierung des Helligkeits- und Kantenbildes

2.1.6 Polynomapproximation 1. Grades

Aufgrund der normalisierten Bildwerte im Helligkeits- und Kantenbild wird eine Gerade errechnet, die eingefügt im Originalbild, sich mit der Richtung des Klebebandes deckt. Es muss ein überbestimmtes Gleichungssystem gelöst werden, es wird nach der Lösung mit dem kleinsten quadratischen Fehler gesucht.

Im Folgendem wird der Algorithmus beschrieben, wie man in einem Bild eine festgelegte Linie finden kann. Teile des Listings (MATLAB Datei *test.m*) wurden aus [2] übernommen. Dieser Algorithmus zeigte gute Resultate in Bezug auf das Finden einer Linie im Bild. Es musste zuerst mit Bildern ohne darin enthaltenen Störungen (zusätzliche Störlinie, weisse Gebiete) gearbeitet werden, damit der Algorithmus ausgiebig getestet werden konnte.

```

% build the y=Ax system
[rmax, cmax]=size(edgePicture);
order=1; % the order of the approximation
A=zeros(rmax*cmax,order+1);
y=zeros(rmax*cmax,1);
for r=1:rmax
    %build up the weight vector
    R=[1];
    for o=1:order
        R=[r^o, R]
    end
    for c = 1:cmax
        index=(r-1)*cmax+c;
        % the row is linearly important, so is the picture intensity and the value of the edge picture
        A(index,:)=(r/rmax)*intensityPicture(r,c)*edgePicture(r,c)*R;
        y(index) =(r/rmax)*intensityPicture(r,c)*edgePicture(r,c)*c;
    end
end
end

```

Listing 2.6: Aufbau der Matrizen für die Polynomapproximation in MATLAB

Das Listing 2.6 zeigt, wie die Matrizen aufgebaut werden. Die Variablen r und c stehen für Zeile (engl. row) und Spalte (engl. column). Angenommen, der Downsampling Faktor betrage 4 und das Originalbild habe die Breite 320 und die Höhe 240 Pixel, dann haben die abgetasteten Bilder (Helligkeits- und Kantenbild) eine Breite von 80 und eine Höhe von 60 Pixel.³

Aus Listing 2.6 würde $r_{\max}=60$ und $c_{\max}=80$. Es werden dann zwei Matrizen A und y gebildet und in der inneren FOR-Schleife mit Werten gefüllt. Der Zusammenhang zwischen den Variablen r , c und $index$ ist der folgende:

³Eigentlich wäre die Höhe der Bilder $60 - 60/4 = 45$, weil für die Polynomapproximation das oberste Viertel der Bilder nicht verwendet wird. Im Folgendem wird der Algorithmus ohne die genannte Bildverkleinerung vorgestellt, die Arbeitsweise der Polynomapproximation wird dadurch ja nicht beeinträchtigt.

R	r	c	index
[1 1]	1	1	1
[1 1]	1	2	2
[1 1]	1	3	3
[1 1]	1	4	4
⋮	⋮	⋮	⋮
[1 1]	1	80	80
[2 1]	2	1	81
[2 1]	2	2	82
[2 1]	2	3	83
⋮	⋮	⋮	⋮
[60 1]	60	80	4800

Wie man weiter erkennen kann, ergibt es für die Matrix A 2 Spalten und 4800 Zeilen, für die Matrix y eine Spalte und 4800 Zeilen. Nun muss das überbestimmte Gleichungssystem $A \cdot x = y$ gelöst werden:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{48001} & a_{48002} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{4800} \end{bmatrix}$$

x_1 ist die Steigung der Geraden und x_2 ist der Offset bezüglich der linken oberen Ecke des Bildes. In MATLAB würde der Befehl $x = A \backslash y$ die optimalen x liefern, d.h. das x , dass den kleinsten quadratischen Fehler erzeugt. Das Problem dabei ist, dass eine Matrix invertiert werden muss und diese Rechenoperation sehr aufwendig ist. Für eine 2×2 Matrix ist eine Inversion sehr einfach durchführbar. Von der Matrix

$$B = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

erhält man deren Invertierte durch

$$B^{-1} = \frac{1}{\det B} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} .$$

Der folgende Code-Ausschnitt löst das überbestimmte Gleichungssystem:

```
% in case (A'*A) is a 2x2
Q=A'*A;
Qi=Q;
temp=Qi(1,1);
Qi(1,1)=-Qi(2,2);
Qi(2,2)=-temp;
temp=Q*Qi;
Qi=Qi/temp(1,1);
x=Qi*A'*y;
```

Listing 2.7: Lösen des überbestimmten Gleichungssystems in MATLAB ohne die Benutzung der Links-Division $x = A \backslash y$

Im folgendem Listing 2.8 ist der C-Code der Polynomapproximation abgebildet. Wie man weiter erkennen kann, wird durch den Einsatz der CVM Library die Notation für Matrizen derjenigen Notation von MATLAB sehr ähnlich, dadurch wird das Umschreiben von MATLAB-Code in C-Code erleichtert.

```
// Polynomapproximation 1. Order
//-----
rmatrix A( FINALHEIGHT*FINALWIDTH, 2 );
rmatrix AT( 2, FINALHEIGHT*FINALWIDTH );
rmatrix y( FINALHEIGHT*FINALWIDTH, 2 );
rmatrix R( 1, 2 );
rmatrix Q( 2, 2 );
```

```

rmatrix Qi( 2, 2 );
rmatrix templ( 2,2 );
rmatrix x( 2,2 );
int index=0;
double temp=0;

for( i=1; i<=FINALHEIGHT; i++){
  R(1,1) = i;
  R(1,2) = 1;
  for( j=1; j<=FINALWIDTH; j++){
    index = (i - 1) * FINALWIDTH + j;
    A(index,1) = ((double)i / FINALHEIGHT)
                 * edgePicture(i,j) * intensityPicture(i,j) * R(1,1);
    A(index,2) = ((double)i / FINALHEIGHT)
                 * edgePicture(i,j) * intensityPicture(i,j) * R(1,2);
    y(index,1) = ((double)i / FINALHEIGHT)
                 * edgePicture(i,j) * intensityPicture(i,j) * j;
    y(index,2) = 0;
  }
}

AT.transpose(A);
Q = AT * A;
Qi = Q;
temp = Qi(1,1);
Qi(1,1) = -Qi(2,2);
Qi(2,2) = -temp;
templ = Q * Qi;
Qi = Qi / templ(1,1);
x = Qi * AT * y;
//-----

```

Listing 2.8: C-Code für die Polynomapproximation

2.2 Line Locking

Bei der Polynomapproximation werden die Zeilen des Helligkeits- und Kantenbildes von oben nach unten linear gewichtet, und zwar haben tiefer gelegene Bildzeilen mehr Gewicht als die höher gelegenen Bildzeilen. Es findet aber bis anhin keine Gewichtung innerhalb einer Bildzeile statt.

Diese Version der Bildverarbeitung ist einigermaßen robust. Einzige Schwäche ist, dass sie stark auf Störungen, also zusätzliche weisse Gebiete im Bildfeld, reagiert. Dadurch kann die Linie auf dem Boden nicht exakt oder gar nicht mehr wiedergegeben werden. Wenn man bedenkt, wie die Polynomapproximation gemacht wird, ist dieses Verhalten des Programms nicht falsch, aber für die Steuerung des Autos denkbar ungeeignet.

Die Lösung für dieses Problem setzt sich aus der Gewichtung des Helligkeits- und Kantenbildes mit einem sogenannten Gewichtungsbild zusammen. Voraussetzung für dieses Vorgehen ist, dass das Auto zu Beginn auf die Linie ausgerichtet wird und dadurch die Linie gefunden wird. Danach wird aufgrund der korrekten Linie ein Gewichtungsbild erstellt, das mit dem nächsten von der Kamera kommenden Bild pixelweise multipliziert wird.

Der Aufbau des Gewichtungsbildes ist sehr einfach, vorausgesetzt man hat die richtigen Koordinaten der Linie (Steigung und Offset). Die Gewichtung eines Pixels in einer Zeile des Gewichtungsbildes ist abhängig von der Distanz dieses Pixels zur Linie. Angenommen, die Linie verläuft vertikal in der Mitte des Bildes, so ist das daraus berechnete Gewichtungsbild in der Mitte hell und wird gegen den Rand hin dunkler; so werden die irrelevanten Bildbereiche ausgeblendet und gleichzeitig wird eine Verbesserung der Linien-Findung erreicht.

Abbildung 2.1 zeigt, wie dieses Feature in das bestehende Bildverarbeitungsprogramm eingefügt wird.

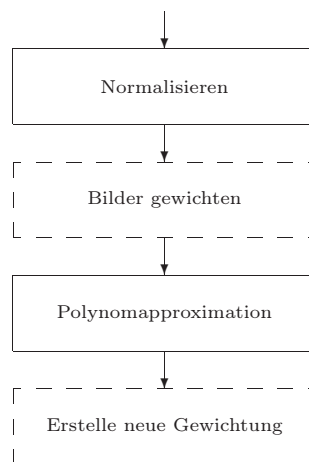


Abbildung 2.1: Bildverarbeitung mit Line Locking

2.2.1 Lineare Gewichtung

Die Bilder nach der Selektion (Abbildung haben eine Breite von 78 Pixel und eine Höhe von 44 Pixel, letzteres ist für das Line Locking nicht von Bedeutung. Für die nachfolgende Diskussion der Gewichtungsfunktionen geht man von einer Linie aus, die vertikal durch die Mitte des Bildes geht.

Eine lineare Gewichtung nach Abbildung 2.2 führt bereits zu einer sehr robusten Fixierung der Linie. Allerdings erkennt man auch, dass die weit entfernten Pixel immer noch einen relativ grossen Gewichtungsfaktor haben, z.B. hat das 30. Pixel immer noch ein Gewicht von etwa 0.25 oder 25%. Mit anderen Worten, die lineare Gewichtung ist nicht sehr selektiv und hat eine relativ breite Streuung.

Der untere Teil der Abbildung 2.2 visualisiert die im oberen Teil dargestellte Gewichtungsfunktion, so lässt sich der Zusammenhang zwischen Gewichtungsfunktion und Gewichtungsbild besonders anschaulich darstellen.

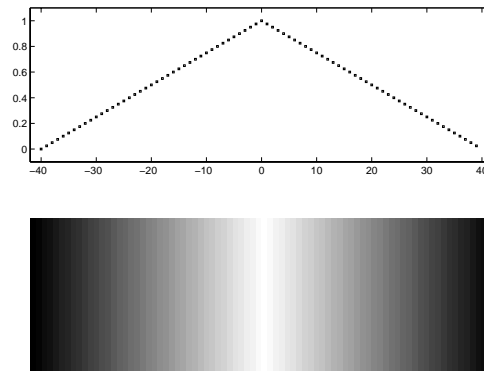


Abbildung 2.2: Lineare Gewichtung

2.2.2 Hyperbolische Gewichtung

Soll die Gewichtung selektiver sein, drängen sich andere Gewichtungsfunktionen auf. Die im folgenden dargestellten Funktionen zeigen nur eine kleine Auswahl an möglichen hyperbolischen Funktionen, von denen vier ausgewählt wurden.

Abbildung 2.3(a):
$$g(x) = \begin{cases} 1 & , x = 0 \\ 1/|x| & , x \neq 0 \end{cases}$$

Abbildung 2.3(b):
$$g(x) = \begin{cases} 1 & , x = 0 \\ 1/x^2 & , x \neq 0 \end{cases}$$

Abbildung 2.3(c):
$$g(x) = \begin{cases} 1 & , x = 0 \\ \frac{1}{\sqrt{|x|}} & , x \neq 0 \end{cases}$$

Abbildung 2.3(d):
$$g(x) = 1/1.15^{|x|}$$

Probiert man diese Gewichtungsfunktionen aus, so stellt sich heraus, dass die Gewichtungsfunktionen nach Abbildung 2.3(a) und Abbildung 2.3(b) keine zufriedenstellende Resultate für ein stabiles Line Locking hervor bringen; die Gewichtung für das 2. Pixel nach Abbildung 2.3(a) beträgt bereits 0.5 oder 50%, bei der Abbildung 2.3(b) bereits 0.25 oder 25%. Bei diesen Gewichtungen kann es durchaus vorkommen, dass die Bildverarbeitung die Linie nicht mehr findet. Das kann dann geschehen, wenn das Bild stark ruckelt (z.B. wenn das Auto leicht geschubst wird). Die anderen zwei Gewichtungsfunktionen nach Abbildung 2.3(c) und Abbildung 2.3(d) zeigen ein für unser Auto besseres Verhalten.

Die Gewichtungsfunktion nach Abbildung 2.3(c) wurde am Auto ausprobiert und das Verhalten der Bildverarbeitung hinsichtlich Line Locking wurde enorm verbessert. Wenn die Bildverarbeitung mal die richtige Linie hatte, konnte man Störungen ins Bildfeld der Kamera legen und die Linie änderte sich nur ganz wenig in Richtung der Störungen. Somit haben grosse Störungen im Bild nur kleine Auswirkungen, wenn man bedenkt, dass das letzte Pixel bei der Abbildung 2.3(c) nur noch einen Gewichtungsfaktor von etwa 0.19 oder 19% besitzt. Ein Nachteil dieser Gewichtungsfunktion ist, dass die unmittelbar in der Nähe der Linie liegenden Pixel (± 10 Pixel ab der Linie) nicht mehr stark gewichtet werden; man kann auch sagen, der Bereich um die Linie wird nicht stark genug ausgeleuchtet. Wünschenswert wäre eine Gewichtungsfunktion, die den Bereich um die Linie stark ausleuchtet und die am weitesten entfernten Pixel

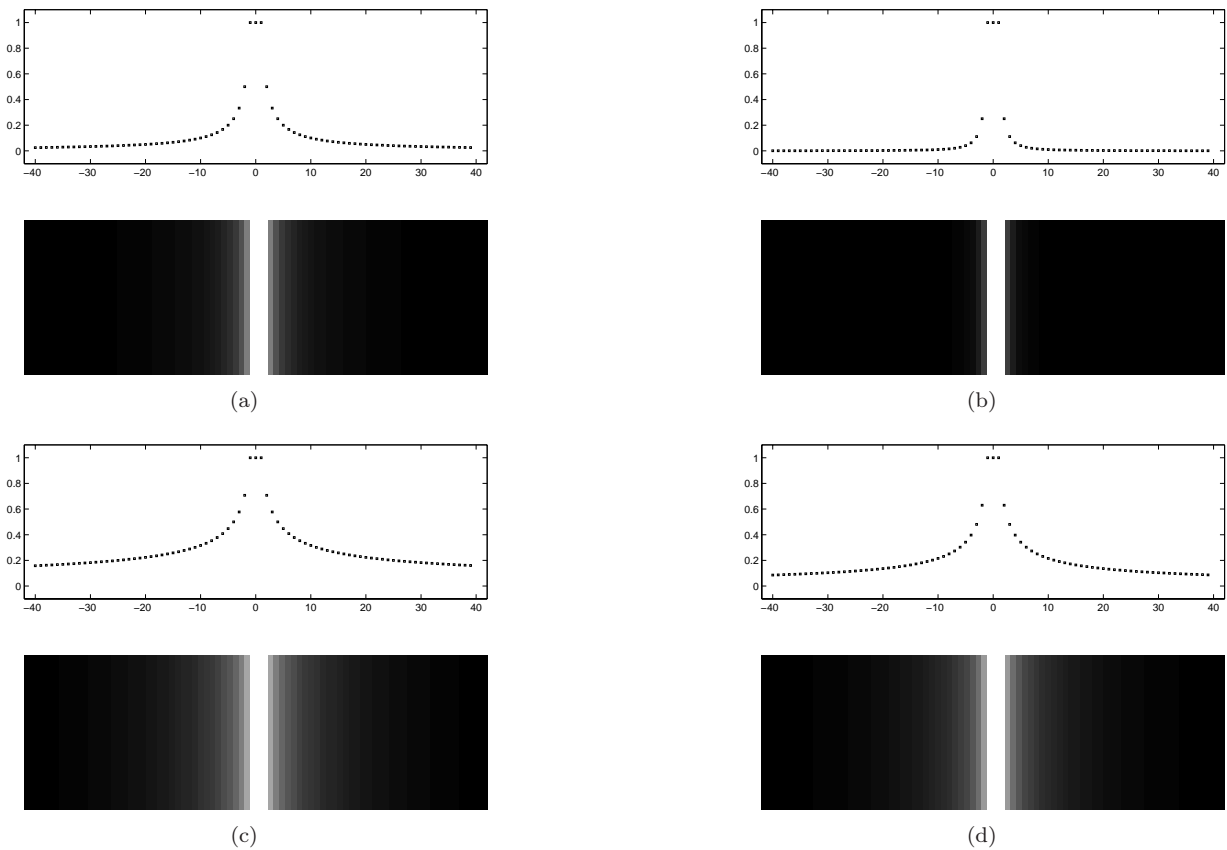


Abbildung 2.3: Hyperbolische Kurven für die Gewichtung

praktisch auf Null setzt. Diese Art Gewichtungsfunktionen werden im nächsten Unterkapitel vorgestellt. Die Gewichtungsfunktion nach Abbildung 2.3(d) ist vollständigshalber abgebildet, aber es wird hier nicht weiter darauf eingegangen.

2.2.3 Gauss'sche Gewichtung

In den letzten zwei Unterkapiteln wurden die lineare Gewichtungsfunktion und einige hyperbolische Gewichtungsfunktionen vorgestellt. Die lineare Gewichtung hat den Nachteil, dass sie nicht selektiv genug ist. Der grösste Nachteil der hyperbolischen Gewichtungsfunktionen ist, dass der Bereich um die Linie nicht stark genug ausgeleuchtet wird und die von der Linie am weitesten entfernten Pixel immer noch mit etwa 0.19 oder 19% gewichtet werden.⁴

Die oben genannten Nachteile können mit der Gauss'schen Funktion als Gewichtungsfunktion wettgemacht werden. Die Gauss'sche Funktion (Gauss'sche Glockenkurve) wird definiert durch

$$g(x) = e^{-(\alpha x)^2} \quad (2.2.1)$$

sie ist symmetrisch zur Ordinatenachse und nähert sich asymptotisch der Abszissenachse umso schneller, je grösser $|\alpha|$ ist.

Wie in Abbildung 2.4(a)-2.4(c) ersichtlich, ist eine Gewichtungsfunktion möglich, die die oben genannten Nachteile der linearen und hyperbolischen Gewichtung nicht mehr besitzt. Einerseits wird der Bereich um die Linie gleichmässig ausgeleuchtet, andererseits werden weit von der Linie entfernte Bereiche ausgeblendet.

⁴Voraussetzung ist immer noch eine vertikale Linie durch die Bildmitte.

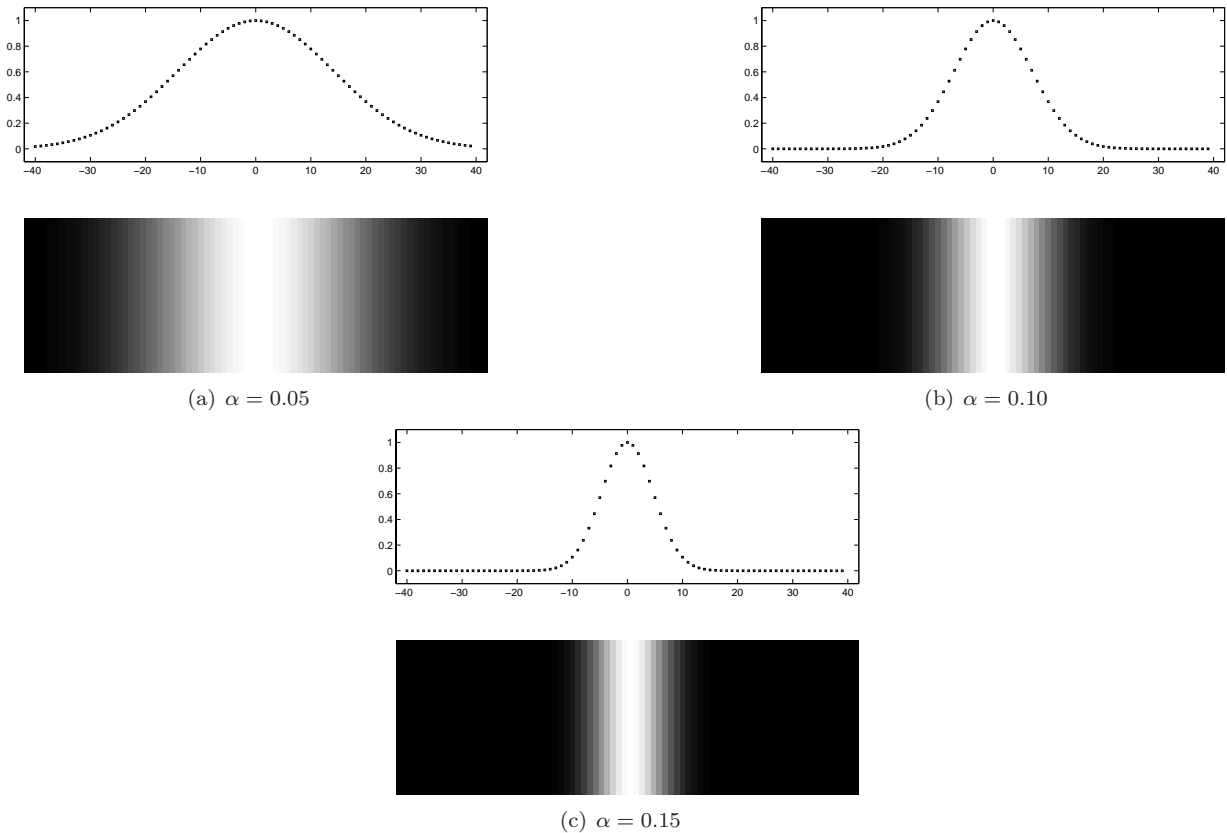


Abbildung 2.4: Gauss'sche Kurven für die Gewichtung

Das Verhalten der Bildverarbeitung mit Line Locking und einer Gauss'schen Gewichtungsfunktion ist gegenüber der linearen und hyperbolischen Gewichtungsfunktion um einiges besser in Bezug auf Störungsempfindlichkeit. Nachdem das erste Bild analysiert wurde und daraus das Gewichtungsbild berechnet wurde, kann man in das Bildfeld der Kamera Störungen (weisses Feld, zusätzliche Linien) legen, und die berechnete Linie weicht kein bisschen von der echten Linie am Boden ab. Nächstes Unterkapitel zeigt die Testergebnisse mit eingeschaltetem Line Locking.

2.2.4 Tests

Das Verhalten der Bildverarbeitung mit eingeschaltetem Line Locking wurde ausgiebig getestet und die Resultate sind in den Abbildungen 2.6-2.10 ersichtlich. Es wurde die Gewichtungsfunktion aus Abbildung 2.5 verwendet.

Das Bildverarbeitungsprogramm lässt sich durch die Variable `WRITE_TEST_PICTURES`⁵ in den schrittweisen Modus schalten. In diesem Modus analysiert das Programm ein Bild und stoppt dann den Programmablauf (akustisch wahrnehmbar durch einen Pieps). Das Original-, Kanten-, Linien- und Gewichtungsbild werden gespeichert, sodass diese Bilder zu Dokumentationszwecken umbenannt oder in ein anderes Verzeichnis kopiert werden können. Mit einem Klick auf den Start-Knopf analysiert das Programm das nächste Bild. Nun kann zwischen dem Betätigen des Start-Knopfes eine Störung ins Bild gelegt und dann das Verhalten der Bildverarbeitung mit aktiviertem Line Locking⁶ beobachtet werden.

⁵Listing D.2 Seite 45, Zeile 8

⁶Das Line Locking lässt sich ebenfalls ein- und ausschalten; Listing D.2 Seite 45, Zeile 6

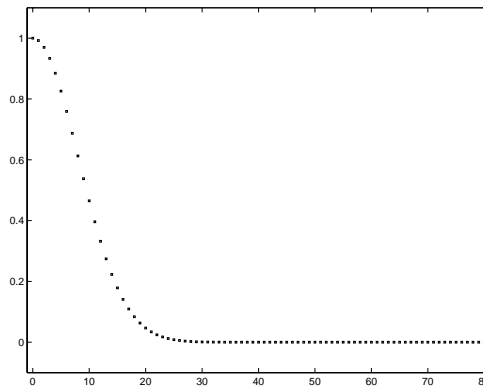


Abbildung 2.5: Gewichtungsfunktion gemäss Formel (2.2.1) mit $\alpha = 0.0875$

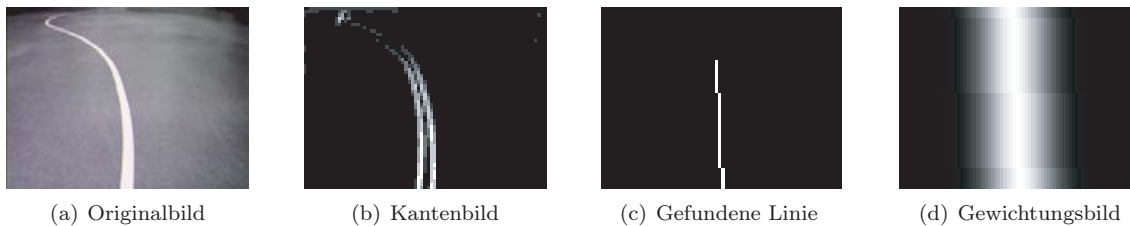


Abbildung 2.6: 1. Durchgang ohne Line Locking, die Linie muss zuerst gefunden werden

Die Voraussetzungen für das erste Bild, dass das Bildverarbeitungsprogramm analysieren soll, sind in Abbildung 2.6(a) gezeigt; gleichmässige Beleuchtung des Blickfeldes der Kamera und eine saubere, klar identifizierbare und helle Linie am Boden. Daraus werden nun die Liniendaten, Steigung und Offset, berechnet und aus diesen Daten wird ein Gewichtungsbild (Abbildung 2.6(d)) für das nächste zu analysierende Bild erstellt.

Nachdem der Algorithmus die Linie gefunden hat, wird in das Bild (Abbildung 2.7(a)) eine weitere Linie hinzugefügt, die die Störung simulieren soll. Die Störung ist im Kantenbild gut erkennbar und weiter ist in Abbildung 2.7(c) zu erkennen, dass die Störung praktisch keinen Einfluss auf die Fahrspur (richtige Linie) hat. Würde man das Gewichtungsbild aus Abbildung 2.6 über das Original- oder Kantenbild aus Abbildung 2.7 legen, so wäre deutlich erkennbar, dass die Störung relativ stark ausgeblendet wird.

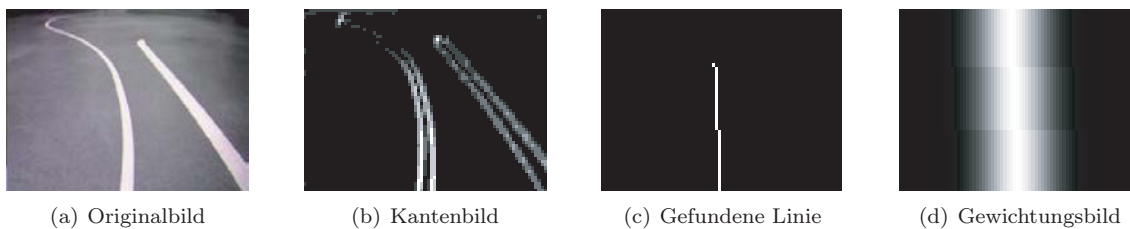


Abbildung 2.7: 2. Durchgang mit Line Locking und einer kleinen Störung

Im 3. Durchgang wurde zur vorhergehenden Störung eine neue Störung hinzugefügt. Die neue Störung soll die Fahrspur verkürzen und die andere Störlinie nicht gross verdecken, wie in Abbildung 2.8(a) gut zu erkennen ist. Und wieder berechnet der Algorithmus aufgrund des letzten errechneten Gewichtungsbildes (Abbildung 2.7(d)) die neue Linie, die sich mit der Fahrlinie deckt. Bemerkenswert sind die visualisierten Störungen im Kantenbild (Abbildung 2.8(b)), und wie sie einen vernachlässigbaren kleinen Einfluss auf die Bildverarbeitung haben.

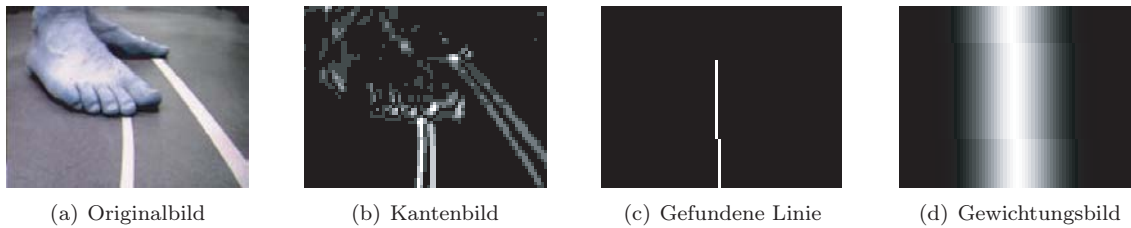


Abbildung 2.8: 3. Durchgang mit Line Locking und durch eine Störung verkürzter Linie

Wie eingangs schon erwähnt, stoppt das Programm nach jeder Bildanalyse die weitere Ausführung, somit kann zwischen zwei aufeinander folgenden Bildern eine beliebige Störung hinzugefügt oder entfernt werden. Nach dem 3. Durchgang wurde der Fuss entfernt und es wurden neue Störungen eingebaut (Abbildung 2.9(a)); eine zusätzliche Linie und zwei versetzte weisse Flächen. Auch hier zeigt das Kantenbild aus Abbildung 2.9 deutliche Veränderungen gegenüber dem Kantenbild aus Abbildung 2.6, aber der Bildverarbeitungsalgorithmus findet aufgrund des Gewichtungsbildes die Fahrspur ohne Probleme.

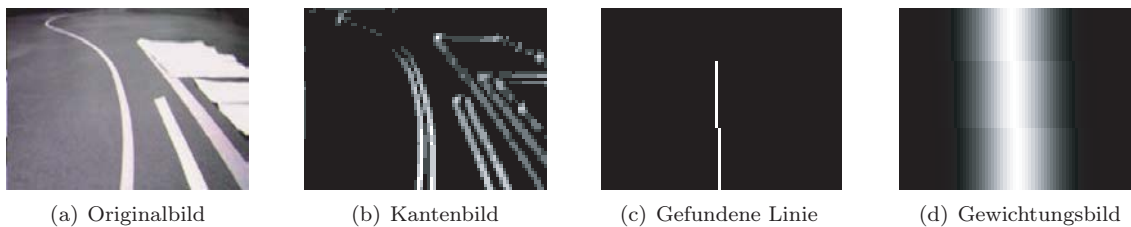


Abbildung 2.9: 4. Durchgang mit Line Locking und mehreren Störungen

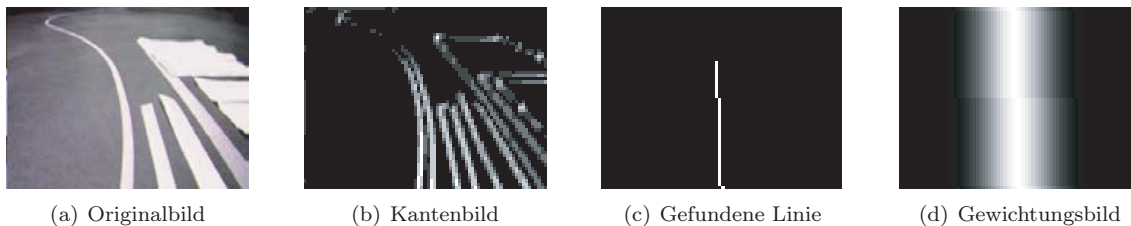


Abbildung 2.10: 5. Durchgang mit Line Locking und noch mehr Störungen

Im letzten Durchgang wurde noch eine zusätzliche Linie als Störung eingefügt. Im Kantenbild der Abbildung 2.10 sind nun schon einige Kanten vorhanden, aber wie schon in den letzten Abschnitten erwähnt, üben diese Störungen keinen merklichen Einfluss auf das Finden der Fahrspur aus.

Die aus Abbildung 2.5 verwendete Gewichtungsfunktion unterstützt die Bildverarbeitung mit guten Ergebnissen, was das Festhalten der Linie betrifft. Natürlich kann der Parameter α verändert werden, wird er kleiner gemacht, so wird die Gewichtungsfunktion breiter und das Bild wird mehr ausgeleuchtet. Wird der gleiche Parameter grösser gemacht, so konvergiert die Gewichtungsfunktion schneller gegen Null, was auch einer geringeren Ausleuchtung des Bildfeldes der Kamera entspricht. Nun sollte aber bei der Wahl des Parameters α darauf geachtet werden, dass dieser nicht zu gross gewählt wird. Bei den Tests mit Line Locking bewegte sich das Auto nicht und die Störungen wurden nur ins Bild gelegt. Wenn das Auto mit aktiviertem Line Locking fährt, so bewegt sich zwangsweise auch die Fahrspur. Der Parameter der Gewichtungsfunktion sollte nun so gewählt werden, dass wenn z.B. sich die Linie zwischen zwei analysierten Bildern stark seitwärts bewegt, sie noch im Ausleuchtungsbereich bleibt. Sonst kann es vorkommen, dass die Bildverarbeitung im Ausleuchtungsbereich keine Fahrspur mehr sieht und dann von der Fahrspur abkommt.

In den vorhergehenden Unterkapitel wurden drei Klassen von Gewichtungsfunktionen beschrieben, wobei diese Gewichtungsfunktionen keinen Anspruch auf Vollständigkeit haben. Man könnte noch n weitere Gewichtungsfunktionen aufstellen und besprechen, aber im Grunde genommen ist es immer der gleiche Analysevorgang. Jede Gewichtungsfunktion hat ihre Vor- und Nachteile und man musste sich bei der vorliegenden Arbeit auf eine Gewichtungsfunktion festsetzen. Zuerst wurde die hyperbolische Gewichtungsfunktion nach Abbildung 2.3(c) implementiert, was schon zu sehr ansehnlichen Ergebnissen führte. Doch vorhandene Störungen im Bild hatten immer noch einen sichtbaren Einfluss auf das Finden der Fahrspur und so nahm man sich mal die Zeit und untersuchte noch andere Gewichtungsfunktionen. Die Gauss'sche Funktion als Gewichtungsfunktion zeigte die eindeutig besten Resultate für das robuste Finden der Fahrspur, es musste nur noch ein geeigneter Wert für den Parameter α bestimmt werden. Dieser Wert wurde zu Beginn iterativ mit MATLAB bestimmt und anschliessend wurden einige Werte in der Bildverarbeitung eingesetzt und dann ausgetestet. Daraus resultierte für α der Wert 0.0875.

2.2.5 Line Locking ausgeschaltet

Es bleibt noch zu erwähnen, wie sich die Bildverarbeitung mit ausgeschaltetem Line Locking verhält. Das erste Bild wird analysiert und die Fahrspur gefunden, die Bildverarbeitung findet die Linie also ohne Probleme. Nun wurde in Abbildung 2.12(a) eine zusätzliche Linie eingefügt und das Resultat, dass der Algorithmus ausgibt ist in Abbildung 2.12(c) dargestellt. Vergleicht man dieses Linienbild mit jenem aus Abbildung 2.11(c), so stellt man fest, dass die Bildverarbeitung von der tatsächlichen Fahrlinie abgekommen ist.

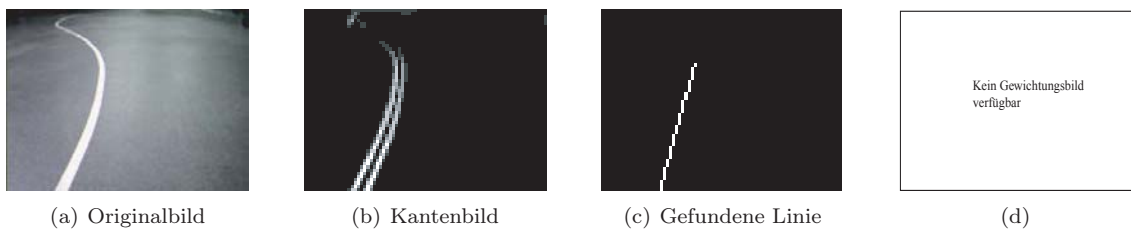


Abbildung 2.11: 1. Durchgang mit Line Locking = OFF

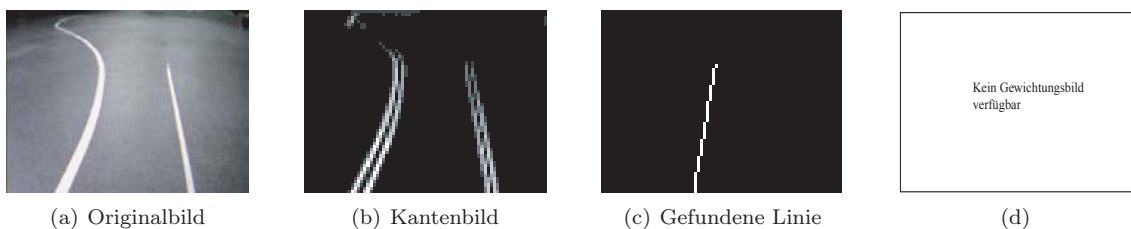


Abbildung 2.12: 2. Durchgang mit Line Locking = OFF und einer Störung

2.3 Kurvenapproximation

2.3.1 1. Algorithmus

Beim ersten Algorithmus wurde die Steuerung nur anhand des Winkels der Gerade vorgenommen. Es zeigte sich aber schnell, dass dies nicht den gewünschten Effekt bringt. Somit wurde später noch der Durchgang der Linie am Bildrand unten dazu addiert. Dies funktionierte etwas besser, aber das Auto steuerte immer noch relativ wild umher.

2.3.2 2. Algorithmus

Im zweiten Algorithmus wurde versucht mit einer Bezierkurve die Gerade möglichst gut zu erwischen. Die beiden Hilfspunkte der Kurve sind im Schnittpunkt der Gerade mit der Mittelsenkrechten des Bildes. Sind die Hilfsvektoren nicht realistisch, werden sie auf ein Wert gezwungen. Sobald die Kurve den eingestellten

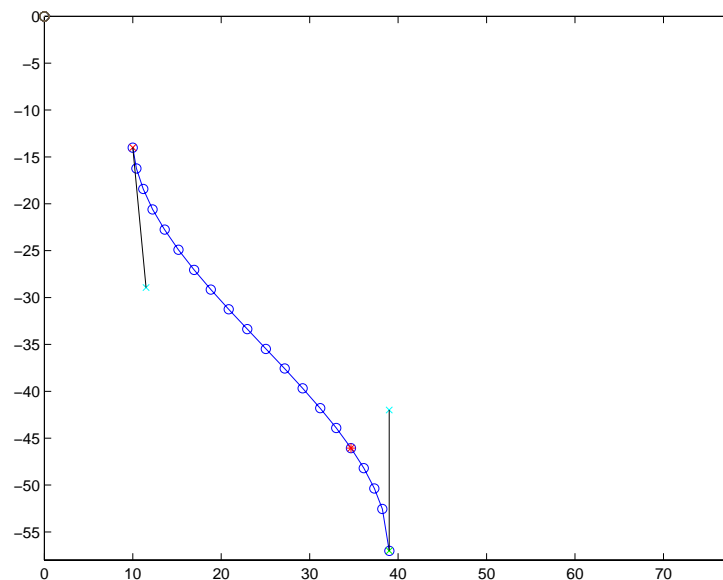


Abbildung 2.13: Kurvenapproximation, 2. Algorithmus

Radius überschreitet, wird die Strecke durch eine Gerade approximiert. Der Winkel der Gerade ist nun auch der Winkel der Steuerung.

Es wurde festgestellt, dass das Auto bei schnellerer Fahrt schlechter der Linie folgt. Der Grund dafür ist, dass die Kamera eine grosse Verzögerung hat. Bis das geschossene Bild verarbeitet werden kann, dauert es zirka eine halbe Sekunde.

2.3.3 3. Algorithmus

Um die Verzögerung wett zu machen, wurde versucht, anhand der letzten drei eingestellten Winkel die Position und Richtung des Autos zu schätzen. Die Gewichtung der Hilfsgeraden der Bezierfunktion wurde nun konstant gehalten, da das Auto einen minimalen Radius besitzt und nicht 90° Kurven fahren kann. Dies ermöglicht auch eine einfachere Funktion.

Die Funktion dieser Schätzung ist abhängig von der Geschwindigkeit des Autos. Ist der Akku fast leer, ist die Schätzung weit daneben! Dazu kommt noch der Reibungswiderstand, welcher sich ändert wenn gesteuert wird.

Falls das Auto nicht allzu schnell fährt, genügt der Algorithmus 2. Ist das Auto etwas schneller unterwegs, wird wegen der Verzögerung der Kamera das Auto von der Linie abweichen. Hier wäre der Algorithmus 3 mit der Positionsschätzung sinnvoll.

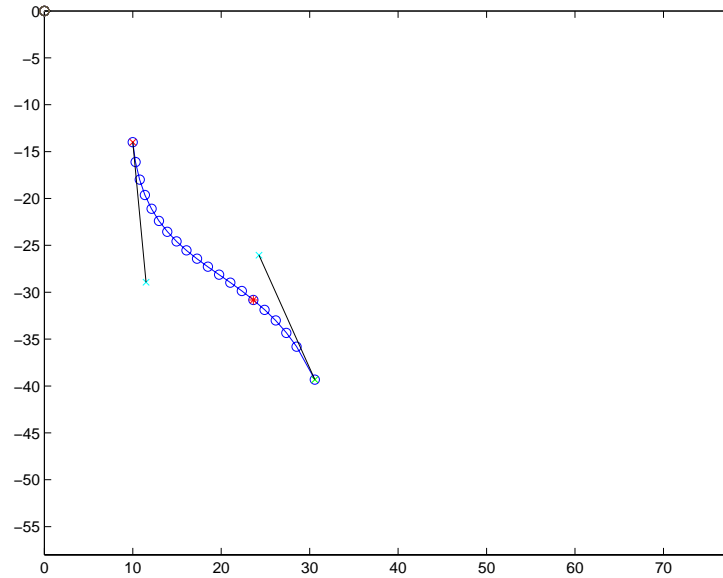


Abbildung 2.14: Kurvenapproximation, 3. Algorithmus

2.3.4 Matlab Code (3. Algorithmus)

Funktionskopf:

```
function Winkel=frog(Offset,Steigung,N,G,r,r2,w1,w2,w3);
% Offset: Offset der Gerade
% Steigung: Steigung der Gerade
% N: Anzahl Schritte der Bezier-Funktion
% G: Gewichtung der Bezier-Funktion in Pixel
% r: Aktionsradius des Autos
% r2: Radius für die Winkelberechnung
% wn: die letzten drei Radeinstellungen in°
```

Startpunkt:

```
%X-Verschiebungsfaktor
x=0.5*sin(w1/180*pi)+0.3*sin((w1+w2)/180*pi)+0.2*sin((w1+w2+w3)/180*pi);
%Y-Verschiebungsfaktor
y=0.5*cos(w1/180*pi)+0.3*cos((w1+w2)/180*pi)+0.2*cos((w1+w2+w3)/180*pi);
%ãGeschätzter Startpunkt
A=[(39 + r * x) (57 - r * y)];
```

Zuerst wird die Verschiebung in x- und y- Richtung berechnet und anschliessend mit dem Aktionsradius gewichtet und dem Nullpunkt dazuaddiert.

Hilfspunkt 1:

```
H1=[(A(1) + G * x) (A(2) - G * y)]; %Hilfspunkt1
```

Der erste Hilfspunkt liegt in der Verlängerung des Autos. Somit muss man noch die Gerade durch den Startpunkt um die Gewichtung verlängern.

Zielpunkt:

```
B=[Offset 14]; %Zielpunkt
```

Der Zielpunkt befindet sich nicht ganz am oberen Bildrand

Hilfspunkt 2:

```
w=-atan(Steigung); %Winkel der gefundenen geraden
x=sin(w); %X-Verschiebungsfaktor
y=cos(w); %Y-Verschiebungsfaktor
H2=[(Offset - G * x) (14 + G * y)]; %Hilfspunkt2
```

Zuerst wird der Winkel der gefundenen Gerade anhand der Steigung berechnet. Danach wird mit dem Winkel die y- bzw. x- Verschiebung berechnen und anschliessend mit der Gewichtungsstrecke multipliziert und dem Zielpunkt addiert bzw. subtrahiert.

Die Bezier- Kurve:

```
%Bezier-Kurve initialisierung
x=zeros(N+1); %X-Array
x(1)=A(1); x(N+1)=B(1);
y=zeros(N+1); %Y-Array
y(1)=A(2); y(N+1)=B(2);
found=0; %Falls der erste Punkt der Kurve ausserhalb des Radius ist, wird found 1
%Bezier-Kurve berechnung
for i = 2:1:N,
    t=i*1.0/(N);
    x(i)=A(1)*(1-t)^3 + H1(1)*3*(1-t)^2*t + H2(1)*3*(1-t)*t^2 + B(1)*t^3;
    y(i)=A(2)*(1-t)^3 + H1(2)*3*(1-t)^2*t + H2(2)*3*(1-t)*t^2 + B(2)*t^3;
    if (sqrt((x(i)-A(1))^2+(y(i)-A(2))^2)>r2 && found==0)
        found=1;
        P=[x(i) y(i)];
    end
end
```

Zuerst werden mal die Arrays initialisiert und der Startpunkt und Zielpunkt hineingeschrieben. Während dem berechnen der Kurve wird noch geprüft, ob der Punkt ausserhalb des Aktionsradius des Autos ist. Dafür wird die Variable "found" benützt. In der Schleife wird nun Punkt für Punkt die Bezierkurve⁷ berechnet und im Array gespeichert.

Grafische Darstellung:

```
%Grafische Darstellung
plot(x,-y,'-o'); %Bezier-Kurve
hold on;
plot(A(1),-A(2),'gx'); %Startpunkt
plot(B(1),-B(2),'rx'); %Zielpunkt
plot(H1(1),-H1(2),'cx'); %Hilfspunkt1
plot(H2(1),-H2(2),'cx'); %Hilfspunkt2
plot([A(1) H1(1)],[-A(2) -H1(2)],'k'); %Gewichtungsvektor1
plot([H2(1) B(1)],[-H2(2) -B(2)],'k'); %Gewichtungsvektor2
plot(P(1),-P(2),'r*'); %Punkt ausserhalb des Radius
hold off;
axis([0 78 -58 0]); %Skalierung
```

Die y- Werte werden negativ eingesetzt, da das Bild sonst im Matlab auf dem Kopf stehen würde. In C++ ist die Koordinate (1/1) oben links.

Die Winkelberechnung:

```
Winkel=atan((39-P(1))/(57-P(2)))*180/pi+w1+w2+w3; %Winkel in Grad
```

⁷<http://www.math.ucla.edu/~baker/java/hoefer/Bezier.htm> (Stand 11. Juli 2003)

Wie schon oben angetönt, wird die Bezier- Kurve zwischen dem ersten Punkt ausserhalb des Aktionsradius des Autos und dem Startpunkt mit einer Geraden approximiert. Anhand dieser Geraden wird nun der Winkel berechnet.

2.4 Bestandteile des Hauptprogramms

In den letzten Unterkapiteln wurde hauptsächlich die Bildverarbeitung erläutert. Im Folgendem werden noch die Funktionen im Hauptprogramm kurz vorgestellt. Auf eine ausführliche Dokumentation dieser Funktionen wird verzichtet, da der Code selbst schon kommentiert ist. Aber es sollen noch einige interessante Details aufgelistet werden.

2.4.1 Initialisierung

Bei der Initialisierung werden die ActiveX-Komponente und das Socket-Interface initialisiert. Wir können das Bild nicht über eine ActiveX-Funktion in den Speicher bringen, darum extrahieren wir das Bild aus dem Bildschirm und speichern es dann als BITMAP. Wenn jetzt ein anderes Fenster, z.B. der Windows Internet Explorer das Bild von der Kamera teilweise verdeckt, dann ist die Kante des Internet Explorers im Kantenbild sichtbar und die Bildverarbeitungssoftware wird stark gestört. Abhilfe schafft die Zeile `SetWindowPos()`. Sie bewirkt, dass unser Programmdialog immer im Vordergrund ist und somit durch kein anderes Programmfenster überdeckt werden kann.

```

BOOL CMY80211CarDlg::OnInitDialog(){
    ...
    SetWindowPos(&this->wndTopMost, 0, 0, 0, 0, SWP_NOMOVE|SWP_NOSIZE);

    // Run the ActiveX component
    m_xplug.SetRemoteHost("192.168.10.200");
    m_xplug.SetRemotePort(8481);
    m_xplug.SetTimeout(5);
    m_xplug.SetAuthType(1);
    m_xplug.SetPreviewFrameRate(1);
    m_xplug.SetPreviewWidth(320);
    m_xplug.SetPreviewHeight(240);
    m_xplug.SetDeviceSerialNo("");
    m_xplug.Play();

    // Initialize our socket
    WSADATA wsaData;
    WSASStartup(MAKEWORD(1,1), &wsaData);

    mySocket = socket(AF_INET, SOCK_DGRAM, 0);

    addrSrv.sin_family = AF_INET;
    addrSrv.sin_addr.s_addr = inet_addr("192.168.10.20");
    addrSrv.sin_port = htons(3000);
    ...
}

```

Listing 2.9: Code-Ausschnitt aus der Member-Funktion `OnInitDialog()`

2.4.2 Senden von UDP Paketen

Die Member-Funktion `sendUDP()` sendet UDP Pakete zum Auto. Mit der Variable `speedPacket` wird gesagt, das jede n-te Paket die Geschwindigkeitsinformation enthält, sonst werden immer Richtungsinformationen gesendet.

Es wird nicht weiter auf diese Funktion eingegangen, da sie selbsterklärend ist.

2.4.3 Anzeigen des Kantenbildes

Diese Member-Funktion zeigt das Kantenbild unterhalb des Originalbilds an. Wichtig dabei ist, dass in der letzten Zeile der Befehl `DeleteObject(hPicture)` aufgerufen wird; diese Anweisung gibt den durch die `LoadImage()`-Funktion reservierten Speicher wieder frei. Vergisst man diese Anweisung, so entsteht ein Speicherleck, welches man während dem Betrieb mit dem Task Manager beobachten kann.

```

void CMY80211CarDlg::displayEdgePicture( LPTSTR pFile, CMY80211CarDlg* pDlg ){
    CClientDC hScreenDC( pDlg );
    CDC dc;
    dc.CreateCompatibleDC( &hScreenDC );
}

```

```

HANDLE hPicture = LoadImage( NULL, pFile, IMAGE_BITMAP, 320, 240, LR_LOADFROMFILE );

HBITMAP* myPicture = ( HBITMAP* ) hPicture;
dc.SelectObject( myPicture );
hScreenDC.BitBlt( 11, 260, 320, 240, &dc, 0, 0, SRCCOPY );

DeleteObject( hPicture );
}

```

Listing 2.10: Member-Funktion displayEdgePicture(...)

2.4.4 Extrahieren des Bildes aus dem Bildschirm

Es musste eine Funktion geschrieben werden, welche das von der ActiveX-Komponente dargestellte Bild aus dem Bildschirm extrahierte, weil in der kurzen Zeit keine Lösung gefunden wurde um direkt an die Bilddaten zu gelangen.

```

void CMy80211CarDlg::extractPicture( LPTSTR pFile, CMy80211CarDlg* pDlg ) {
    CClientDC hScreenDC( pDlg );
    CDC hmemDC;
    hmemDC.CreateCompatibleDC( &hScreenDC );

    HBITMAP hmemBM = CreateCompatibleBitmap( hScreenDC, 320, 240 );
    SelectObject( hmemDC, hmemBM );

    BitBlt( hmemDC, 0, 0, 320, 240, hScreenDC, 11, 11, SRCCOPY );

    HGLOBAL hpxldata = GlobalAlloc( GMEM_FIXED, 320*240*3 );
    void FAR* lpvpxldata = GlobalLock( hpxldata );

    BITMAPINFO bmiInfo;
    bmiInfo.bmiHeader.biSize = 40;
    bmiInfo.bmiHeader.biWidth = 320;
    bmiInfo.bmiHeader.biHeight = 240;
    bmiInfo.bmiHeader.biPlanes = 1;
    bmiInfo.bmiHeader.biBitCount = 24;
    bmiInfo.bmiHeader.biCompression = 0;
    bmiInfo.bmiHeader.biSizeImage = 0;
    bmiInfo.bmiHeader.biXPelsPerMeter = 0;
    bmiInfo.bmiHeader.biYPelsPerMeter = 0;
    bmiInfo.bmiHeader.biClrUsed = 0;
    bmiInfo.bmiHeader.biClrImportant = 0;

    BITMAPFILEHEADER bmFileHeader;
    bmFileHeader.bfType = 19778;
    bmFileHeader.bfSize = (320*240*3)+40+14;
    bmFileHeader.bfReserved1 = 0;
    bmFileHeader.bfReserved2 = 0;
    bmFileHeader.bfOffBits = 54;

    GetDIBits( hmemDC, hmemBM, 0, 240, lpvpxldata, &bmiInfo, DIB_RGB_COLORS );

    CFile file;
    file.Open( pFile, CFile::modeWrite|CFile::modeCreate );

    file.Write( &bmFileHeader, sizeof(bmFileHeader) );
    file.Write( &bmiInfo, sizeof(bmiInfo) );
    file.Write( lpvpxldata, 320*240*3 );

    file.Close();
    GlobalUnlock( hpxldata );
    GlobalFree( hpxldata );
    DeleteObject( hmemBM );
    DeleteDC( hmemDC );
    ReleaseDC( &hScreenDC );
}

```

Listing 2.11: Member-Funktion extractPicture(...)

Listing 2.11 zeigt die Funktion. Eine der wichtigen Anweisungen ist `BitBlt(hmemDC, 0, 0, 320, 240, hScreenDC, 11, 11, SRCCOPY)`. Diese Anweisung bewirkt, dass der Inhalt des Bildschirms, auf den durch das `CClientDC hScreenDC` zugegriffen wird, in einen kompatiblen DC, in unserem Fall `hmemDC`, transferiert wird.

Es wird auch globaler Speicher alloziert, das rührt daher, dass die ganze Bildextraktion in einem Thread geschieht (Thread ist global). Danach werden die BITMAP Headers initialisiert und das Bild geschrieben. Am Schluss muss der ganze allozierte Speicher freigegeben werden, sonst gibt es einen Memory-Leak.

3 PIC Board für die Cardbus-Wirelesskarte

Das Board "ER21" ist eine komplette Lösung für eine Schnittstelle im Wireless- Netzwerk. Es besteht aus einer Cardbus- Wirelesskarte (nur 5V Typen), einem PIC18F451, einem RJ45 Netzwerkanschluss und einem seriellen Port.

Auf dem PIC ist bereits eine funktionsfähige Software installiert, die verschiedene Funktionen beinhaltet:

- Webserver zur Überprüfung der Funktion und zur Konfiguration des Boardes.
- Timeserver
- Echoserver
- ... und noch vieles mehr.

Die Software des PIC's besteht aus der Hauptdatei *p18web.c* und mehreren Include-Dateien.

3.1 Zusammenfassung der Änderungen

- In der Datei *p18web.c* wurde die Baudrate für die serielle Kommunikation von 9600 auf 2048 gesetzt.
- In der Datei *p18_ser.c* wurde ein Delay beim senden der Daten über die Serielle Schnittstelle eingefügt, damit die Bytes einzeln gesendet werden und nicht aneinander.
- In der Datei *p18_udp.c* wurde der Frog-Server eingefügt.

3.1.1 Baudrate

Es musste ein zweiter PIC eingesetzt werden, da beim Board mit der Wirelesskarte nur ein Port für die Pulsweitenmodulation frei war. Beim Board war schon ein RS232 Stecker vorhanden und es wurde entschieden, diesen für die Kommunikation zwischen den PIC's zu nutzen. Da die zwei PIC's nicht die gleiche Taktfrequenz haben, ergeben sich für die Baudrate jeweils verschiedene Werte:

$$\begin{aligned} \text{Baud Rate} &= F_{osc}/(16(N + 1)) && \text{für } BRGH = 1 \quad (\text{High speed}) \\ \text{Baud Rate} &= F_{osc}/(64(N + 1)) && \text{für } BRGH = 0 \quad (\text{Low speed}) \end{aligned}$$

Für den PIC18, der mit 20MHz läuft, wurde BRGH = 0 und für den PIC16, der mit 32,768 kHz läuft, BRGH = 1 gewählt. Somit ergibt es folgende mögliche, gemeinsame Baudraten:

PIC18		PIC16	
Baud Rate	N	Baud Rate	N
2056	151		
2042	152	2048	0
2029	153		

Die Einstellung der Baudrate erfolgt in der Datei *p18web.c* in Zeile 202: `#define SER_BAUD 2048`
Falls der PIC neu programmiert wird, muss er über die serielle Schnittstelle neu konfiguriert werden. Dazu muss der Baustein MAX232 auf dem Board eingesetzt werden. Da das Windows-Terminal die Baudrate von 2048 nicht unterstützt, kann nun mit dem Realterm die gewünschten Einstellungen vorgenommen werden.

3.1.2 Verzögerung

Die Rechnung für die Baudrate ist nicht ganz korrekt, denn die Zahlen gehen nicht ganz auf und somit gibt es Synchronisationsprobleme, welche mit der Zeitverzögerung von 10ms wettgemacht werden. In der Datei *p18_ser.c* Zeile 39 wurde die Verzögerung implementiert:

```
// Send a string to the serial port
void ser_puts(char *str) {
    char c;

    while ((c = *str++) != 0)
    {
        serial_putchar(c);
        //insert by rguldene
        delay_ms(10);
        //end insert
    }
}
```

Listing 3.1: Code-Ausschnitt *p18_ser.c*

Die Zeitverzögerung sorgt dafür, dass jedes einzelne Byte einzeln gesendet wird und somit der Empfänger jeweils neu synchronisiert wird. Ohne diese Verzögerung sendet der PIC alle vier Bytes aneinander, was dazu führt, dass der zweite PIC das dritte Byte nicht versteht.

3.1.3 UDP to SERIAL

Die Steuerdaten werden via UDP- Pakete dem PIC gesendet. Die Daten müssen dann an den seriellen Port gelangen. In der Datei *p18_udp.c* werden die empfangenen Pakete entsprechend behandelt. Der Code wurde so erweitert, dass alle Pakete, die auf den Port 3000 empfangen werden, direkt an die serielle Schnittstelle weitergeleitet werden.

```
/* Receive an incoming UDP datagram, return 0 if invalid */
BOOL udp_recv(void){
    ...
    if(locport == ECHOPORT)           // Echo: return copy of data
    {
        setpos_txin(UDPIPHDR_LEN);
        copy_rx_tx(udplen - UDPHDR_LEN);
        udp_xmit();
        DEBUG_PUTC('U');
    }
    else if(locport == DAYPORT)       // Daytime: return string
    {
        print_lcd = print_serial = FALSE;
        print_net = TRUE;
        setpos_txin(UDPIPHDR_LEN);
        putstr(DAYMSG);
        udp_xmit();
    }
}

#if INCLUDE_DHCP
    else if(locport == DHCPCLIENT_PORT) // DHCP client
        dhcp_handler();
#endif

#if INCLUDE_TIME
    else if(locport == TIMECLIENT_PORT) // Time client
        time_handler();
#endif

// new included from r.guldener
    else if(locport == FROG_PORT)       // FROG client
    {
        BYTE received;
        while(get_byte(received))
        {
            ser_puts(received);
        }
    }
// end included r.guldener
    ...
}
```

Listing 3.2: Code-Ausschnitt *p18_udp.c*

4 Ansteuerung der Servos

Abbildung 4.1 zeigt das Signal zum Ansteuern der Servos. Die Spannung beträgt +5V und die Periodendauer der Pulsweitenmodulation ist 20ms. Der Servo ist in der Neutralstellung, wenn ein Puls der Länge 1.5ms ansteht. Weicht der Puls um $\pm 0.9\text{ms}$ ab, fährt der Servo um 90° nach links oder rechts.

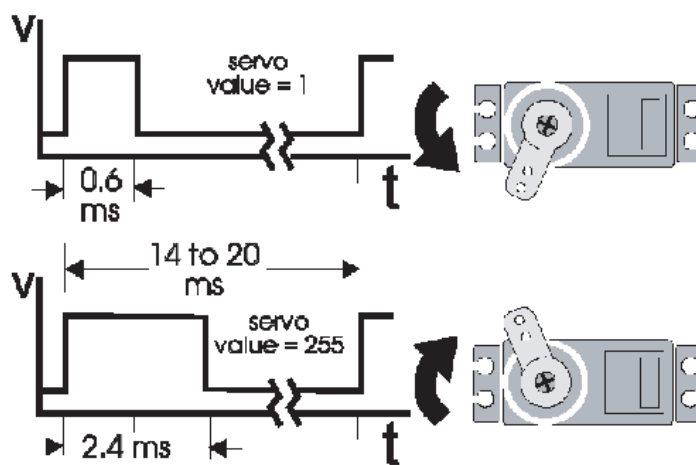


Abbildung 4.1: Ansteuerung der Servomotoren

4.1 Taktfrequenz und Pulsweitenmodulation

Die Periode der Pulsweitenmodulation ist abhängig von der Taktfrequenz des PIC's. Somit kann nicht jeder beliebige Quarz eingesetzt werden. Die folgende Rechnung zeigt, welche Quarze eingesetzt werden können:

$$\text{PWM periode} = \frac{1}{f_{osc}} \cdot (\text{Periode} + 1) \cdot 4 \cdot \text{T2_DIV}$$

bzw.

$$f_{osc} = \frac{1}{\text{PWM periode}} \cdot (\text{Periode} + 1) \cdot 4 \cdot \text{T2_DIV}$$

Periode: 0 ... 255

T2_DIV: 1, 4, 16 (Prescaler)

Für die Periode von 20ms ergibt das folgende Resultate:

$$f_{osc} = \frac{1}{0.02} \cdot (0 + 1) \cdot 4 \cdot 1 = 200\text{Hz}$$

$$f_{osc} = \frac{1}{0.02} \cdot (255 + 1) \cdot 4 \cdot 16 = 819.2\text{kHz}$$

Die Änderung der Pulsbreitenmodulation ist je nach Taktfrequenz unterschiedlich:

$$\Delta t = \frac{1}{f_{osc}} \cdot T2_DIV$$

$$\Delta t = \frac{1}{200Hz} \cdot 1 = 5ms$$

$$\Delta t = \frac{1}{800kHz} \cdot 1 = 20\mu s$$

Bei 800kHz ist die Änderung 20 μ s. Dazwischen pendelt die Änderung hin und her. Der springende Punkt ist jeweils das Umschalten des Teilers T2_DIV.

Es wird ein Uhrenquarz mit der Frequenz 32768 Hz benutzt, dies ergibt eine Pulsänderung von 30 μ s.

Es wurde später festgestellt, dass der PIC zu langsam ist. Die Kommunikation mit der seriellen Schnittstelle stockte. Als der PIC nach der Behebung dieses Problems einwandfrei funktionierte, wurde nichts mehr Neues in den PIC implementiert, weil neue Funktionen wieder Ressourcen belegen würden und dann die serielle Kommunikation wieder nicht funktionieren würde. Eine Abhilfe wäre in diesem Fall eine höhere Taktrate, ein 1-MHz-Quarz wäre für den Einbau durchaus geeignet. Durch den neuen Quarz wäre die Periode der Pulsweitenmodulation etwas kürzer als vorgeschrieben, aber gemäss Abbildung 4.1 sollte das der Servo verkraften können; die Periodendauer wäre in diesem Fall 16.4ms.

Nachfolgend noch die Implementation:

```

setup_ccp2(CCP_PWM); // Configure CCP2 as a PWM
setup_ccp1(CCP_PWM); // Configure CCP1 as a PWM

// The cycle time will be (1/clock)*4*t2div*(period+1)
// In this program clock=32768 and period=163 (below)
// For the three possible selections the cycle time is:
// (1/32768)*4*1*164 = 20 ms or 50 hz

setup_timer_2(T2_DIV_BY_1, 163, 1);

value=49; // Neutralstellung

set_pwm2_duty(value);
set_pwm1_duty(value);

```

Zuerst wird der CPP Ausgang als PWM konfiguriert. Danach muss noch der TIMER2 eingestellt werden und T2_DIV ist der Teiler des Timers. Danach kommt der Wert, wo der Timer ein Overflow auslöst. Mit der letzten Zahl kann der Interrupt des Timers verzögert werden. Nun kann die Pulsdauer fortlaufend verändert werden.

4.2 RS232 Schnittstelle

Die Berechnung der Baudrate wurde schon ausführlich beim PIC- Board erläutert. Für die Konfiguration der Schnittstelle gibt es vordefinierte Routinen:

```

#use delay(clock=32768)
#use rs232(baud=2048, xmit=PIN_C6, rcv=PIN_C7)

```

#use delay wird von der RS232 benutzt und muss zuerst aufgerufen werden. Anschliessend kann die Schnittstelle mit der Baudrate, dem Sendepin und dem Empfangspin eingestellt werden. Mit PUTC() kann man nun einen Charakter senden.

Damit beim Empfänger keine Daten verloren gehen, werden die ankommenden Bytes mittels einer Interruptroutine in einen Zwischenspeicher geschrieben:

```

//Daten von der seriellen Schnittstelle lesen
#int_rda void serial_isr() {
    int t;

    buffer[next_in]=getc();
    t=next_in;
    next_in=(next_in+1) % BUFFER_SIZE;
    if(next_in==next_out)
        next_in=t; // Buffer full !!
}

```

```
}

```

Um es später einfach zu machen, wird ein Bit verknüpft um zu Prüfen, ob Daten im Speicher sind:

```
#define bkbhit (next_in!=next_out)
```

Die Daten müssen auch aus dem Speicher gelesen werden. Dies macht die folgende Funktion:

```
byte bgetc() {
    byte c;

    while(!bkbhit) ;
    c=buffer[next_out];
    next_out=(next_out+1) % BUFFER_SIZE;
    return(c);
}
```

Jetzt muss nur noch der Interrupt eingeschaltet werden:

```
enable_interrupts(global);
enable_interrupts(int_rda);
```

4.3 Hauptprogramm

Der PIC bekommt jeweils vier Bytes, z.B. **H+12** oder **V+18**. Der erste Parameter ist für die Unterscheidung der Servos. **H** bedeutet Horizontal und meint den Steuerservo. **V** wäre für Vertikal und meint den Fahrtenregler. Der zweite Parameter bestimmt das Vorzeichen. Die letzten zwei Werte repräsentieren den Wert selbst.

Somit können Werte von $\pm 90^\circ$ für die Steuerung und $\pm 99\%$ für die Geschwindigkeit übermittelt werden. Beim obigen Beispiel bedeutet dies, dass das Auto 12° nach links einlenken soll.

Falls mit der Übertragung der Bytes etwas nicht in Ordnung ist, erkennt dies der PIC mit grosser Wahrscheinlichkeit:

```
while(true)
{
    if (bkbhit) //Falls Daten im Empfangsbuffer
    {
        temp=bgetc();
        putc(temp);
        switch (recvcounter)
        {
            case 4:
                if ((temp=='H') || (temp=='V'))
                {
                    axes=temp;
                }
                break;
            case 3:
                if (temp=='+')
                    oper = false;
                else if (temp=='-')
                    oper=true;
                else recvcounter=0;
                break;
            case 2:
                if ((temp <= 57) && (temp >= 48))
                    value=(temp-48)*10;
                else recvcounter=0;
                break;
            case 1:
                if ((temp <= 57) && (temp >= 48))
                {
                    value=value+(temp-48);
                    set_pwm(axes,value,oper);
                }
                else recvcounter=0;
                break;
        }
        if (recvcounter <= 0) recvcounter=4;
        else if (--recvcounter==0) recvcounter=4;
    }
}
```

Das Programm läuft in einer Endlosschleife und sobald etwas neues im Speicher der seriellen Schnittstelle steht (`bkbhit=true`), wird das empfangene Byte untersucht.

Damit der PIC weiss, welches Byte er als nächstes empfangen sollte, wird jeweils ein Zähler von vier herunter gezählt. Je nach Zählerstand wird das Byte nach dem Inhalt überprüft. Kommt mal etwas Unerwartetes an, wird der Zähler zurückgesetzt. Kommen alle Bytes an, wird die Funktion `set_pwm` aufgerufen:

```
//setzen der PWM Pulsweite
void set_pwm(byte axes, long value, boolean oper) {
    long wert;
    restart_wdt();

    // Speed limit
    if(value>40) value=40;

    switch (axes)
    {
        case 'H': if (oper) value=value+91; else value=91-value;
                  wert=H[value];
                  set_pwm1_duty(wert);
                  break;
        case 'V': if (!oper) value=value+100; else value=100-value;
                  wert=V[value];
                  set_pwm2_duty(wert);
                  break;
    }
}
```

Als erstes wird der Watchdog-Timer zurückgesetzt. Mehr dazu gleich im nächsten Abschnitt. Damit das Auto nicht mit grösster Geschwindigkeit herumfährt und sich beschädigen kann, wurde der maximale Wert auf $\pm 40\%$ fixiert. Danach wird zwischen den Servos unterschieden, denn die Übergabewerte müssen auf die Pulsdauer der Pulsweitenmodulation angepasst werden. Dies geschieht mit einer Lookup-Table. Da der Index der Tabelle immer positiv ist, muss noch ein Offset dazugerechnet werden. Hat man den Wert geholt, wird die Pulsbreite gesetzt.

4.4 Automatische Abschaltung mittels Watchdog-Timer

Da das Auto viel Elektronik mit sich führt, sind dementsprechend viele Störungsquellen vorhanden. Eine trickreiche Störungsquelle ist, wenn der Datenstrom abbricht und das Auto keine Befehle mehr erhält. Entweder ist der PC abgestürzt oder die Akkus sind leer. Der Watchdog-Timer sorgt dafür, dass der PIC immer wieder zurückgesetzt wird und beim Neustart die Servos wieder in die Neutralstellung setzt. Natürlich ist der Reset nach einem erfolgreichen Empfangen eines Datenpaketes nicht erwünscht. Darum wird in der Funktion `set_pwm` der Timer immer zurückgesetzt.

Mit `setup_counters` kann der Watchdog-Timer initialisiert werden. Hier wird der Timer nach 1152ms einen Reset durchführen:

```
setup_counters(RTCC_INTERNAL, WDT_1152MS);
```

Mit `restart_wdt()` wird der Zähler des Watchdog-Timers zurückgesetzt:

```
restart_wdt();
```

Bei den Testfahrten wurde festgestellt, dass als erstes Modul die Kamera aussteigt und dann keine Bilder mehr sendet.

4.5 Die Verbindung der zwei PIC's

Der PIC für die Steuerung der Servos wird mit dem RS232 Stecker des Boards verbunden. Zu beachten ist, dass der IC MAX232 auf dem Board des PIC18 entfernt und mit Drahtbrücken ersetzt wird. Die Drahtbrücken werden wie folgt gesetzt: P7-P10, P8-P9, P11-P14 und P12-P13.

5 Bedienungsanleitung

5.1 Vorbereitung

5.1.1 Kurs abkleben

- Ziehen Sie mit einem Klebeband einen Rundkurs

Die Bilderkennung zeigt, dass eine Linienbreite von ca. einem Zentimeter gut zu erkennen ist. Ein weisses mattes Klebeband eignet sich sehr gut dazu. Das Auto hat natürlich auch einen maximalen Einschlag. Somit darf der Radius der Strecke nicht kleiner als 2m sein.

5.1.2 ActiveX-Komponente XPLUG.OCX

Es muss noch die ActiveX-Komponente für die Kamera installiert werden.

5.1.3 Installation der Wireless-Karte

- Installieren Sie eine Wireless- Karte und konfigurieren Sie sie wie folgt:
 - Mode: Ad-Hoc
 - IP: 192.168.10.X (X kann zwischen 1 und 254 sein, aber ohne 20 und 200, diese werden von der Kamera und dem PIC benutzt)
 - Subnet: 255.255.255.0

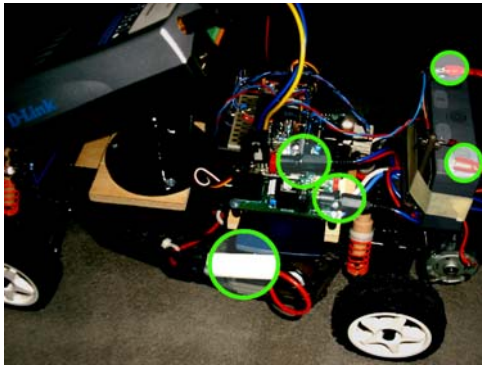
Im Paket befindet sich eine Wireless-Cardbus Karte. Installieren Sie diese mit der zugehörigen Treiber- software von der CD des Herstellers.

5.1.4 Aufladen der Akkus

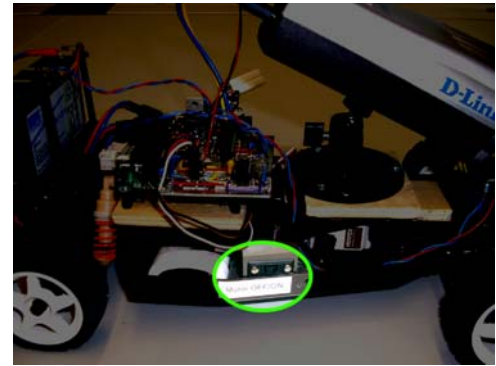
- Trennen Sie den Nickel- Cadmium Akku beim weissen Stecker und verbinden Sie ihn mit dem entsprechenden Ladegerät. Stecken Sie das Netzteil erst jetzt in die Steckdose. Die Ladezeit beim leeren Akku beträgt 10 Stunden. Die rote LED zeigt an, dass geladen wird. Falls die LED am Steckernetzteil nicht leuchtet, überprüfen Sie die Steckverbindung.
- Entfernen Sie die beiden Kabel beim Bleiakku auf dem Heck des Autos und schliessen Sie das Ladegerät 6Bp an. Achten Sie auf die Polarisation! Stecken Sie das Netzteil erst jetzt in die Steckdose. Die Ladezeit beträgt 2.1 Stunden. Das Ladegerät besitzt eine LED. Diese erlischt, wenn der Akku geladen ist.

Wichtig:

- Trennen Sie die Akkus vom Ladegerät nach der Ladezeit!
- Um den Memory- Effekt zu verhindern, sollten die Ni-Cd Akkus vor dem Laden leer sein.



(a)



(b)

Abbildung 5.1: Starten des Autos

5.2 Starten des Autos

- Akkus an das Autos anschliessen, die zwei Rundstecker in die Buchse stecken und den Schalter des Motors auf "ON" schalten (Abbildung 5.1).
- Das Programm Auto starten.
- Mit dem Knopf "START" bzw. "STOP" wird die Bildübermittlung und die Auswertung gestartet oder angehalten.

5.3 Abschalten des Autos

- Das Programm Auto beenden.
- Die Rundstecker entfernen und den Schalter des Motors auf "OFF" schalten (Abbildung 5.1).

5.4 Fehlerbehebung

- Das Programm zeigt kein Bild:
Überprüfen Sie die LED's bei der Kamera. Rechts neben der Linse muss die blaue LED kontinuierlich leuchten, ansonsten ist der Bleiakku auf dem Heck des Autos leer. Laden Sie den Akku.
- Das Auto reagiert nicht:
Überprüfen Sie alle Stecker und insbesondere der Schalter seitlich am Autoboden. Überprüfen sie auch den Ni-Cd Akku im Auto und laden sie ihn, falls nötig, auf.
- Das Programm stürzt ab (runtime error):
Das Programm kann keine Verbindung zur Kamera herstellen. Überprüfen Sie die Einstellung ihrer Cardbus Karte und ob die Kamera mit genügend Spannung versorgt wird (blaue LED an der Kamera muss kontinuierlich leuchten).

5.5 Software Kompatibilität

Dieser Abschnitt betrifft vor allem die Bildverarbeitungssoftware. Die Bildverarbeitungssoftware funktioniert nur auf dem Computer, auf welchem die Software geschrieben wurde. Wie schon erwähnt, wurde dabei eine spezielle Library benutzt (CVM Class Library). Diese Library verursachte auf dem PC keinerlei Probleme, aber zeigte auf anderen Computern ein anderes Verhalten. Es wurde versucht, das Programm zu debuggen, aber das brachte keine Lösung.

Unter anderem wird in der Bildverarbeitungssoftware die Funktion `indofmin()`¹ von der CVM Class Library verwendet. Es hat sich herausgestellt, dass der Aufruf dieser Funktion auf den anderen Labor-Computern das Programm zum Absturz bringt. Eine Lösung für dieses Problem wurde nicht mehr gefunden, weil die Zeit nicht mehr reichte. Es wurde aber noch Kontakt mit dem Programmierer der CVM Class Library aufgenommen und das Problem wird zu einem späteren Zeitpunkt gelöst. Oder man könnte auch versuchen, die Bildverarbeitung mit einer anderen Matrix-Klasse neu zu programmieren, in diesem Fall stehen einige Optionen zur Auswahl.

Auf der Installations-CD befindet sich ein Verzeichnis mit dem Namen SYSTEM, dort sind die für die CVM Class Library benötigten DLLs und LIBs vorhanden.

Literaturverzeichnis

- [1] Hersent O., Gurle D., Petit J.-P.: IP Telephony. Packet-based multimedia communications systems. Addison Wesley 2000.
- [2] Hartmann D., Pecnik M.: Robuste Steuerung eines Remote-Controlled Car, WS02/03, Hochschule Rapperswil 2003.
Die Publikation ist auch erhältlich unter http://medialab.ch/archiv/pdf_studien_diplomarbeiten/1sa03/Robuste_Steuerung_eines_RC_Autos.pdf (Stand 11. Juli 2003)

A ChipWeb 2.17 mit MPLAB 6.x und HI-TECH Compiler 8.20PL4

Um Chipweb mit MPLAB 6.x und HI-TECH Compiler zu bearbeiten, müssen unbedingt einige Änderungen gemacht werden.

A.1 Änderungen MPLAB/HI-TECH

- Damit MPLAB mit dem Compiler funktioniert, muss ein spezielles Update des Compilers heruntergeladen werden. Die Datei ist auf der Seite des Herstellers zu finden:
<http://www.htsoft.com/software/updates/mplab/update.htm>
- Beim oben genannten Link ist auch eine Anleitung, wie MPLAB konfiguriert werden muss, damit die beiden Komponenten zusammenarbeiten.

A.2 Änderungen ChipWeb

- Damit Chipweb kompiliert werden kann und einwandfrei läuft, müssen einzelne Dateien abgeändert werden. IoSoft hat verschiedene Hinweise herausgegeben:
<http://www.iosoft.co.uk/docs/cw217notes.pdf>
<http://www.iosoft.co.uk/support.php>
- *ht_p18.h* Zeile 36

```
__CONFIG(1, OSCSDIS & HS);  
__CONFIG(2, BOREN & PWRTEN & WDTDIS & WDTPS128);  
__CONFIG(3, CCP2RC1);  
__CONFIG(4, DEBUGDIS & LVPDIS & STVREN);
```

Hinweis: BORV20 wird ab der Version 8.20PL3 nicht mehr unterstützt.
- *p18web.c* Zeile 354

```
#if INCLUDE_WLAN  
setup_adc_ports(NO_ANALOGS); // Wireless uses analog I/O pins  
reset_wlan();  
#else
```
- *p18_tcp.c*
Zeile 95 ist: `rack = 0xffff0000L + (DWORD)concount;`
Ändere zu: `rack = ((DWORD)concount << 16) + 0xffff;`
Zeile 113 ist: `if (locport == DAYPORT && (WORD)(rack >> 16) == 0)`
Ändere zu: `if (locport == DAYPORT && (WORD)rack == 0)`
- *p18_wlan.c*
Zeile 37 ist: `#define TRISA_VAL 0xC0`
Ändere zu: `#define TRISA_VAL 0xF0`
Zeile 475 ist: `count = (len + 1) >> 1;`
Ändere zu: `count = len + 1; count >>= 1;`

A.3 Änderungen MPLAB Compiler-Einstellungen

- Compilerswitch "-NOERRATA" einstellen
- Compiler: Kategorie "Output & Debug"
 - Global optimisation = 9
 - post-pass optimisation = enabled
 - check ICD option

B CVM Class Library

This C++ class library encapsulates the concepts of vector, matrix and square matrix in Euclidean space of real and complex numbers. It utilizes BLAS and LAPACK Fortran libraries in order to achieve the best numerical performance. Along with basic vector and matrix arithmetics it contains different algorithms including norms computations, elementary transformations, solving of linear systems of kind $Ax=b$ and $AX=B$, singular value decomposition, matrix rank and determinant computation, non-symmetric eigenvalue problem, LU factorization, square matrix polynoms, square matrix inversion and square matrix exponent. All these algorithms are implemented for real and complex numbers.

B.1 Warum CVM?

Es sollte doch möglich sein, eine Bildverarbeitung ohne eine spezielle Matrix-Klasse erstellen zu können? Die Antwort lautet ja, aber der folgende Code-Ausschnitt führt zu einem Runtime-Error (Abbildung B.1):

```
// testOfNothing.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"

const unsigned int rows = 240;
const unsigned int cols = 320;

int main(int argc, char* argv[]) {
    double originalPicture[rows][cols];
    double lpfPicture[rows-4][cols-4];

    return 0;
}
```

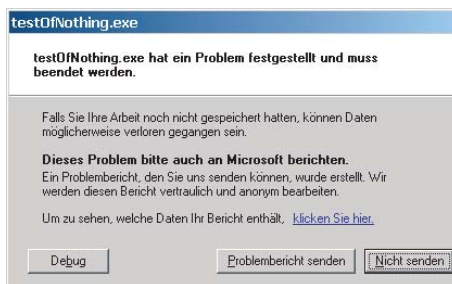


Abbildung B.1: Runtime Error

CVM besitzt unter anderem ein intelligentes Speichermanagement und solche Runtime Errors wie in Abbildung B.1 sind bis jetzt noch nicht aufgefallen.

Um sich nicht weiter mit dem Runtime Error zu beschäftigen, wurde im Internet <http://www.cvmlib.com> diese Library herunter geladen und nach einigen Kopiervorgängen von Header- und DLL-Dateien war diese Library im System integriert und lief einwandfrei.

B.2 Installation

Es ist in diesem Sinne keine eigentliche Installation, es müssen lediglich einige Dateien an die richtige Stelle kopiert werden.

B.2.1 Voraussetzungen

- Windows 2000/XP
- Microsoft Visual C++ 6.0 (Enterprise Edition)

B.2.2 Kopieren der Dateien

- **cvm.dll** und **mkl_support.dll** nach **C:\WINDOWS**
- **cvm.h** und **blas.h** in das Installationsverzeichnis von MS Visual C++ **C:\...\Microsoft Visual Studio\VC98\Include**
- **cvm.lib** in das Installationsverzeichnis von MS Visual C++ **C:\...\Microsoft Visual Studio\VC98\Lib**

Eine andere Möglichkeit wäre, dass man die oben genannten Dateien in das Verzeichnis kopiert, wo die Dateien der Bildverarbeitungssoftware residieren.

B.3 Arbeiten mit CVM

Nachdem die Dateien in die entsprechenden Verzeichnisse kopiert wurden, kann nun in Microsoft Visual C++ ein neues Projekt erstellt werden. Will man mit der CVM Library arbeiten, dann muss an der richtigen Stelle noch `#include <cvm.h>` eingefügt werden. Anschließend nicht vergessen, diese Header-Datei unter *Project*→*Settings*→*Link* mit `cvm.lib` zu *linken*.

C Entwicklungsumgebung MPLAB

Da der PIC18F452 auf einem Bord mit einem ICD-Stecker ausgerüstet ist, muss man den PIC zum Programmieren nicht herausnehmen. Mit dem ICD2 von Microchip lässt sich der PIC einfach programmieren (Abbildung C.1).

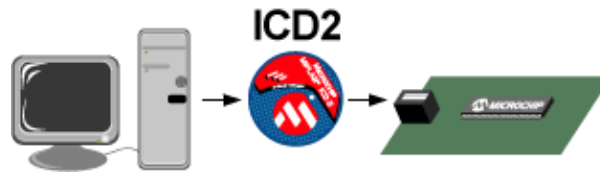


Abbildung C.1: PIC Programmierung mit ICD2

Die dazugehörige Software heisst MPLAB IDE und kann gratis von der Internetseite von Microchip herunter geladen werden. In dieser Semesterarbeit wurde die Version 6.1 benutzt.

Jetzt fehlt nur noch der Compiler. Anfangs wurde mit der Demoversion des Hitech Compilers gearbeitet. Der Weg, bis der Compiler lief, war steinig. Wir brauchten viel Zeit, bis alles lief. In Abschnitt C.1 befindet sich daher eine Anleitung, was im Sourcecode und im MPLAB IDE geändert werden muss, damit das Programm läuft.

Der Hersteller des Boardes arbeitete mit dem CCS Compiler. Somit entschieden wir uns auch diesen Compiler anzuschaffen. Die Installation dieses Compilers funktioniert dann auch komplett automatisch und man braucht kaum noch etwas einzustellen.

Der Sourcecode kann nun im MPLAB IDE editiert, kompiliert und sogleich auf den PIC gebrannt werden.

WICHTIG:

- Dem MPLAB IDE muss die Schnittstelle zum ICD2 noch angegeben werden. Unter Programmer → Settings → Communication auf USB stellen, falls der ICD2 mit dem USB Kabel verbunden ist.
- Der Programmierbereich sollte auf "Full Range" sein: Programmer → Settings → Program: Knopf "Full Range" drücken.
- Das PIC-Board sowie der ICD2 sollte mit Strom versorgt sein.
- Das Board läuft nicht, wenn das Programmierkabel noch angeschlossen ist. Die Pins für die ICD2 Schnittstelle werden auch von der Wireless-Karte benutzt! Die Karte muss aber nicht entfernt werden für das programmieren.

Der PIC16F876 wurde mit dem PICSTART PLUS (Abbildung C.2) von Microchip programmiert:

Der Nachteil war, dass dieser eine alte Firmware besass. Somit war er nicht MPLAB IDE 6.1 tauglich. Neben der neuen Version musste also die alte Version 5.3 von MPLAB IDE installiert werden. Damit der PICSTART mit der neuen Version arbeiten kann, muss ein PIC17C44 mit der neuen Firmware programmiert und in den PICSTART eingesetzt werden.

Der erworbene CCS Compiler unterstützt den PIC16 nicht. Somit musste auf den alten CCS Compiler, der nur PIC16 und PIC17 unterstützt, zurückgegriffen werden. Der Compiler muss nur nach der Anleitung in Abschnitt C.1 installiert werden. Danach kann man den Sourcecode im Programm bearbeiten, kompilieren und den PIC programmieren. Der Nachteil gegenüber dem ICD2 ist die sehr langsame Programmiergeschwindigkeit wegen der seriellen Schnittstelle und dass der PIC jedes Mal vom Board auf den PICSTART gewechselt werden muss.

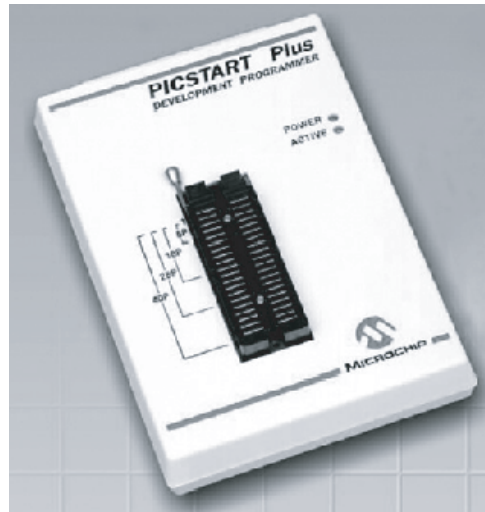


Abbildung C.2: Programmierung des PIC16F876 mit PICSTART

C.1 MPLAB Configuration for MPC

MPLAB, Microchip's IDE, can perform source-level debugging of MPC code. Currently, we support configuring MPLAB 5.x to do this: MPLAB 6.x is forthcoming.

These instructions configure MPLAB to perform source-level debugging with MPC.

C.1.1 MPC configuration

Before configuring MPLAB, ensure that the following environment variables are set (probably through your AUTOEXEC.BAT):

```
set INCLUDE=c:\mpc\include
rem substitute the path to the device
rem headers directory on your MPC installation

set LIBRARY=c:\mpc
rem substitute the path to the MPC
rem program directory, where mpc12.lib,
rem mpc14.lib, and mpc16.lib reside
```

Restart if necessary.

C.1.2 MPLAB configuration

To enable support for the Byte Craft compiler within MPLAB, do the following:

- Select *Project/Install Language Tool*. From the "Install language tool" window (Abbildung C.3):
 1. Select *Byte Craft* from the "language" drop box.
 2. Select *Assembler/C compiler* from the "Tool Name" drop box.
 3. Type in the path and filename of the MPC compiler in the "Executable" text field. Beneath the executable name, select "Command Line".
- In MPLAB, select *Project/New Project*.
- From the "New Project" dialog, select the directory and file name for the project.

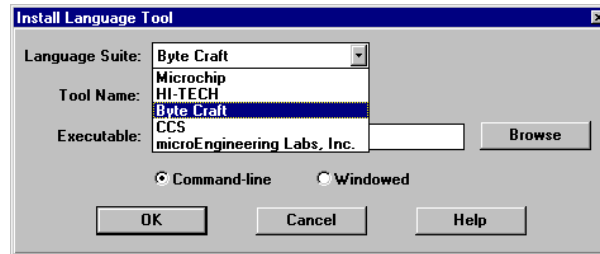


Abbildung C.3: Install Language Tool Dialog

- Within the "Edit Project" dialog (Abbildung C.4):
 - Select *Byte Craft* from the "Language tool suite" drop box.
 - Select your_filename.HEX in the "Project files" area, and select the *Node properties* button.

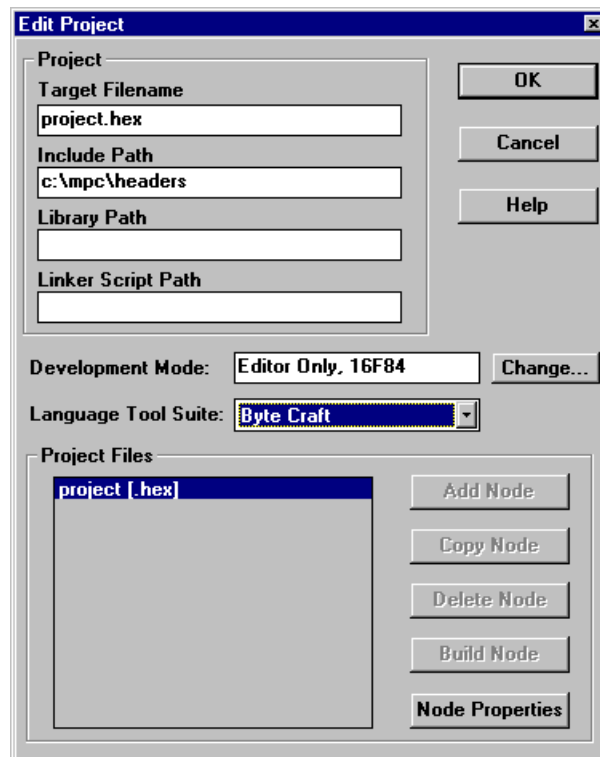


Abbildung C.4: New Project Dialog

- Within the "Node properties" window (Abbildung C.5):
 - Select *Assembler/C compiler* from the "language tool" drop box. In the "Options" area, choose the following options:
 - * Hex Format: select the *Intel* radio button.
 - * Listing file: select the *on* switch.
 - * Error file: select the *on* switch.
 - * Source type: select the *C* radio button.

You should see `+di +l +e -m` in the "command line" text field. Choose *OK*.

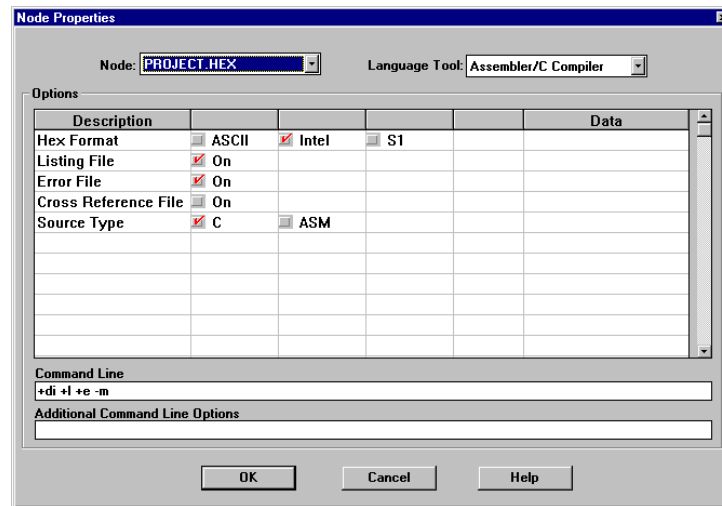


Abbildung C.5: Node Properties Dialog

- Choose *Add Node*, and add in your source file as a project node. If you do not have a source file yet, be sure to add it in whenever possible.
- Choose *OK*. The project is now configured to use Byte Craft's compiler.

Finally, place the following line at the top of your source file:

```
#pragma option +v;
```

Once you have completed this procedure, you should be able to perform C source level debugging within the MPLAB IDE.

C.1.3 DOS Box problems

If the compiler does not exit and close its DOS window, or if MPLAB reports that the compiler has timed out, one of two settings is probably incorrect.

If necessary, choose *No* in the "Build Error" dialog in MPLAB to return to editing. Use *ALT+Tab* to switch to the MPC.EXE compiler window (an MS-DOS box), and press *Control+C* to exit the compiler. To fix the problem, try the following:

1. Within MPLAB, choose *Project/Install Language Tool*.
2. Select *Byte Craft* in the "Language Suite" list.
3. Ensure that you have selected the "Command Line" option for the compiler, as mentioned above.
4. If not, select "Command Line" and click *OK*. Try compiling again.

If that does not fix the problem, do the following:

1. Open the Windows Explorer, and navigate to the MPC program directory.
2. Select the compiler executable (MPC.EXE), and right-click on it. Choose *Properties* from the shortcut menu.
3. In the Properties dialog, choose the *Program* tab.

4. If *Close on exit* is not checked, check it and click *OK*.
5. Try compiling again.

If you have problems after using these instructions, please contact support@bytecraft.com.
MPLAB is a trademark of Microchip Technology Inc.

D Listings der Bildverarbeitung

```
1 #include <windows.h>
2 #include <stdio.h>
3 #include <fstream.h>
4 #include <iostream.h>
5 #include <cvm.h>
6 #include <math.h>
7
8
9
10
11
12 // The CVM library has its own namespace
13 using namespace cvm;
14
15
16
17 // This is the "heart" of our application. It analyzes the picture and finds the line
18 // finally
19 void findTheLine();
20
21
22
23 // This function calculates the angle from the line parameters and returns it
24 // Algorithm Version 2
25 double ANGLE( double pOffset, double pRise, unsigned short pN, double pG=20, double pr=21, double pr2=20 );
26
27
28
29 // This function stores the pixel information in a matrix; pixels are converted into
30 // gray-scale
31 void BMP2MATRIX( BYTE* pBuffer, int pWidth, int pHeight, rmatrix* pPicture );
32
33
34
35 // This function writes the stored pixel-data from the matrix into a bitmap-buffer and
36 // returns a pointer to this buffer
37 BYTE* MATRIX2BMP( int pWidth, int pHeight, long* pNewSize, rmatrix* pPicture );
38
39
40
41 // This function loads a bitmap and returns a pointer to the bitmap-buffer
42 BYTE* LoadBMP( int* pWidth, int* pHeight, long* pSize, char* pFile );
43
44
45
46 // This function saves the bitmap; the first parameter is the pointer to the bitmap-
47 // buffer which want to be saved
48 BOOL SaveBMP( BYTE* pBuffer, int pWidth, int pHeight, long pPaddedSize, char* pFile );
```

Listing D.1: imageProcessing.h

```
1 #include "imageProcessing.h"
2
3 #define ON 1
4 #define OFF 0
5
6 bool LINE_LOCKING = ON;
7
8 bool WRITE_TEST_PICTURES = OFF;
9
10
11
12 // For testing purposes
13 // fstream target1("outfile1.xls", ios::out);
14 // fstream target2("outfile2.xls", ios::out);
15
16
17 static const double PI = 3.14159265;
```

```

18
19 // Downsampling factor
20 static const unsigned short ds = 4;
21
22 // Picture dimensions
23 //
24 static const unsigned int WIDTH = 320;
25 static const unsigned int HEIGHT = 240;
26 static const unsigned int SMALLWIDTH = WIDTH/ds;
27 static const unsigned int SMALLHEIGHT = HEIGHT/ds;
28 static const unsigned int FINALWIDTH = WIDTH/ds - 2;
29 static const unsigned int FINALHEIGHT = (HEIGHT/ds - 2) - (HEIGHT/ds - 2)/4;
30
31
32 // This are the line parameters extracted
33 // from the picture
34 static double sampleOffset = FINALWIDTH/2;
35 static double sampleRise = 0;
36 static double last3Rise[3];
37 static long localCounter = 0;
38
39
40
41 // As stated in imageProcessing.h, the CVM library
42 // has its own namespace
43 using namespace cvm;
44
45 // Next 4 variables serve for the Line-Locking
46 // effect
47 //
48 // weighting the next picture with this
49 static rmatrix weightingPicture(FINALHEIGHT,FINALWIDTH);
50
51 // At program begin, we initialize weightingLine;
52 // the distance from a pixel in a row to the pixel who
53 // represents the line corresponds to the index in
54 // weightingLine[index=distance from line]
55 double weightingLine[78];
56
57 // Parameter for the gaussian function
58 static const double a = 0.0875;
59 //static const double a = 0.05;
60
61 // First picture has no weightingPicture to be weighted with
62 static bool firstTime = true;
63
64
65 // Number of steps in bezier function
66 static const unsigned short NB = 20;
67
68 // Vectors with bezier point information
69 static rvector X(NB+1);
70 static rvector Y(NB+1);
71
72
73 // This variable holds the value for moving the wheels of the
74 // car into the correct position
75 double setAngle = 0;
76
77
78
79 BYTE* LoadBMP( int* pWidth, int* pHeight, long* pSize, char* pFile )
80 {
81     // Declare bitmap structures
82     BITMAPFILEHEADER bmpheader;
83     BITMAPINFOHEADER bmpinfo;
84
85     // Value to be used in ReadFile() functions
86     DWORD bytesread;
87
88     // Open file to read from
89     HANDLE file = CreateFile( pFile , GENERIC_READ, FILE_SHARE_READ,
90                             NULL, OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL );
91
92     if( NULL == file ){
93         // Could not open file
94         return NULL;
95     }
96
97     // Read file header

```

```

98     if( ReadFile( file, &bmpheader, sizeof(BITMAPFILEHEADER), &bytesread, NULL ) == false ){
99         CloseHandle( file );
100         return NULL;
101     }
102
103     // Read bitmap info
104     if( ReadFile( file, &bmpinfo, sizeof(BITMAPINFOHEADER), &bytesread, NULL ) == false ){
105         CloseHandle( file );
106         return NULL;
107     }
108
109     // Check if file is a bitmap
110     if( bmpheader.bfType != 'MB' ){
111         CloseHandle( file );
112         return NULL;
113     }
114
115     // Get image measurements
116     *pWidth = bmpinfo.biWidth;
117     *pHeight = abs( bmpinfo.biHeight );
118
119     // Check if bitmap is uncompressed
120     if( bmpinfo.biCompression != BI_RGB ){
121         CloseHandle( file );
122         return NULL;
123     }
124
125     // Check if we have 24 bit bitmap
126     if( bmpinfo.biBitCount != 24 ){
127         CloseHandle( file );
128         return NULL;
129     }
130
131     // Create buffer to hold the data
132     *pSize = bmpheader.bfSize - bmpheader.bfOffBits;
133     BYTE* Buffer = new BYTE[*pSize];
134
135     // Move file pointer to start of bitmap data
136     SetFilePointer( file, bmpheader.bfOffBits, NULL, FILE_BEGIN );
137
138     // Read bitmap data
139     if ( ReadFile( file, Buffer, *pSize, &bytesread, NULL ) == false ){
140         delete [] Buffer;
141         CloseHandle( file );
142         return NULL;
143     }
144
145     // everything successful here: close file and return buffer
146     CloseHandle( file );
147     return Buffer;
148 }
149
150 BOOL SaveBMP( BYTE* pBuffer, int pWidth, int pHeight, long pPaddedSize, char* pFile )
151 {
152     // Declare bitmap structures...
153     BITMAPFILEHEADER bmfh;
154     BITMAPINFOHEADER info;
155
156     // ... and initialize them to zero
157     memset( &bmfh, 0, sizeof(BITMAPFILEHEADER) );
158     memset( &info, 0, sizeof(BITMAPINFOHEADER) );
159
160     // Fill the fileheader with data
161     bmfh.bfType = 0x4d42; // 0x4d42 = 'BM'
162     bmfh.bfReserved1 = 0;
163     bmfh.bfReserved2 = 0;
164     bmfh.bfSize = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) + pPaddedSize;
165     bmfh.bfOffBits = 0x36; // Number of bytes to start of bitmap bits
166
167     // Fill the infoheader
168     info.biSize = sizeof(BITMAPINFOHEADER);
169     info.biWidth = pWidth;
170     info.biHeight = pHeight;
171     info.biPlanes = 1; // We only have one bitplane
172     info.biBitCount = 24; // RGB mode is 24 bits
173     info.biCompression = BI_RGB;
174     info.biSizeImage = 0; // Can be zero for 24 bit images
175     info.biXPelsPerMeter = 0x0ec4; // Paint uses this value
176     info.biYPelsPerMeter = 0x0ec4;
177     info.biClrUsed = 0; // We are in RGB mode and have no palette

```

```

178     info.biClrImportant = 0; // All colors are important
179
180     // Now we open the file to write to
181     HANDLE file = CreateFile ( pFile , GENERIC_WRITE, FILE_SHARE_READ,
182                             NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );
183
184     if( file == NULL ){
185         CloseHandle( file );
186         return false;
187     }
188
189     // Write fileheader
190     unsigned long bwritten;
191     if( WriteFile( file, &bmfh, sizeof(BITMAPFILEHEADER), &bwritten, NULL ) == false ){
192         CloseHandle( file );
193         return false;
194     }
195
196     // Write infoheader
197     if( WriteFile( file, &info, sizeof(BITMAPINFOHEADER), &bwritten, NULL ) == false ){
198         CloseHandle( file );
199         return false;
200     }
201
202     // Write image data...
203     if( WriteFile( file, pBuffer, pPaddedSize, &bwritten, NULL ) == false ){
204         CloseHandle( file );
205         return false;
206     }
207
208     // ... and close file
209     CloseHandle( file );
210
211     return true;
212 }
213
214
215 void BMP2MATRIX( BYTE* pBuffer, int pWidth, int pHeight, rmatrix* pPicture )
216 {
217     // Find the number of padding bytes
218     int padding = 0;
219     int scanlinebytes = pWidth * 3;
220     while( ((scanlinebytes + padding) % 4) != 0 ){
221         padding++;
222     }
223
224     // Get the padded scanline width
225     int psw = scanlinebytes + padding;
226
227     long bufpos = 0;
228     long tempX = 0;
229
230     for( int y=0; y<pHeight; y++ ){
231         for( int x=0; x<3*pWidth; x+=3, tempX++ ){
232             bufpos = (pHeight - y - 1) * psw + x;
233             // We read the pixel data from the buffer and write it to our matrix,
234             // which holds our gray-scaled picture
235             (*pPicture)(y+1,((tempX%320)+1)) = 0.299*pBuffer[bufpos] + 0.587*pBuffer[bufpos + 1]
236                 + 0.114*pBuffer[bufpos + 2];
237         }
238     }
239 }
240
241
242 BYTE* MATRIX2BMP( int pWidth, int pHeight, long* pNewSize, rmatrix* pPicture )
243 {
244     // We have to find out with how many bytes we have to padd for the next
245     // DWORD boundary
246     // DWORD = 4 Bytes
247     int padding = 0;
248     int scanlinebytes = pWidth * 3;
249     while( ( scanlinebytes + padding ) % 4 != 0 ) {
250         padding++;
251     }
252
253     // Get the padded scanline width
254     int psw = scanlinebytes + padding;
255
256     // We can already store the size of the new padded buffer...
257     *pNewSize = pHeight * psw;

```

```

258
259 // ... and create new buffer
260 BYTE* newbuf = new BYTE[*pNewSize];
261
262 // Fill the buffer with zero bytes then we don't have to add extra padding
263 // zero bytes later on
264 memset( newbuf, 0, *pNewSize );
265
266 long bufpos = 0;
267 long newpos = 0;
268 long tempX = 0;
269 double grayValue;
270 double myRise = sampleRise;
271 double myOffset = sampleOffset;
272
273 // Now we loop trough our matrix and write data to the new buffer
274 for ( int y = 0; y < pHeight; y++ ){
275     for ( int x = 0; x < 3 * pWidth; x+=3, tempX++ ){
276         // Position in padded buffer
277         newpos = ( pHeight - y - 1 ) * psw + x;
278
279         // Extract the gray-scaled color information from our matrix
280         grayValue = (*pPicture)(y+1,((tempX%pWidth)+1));
281
282         // GrayValue is assigned to Red, Green and Blue,
283         // this results in a final grayscale picture...
284         newbuf[newpos] = grayValue;
285         newbuf[newpos + 1] = grayValue;
286         newbuf[newpos + 2] = grayValue;
287
288         if( !WRITE_TEST_PICTURES ){
289             // ... and of course, write a green line to visualize the proper operation
290             // of the function findTheLine()
291
292             if( y>=((HEIGHT/ds-2)/4) && ((tempX%pWidth)+1)==((int)myOffset) ){
293                 // Blue
294                 newbuf[newpos] = 0;
295                 // Green
296                 newbuf[newpos + 1] = 255;
297                 // Red
298                 newbuf[newpos + 2] = 255;
299             }
300         }
301     }
302     if( !WRITE_TEST_PICTURES ){
303         if( y>=((HEIGHT/ds-2)/4) ){
304             myOffset += myRise;
305         }
306     }
307 }
308
309 return newbuf;
310 }
311
312 double ANGLE( double pOffset, double pRise, unsigned short pN, double pG, double pr, double pr2 )
313 {
314     rvector A(2);
315     rvector B(2);
316     rvector H1(2);
317     rvector H2(2);
318     rvector P(2);
319
320     double averageAngle = (last3Rise[0]+last3Rise[1]+last3Rise[2])/3;
321     double x = sin(averageAngle*PI/180);
322     double y = cos(averageAngle*PI/180);
323
324     B(1) = pOffset;
325     B(2) = (HEIGHT/ds-2)/4;
326
327     A(1) = FINALWIDTH/2+pr*x;
328     A(2) = SMALLHEIGHT-pr*y;
329
330
331     H1(1) = A(1)+pG*x;
332     H1(2) = A(2)-pG*y;
333
334     x = sin(-atan(pRise));
335     y = cos(-atan(pRise));
336
337

```

```

338 H2(1) = pOffset-pG*x;
339 H2(2) = (HEIGHT/ds-2)/4+pG*y;
340
341
342 X(1) = A(1);
343 X(pN+1) = B(1);
344
345 Y(1) = A(2);
346 Y(pN+1) = B(2);
347
348 BOOL found = FALSE;
349
350 double t;
351 for( int i=2; i<=pN; i++ ){
352     t = (double)i/pN;
353     X(i) = A(1)*(1-t)*(1-t)*(1-t) + 3*H1(1)*(1-t)*(1-t)*t + 3*H2(1)*(1-t)*t*t + B(1)*t*t*t;
354     Y(i) = A(2)*(1-t)*(1-t)*(1-t) + 3*H1(2)*(1-t)*(1-t)*t + 3*H2(2)*(1-t)*t*t + B(2)*t*t*t;
355
356     if( sqrt( (X(i)-A(1))*(X(i)-A(1)) + (Y(i)-A(2))*(Y(i)-A(2)) ) > pr2 && found==FALSE ){
357         found = TRUE;
358         P(1) = X(i);
359         P(2) = Y(i);
360     }
361 }
362
363 return ( atan( (FINALWIDTH/2-P(1))/(SMALLHEIGHT-P(2)) )*180/PI + averageAngle )/2;
364 }
365 }
366
367 void findTheLine()
368 {
369     // CVM library has its own namespace
370     using namespace cvm;
371
372     // Some variables
373     int i=0, j=0, m=0, n=0;
374
375     // Load the bitmap into a matrix
376     int tWidth;
377     int tHeight;
378     long tSize;
379     long tNewSize;
380
381     rmatrix myPixell(HEIGHT,WIDTH);
382
383     BYTE* imageData = LoadBMP( &tWidth, &tHeight, &tSize, "rawPicture.bmp" );
384     BMP2MATRIX( imageData, tWidth, tHeight, &myPixell );
385     delete [] imageData;
386
387     // Antialiasing Filter and Downsampling in one step
388     //-----
389     rmatrix intensityPictureB( SMALLHEIGHT, SMALLWIDTH );
390
391     for( i=1, m=1; i<=HEIGHT-ds; i+=ds, m++ ){
392         for( j=1, n=1; j<=WIDTH-ds; j+=ds, n++ ){
393             intensityPictureB(m,n) = (myPixell(i,j) + myPixell(i,j+1) + myPixell(i,j+2)
394                 + myPixell(i,j+3) + myPixell(i+1,j) + myPixell(i+1,j+1)
395                 + myPixell(i+1,j+2) + myPixell(i+1,j+3) + myPixell(i+2,j)
396                 + myPixell(i+2,j+1) + myPixell(i+2,j+2) + myPixell(i+2,j+3)
397                 + myPixell(i+3,j) + myPixell(i+3,j+1) + myPixell(i+3,j+2)
398                 + myPixell(i+3,j+3)) / (ds*ds);
399         }
400     }
401     //-----
402
403     // Filter the picture with a highpass to detect edges
404     //-----
405     rmatrix edgePictureB( SMALLHEIGHT-2, SMALLWIDTH-2 );
406
407     double k = 1/sqrt(8);
408     double s = 0.5;
409
410     for( i=2, m=1; i<=SMALLHEIGHT-1; i++, m++ ){
411         for( j=2, n=1; j<=SMALLWIDTH-1; j++, n++ ){
412             edgePictureB(m,n) = abs(-k*intensityPictureB(i-1,j-1) - s*intensityPictureB(i-1,j)
413                 + k*intensityPictureB(i-1,j+1) - s*intensityPictureB(i,j-1)
414                 + s*intensityPictureB(i,j+1) - k*intensityPictureB(i+1,j-1)
415                 + s*intensityPictureB(i+1,j) + k*intensityPictureB(i+1,j+1));
416         }
417     }

```

```

418 //-----
419
420
421 // Grab lower part of the picture
422 //-----
423 rmatrix intensityPicture( FINALHEIGHT, FINALWIDTH );
424 rmatrix edgePicture( FINALHEIGHT, FINALWIDTH );
425
426 for( i=14, m=1; m<=FINALHEIGHT; i++, m++ ){
427     for( j=1, n=1; j<=FINALWIDTH; j++, n++ ){
428         intensityPicture(m,n) = intensityPictureB(i,j);
429         edgePicture(m,n) = edgePictureB(i,j);
430     }
431 }
432 //-----
433
434 // Normalize both pictures
435 //-----
436 double avgEdge=0, avgIntensity=0, tempEdge=0, tempIntensity=0, stdEdge=0, stdIntensity=0;
437 double minIntensity=255, minEdge=255, tempIntensityMin=0, tempEdgeMin=0;
438
439 int minIntensityPOS = intensityPicture.indofmin();
440 int minEdgePOS = edgePicture.indofmin();
441
442 int rowI = minIntensityPOS % FINALHEIGHT;
443 int colI = (minIntensityPOS-(minIntensityPOS % FINALHEIGHT))/FINALHEIGHT+1;
444 int rowE = minEdgePOS % FINALHEIGHT;
445 int colE = (minEdgePOS-(minEdgePOS % FINALHEIGHT))/FINALHEIGHT+1;
446
447 if( rowI == 0 ){
448     rowI = FINALHEIGHT;
449     colI--;
450 }
451
452 if( rowE == 0 ){
453     rowE = FINALHEIGHT;
454     colE--;
455 }
456
457 minIntensity = intensityPicture(rowI,colI);
458 minEdge = edgePicture(rowE,colE);
459
460 for( i=1; i<=FINALHEIGHT; i++ ){
461     for( j=1; j<=FINALWIDTH; j++ ){
462         // tempIntensity & tempEdge hold the sum of all pixels
463         tempIntensity += abs(intensityPicture(i,j)-minIntensity);
464         tempEdge += abs(edgePicture(i,j)-minEdge);
465     }
466 }
467
468 avgIntensity = tempIntensity / (FINALHEIGHT*FINALWIDTH);
469 avgEdge = tempEdge / (FINALHEIGHT*FINALWIDTH);
470
471 tempIntensity=0;
472 tempEdge=0;
473
474 for( i=1; i<=FINALHEIGHT; i++ ){
475     for( j=1; j<=FINALWIDTH; j++ ){
476         tempIntensity += ((intensityPicture(i,j) - avgIntensity)
477             * (intensityPicture(i,j) - avgIntensity));
478         tempEdge += ((edgePicture(i,j) - avgEdge) * (edgePicture(i,j) - avgEdge));
479     }
480 }
481
482 stdIntensity = sqrt(tempIntensity/(FINALHEIGHT*FINALWIDTH-1));
483 stdEdge = sqrt(tempEdge/(FINALHEIGHT*FINALWIDTH-1));
484
485 for( i=1; i<=FINALHEIGHT; i++ ){
486     for( j=1; j<=FINALWIDTH; j++ ){
487         intensityPicture(i,j) = intensityPicture(i,j) / stdIntensity;
488         edgePicture(i,j) = edgePicture(i,j) / stdEdge;
489     }
490 }
491 //-----
492
493 // Weighting intensityPicture and edgePicture with the weightingPicture
494 //-----
495 if( LINE_LOCKING ){
496     if( !firstTime ){
497         for( i=1; i<=FINALHEIGHT; i++ ){

```

```

498         for( j=1; j<=FINALWIDTH; j++ ){
499             intensityPicture(i,j) = intensityPicture(i,j) * weightingPicture(i,j);
500             edgePicture(i,j) = edgePicture(i,j) * weightingPicture(i,j);
501         }
502     }
503 }
504 else{
505     for( int lineDistance=0; lineDistance<=77; lineDistance++ ){
506         weightingLine[lineDistance]=exp(-(a*lineDistance)*(a*lineDistance));
507     }
508
509     double distance;
510     double myRise = sampleRise;
511     double myOffset = sampleOffset;
512
513     for( i=1; i<=FINALHEIGHT; i++ ){
514         for( j=1; j<=FINALWIDTH; j++ ){
515             distance = abs( (int)myOffset - j );
516             weightingPicture(i,j) = weightingLine[(unsigned short)distance];
517         }
518         myOffset += myRise;
519     }
520
521     for( i=1; i<=FINALHEIGHT; i++ ){
522         for( j=1; j<=FINALWIDTH; j++ ){
523             intensityPicture(i,j) = intensityPicture(i,j) * weightingPicture(i,j);
524             edgePicture(i,j) = edgePicture(i,j) * weightingPicture(i,j);
525         }
526     }
527 }
528 }
529 firstTime =false;
530 //-----
531 // Polynomapproximation 1. Order
532 //-----
533 rmatrix A( FINALHEIGHT*FINALWIDTH, 2 );
534 rmatrix AT( 2, FINALHEIGHT*FINALWIDTH );
535 rmatrix y( FINALHEIGHT*FINALWIDTH, 2 );
536 rmatrix R( 1, 2 );
537 rmatrix Q( 2, 2 );
538 rmatrix Qi( 2, 2 );
539 rmatrix templ( 2,2 );
540 rmatrix x( 2,2 );
541 int index=0;
542 double temp=0;
543
544 for( i=1; i<=FINALHEIGHT; i++ ){
545     R(1,1) = i;
546     R(1,2) = 1;
547     for( j=1; j<=FINALWIDTH; j++ ){
548         index = (i - 1) * FINALWIDTH + j;
549         A(index,1) = ((double)i / FINALHEIGHT)
550             * edgePicture(i,j) * intensityPicture(i,j) * R(1,1);
551         A(index,2) = ((double)i / FINALHEIGHT)
552             * edgePicture(i,j) * intensityPicture(i,j) * R(1,2);
553         y(index,1) = ((double)i / FINALHEIGHT)
554             * edgePicture(i,j) * intensityPicture(i,j) * j;
555         y(index,2) = 0;
556     }
557 }
558 }
559
560 AT.transpose(A);
561 Q = AT * A;
562 Qi = Q;
563 temp = Qi(1,1);
564 Qi(1,1) = -Qi(2,2);
565 Qi(2,2) = -temp;
566 templ = Q * Qi;
567 Qi = Qi / templ(1,1);
568 x = Qi * AT * y;
569 //-----
570 // Draw the line to the Edge-Picture
571 //-----
572 sampleRise = x(1,1);
573 sampleOffset = x(2,1)-1;
574
575 if( !localCounter ){

```

```

578     last3Rise[0] = sampleRise;
579     last3Rise[1] = sampleRise;
580     last3Rise[2] = sampleRise;
581     localCounter++;
582 }
583 else{
584     last3Rise[ localCounter % 3 ] = sampleRise;
585     localCounter++;
586 }
587
588 setAngle = ANGLE( sampleOffset, sampleRise, NB );
589
590 tWidth = SMALLWIDTH-2;
591 tHeight = SMALLHEIGHT-2;
592 BYTE* imageData2 = MATRIX2BMP( tWidth, tHeight, &tNewSize, &edgePictureB );
593
594 BOOL success = SaveBMP( imageData2, tWidth, tHeight, tNewSize, "edgePicture.bmp" );
595 // There is no need to test for success (true/false) but it could be implemented
596 // fail-safe
597
598 delete [] imageData2;
599
600 if( WRITE_TEST_PICTURES ){
601     double testRise = sampleRise;
602     double testOffset = sampleOffset;
603
604     tWidth = FINALWIDTH;
605     tHeight = FINALHEIGHT;
606
607     rmatrix linePicture( FINALHEIGHT,FINALWIDTH );
608
609     for( i=1; i<=FINALHEIGHT; i++ ){
610         for( j=1; j<=FINALWIDTH; j++ ){
611             if( i>=((HEIGHT/ds-2)/4) && j==(int)testOffset ){
612                 linePicture(i,j) = 255;
613             }
614             else{
615                 linePicture(i,j) = 0;
616             }
617         }
618         testOffset += testRise;
619     }
620
621     BYTE* imageData3 = MATRIX2BMP( tWidth, tHeight, &tNewSize, &linePicture );
622     success = SaveBMP( imageData3, tWidth, tHeight, tNewSize, "linePicture.bmp" );
623     delete [] imageData3;
624 }
625
626 //-----
627 // Make the weightingPicture
628 //-----
629 if( LINE_LOCKING ){
630     double distance;
631     double myRise = sampleRise;
632     double myOffset = sampleOffset;
633
634     for( i=1; i<=FINALHEIGHT; i++ ){
635         for( j=1; j<=FINALWIDTH; j++ ){
636             distance = abs( (int)myOffset - j );
637             weightingPicture(i,j) = weightingLine( (unsigned short)distance );
638         }
639         myOffset += myRise;
640     }
641 }
642 //-----
643
644
645
646 if( WRITE_TEST_PICTURES && LINE_LOCKING ){
647     tWidth = FINALWIDTH;
648     tHeight =FINALHEIGHT;
649
650     rmatrix testWeightingPicture( FINALHEIGHT,FINALWIDTH );
651
652     for( i=1; i<=FINALHEIGHT; i++ ){
653         for( j=1; j<=FINALWIDTH; j++ ){
654             testWeightingPicture(i,j) = 255 * weightingPicture(i,j);
655         }
656     }
657

```

```

658     BYTE* imageData4 = MATRIX2BMP( tWidth, tHeight, &tNewSize, &testWeightingPicture );
659     success = SaveBMP( imageData4, tWidth, tHeight, tNewSize, "weightingPicture.bmp" );
660     delete [] imageData4;
661 }
662 }

```

Listing D.2: imageProcessing.cpp

```

1 // 80211CarDlg.h : header file
2 //
3 //{{AFX_INCLUDES()
4 #include "gif89a.h"
5 //}}AFX_INCLUDES
6
7 #if !defined(AFX_80211CARDLG_H_C35E8F58_E592_44FF_A50F_0790693C9B9D__INCLUDED_)
8 #define AFX_80211CARDLG_H_C35E8F58_E592_44FF_A50F_0790693C9B9D__INCLUDED_
9
10 #include "winsock2.h"
11
12 #if _MSC_VER > 1000
13 #pragma once
14 #endif // _MSC_VER > 1000
15
16 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
17 // CMy80211CarDlg dialog
18
19 class CMy80211CarDlg : public CDialog
20 {
21 // Construction
22 public:
23     CMy80211CarDlg(CWnd* pParent = NULL); // standard constructor
24
25     // This function extracts the picture from the screen...
26     // because we don't know exactly how the ActiveX component
27     // xplug works
28     void extractPicture( LPTSTR pFile, CMy80211CarDlg* pDlg );
29
30     // This function places the edge picture right under the
31     // original picture
32     void displayEdgePicture( LPTSTR pFile, CMy80211CarDlg* pDlg );
33
34     // Send a datagram to our car
35     void sendUDP();
36
37     // Variables for Mutex
38     BOOLEAN m_SB;
39     HANDLE m_hT;
40     HANDLE m_hMSB;
41
42     SOCKADDR_IN addrSrv;
43     SOCKET mySocket;
44
45 // Dialog Data
46 //{{AFX_DATA(CMy80211CarDlg)
47     enum { IDD = IDD_MY80211CAR_DIALOG };
48     CGif89a m_xplug;
49 //}}AFX_DATA
50
51 // ClassWizard generated virtual function overrides
52 //{{AFX_VIRTUAL(CMy80211CarDlg)
53 protected:
54     virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
55 //}}AFX_VIRTUAL
56
57 // Implementation
58 protected:
59     HICON m_hIcon;
60
61     // Generated message map functions
62 //{{AFX_MSG(CMy80211CarDlg)
63     virtual BOOL OnInitDialog();
64     afx_msg void OnPaint();
65     afx_msg HCURSOR OnQueryDragIcon();
66     afx_msg void OnStart();
67     afx_msg void OnStop();
68     virtual void OnOK();
69 //}}AFX_MSG
70     DECLARE_MESSAGE_MAP()
71 };

```

```

72 //{{AFX_INSERT_LOCATION}}
73 // Microsoft Visual C++ will insert additional declarations immediately before the previous line.
74
75
76 #endif // !defined(AFX_80211CARDLG_H_C35E8F58_E592_44FF_A50F_0790693C9B9D_INCLUDED_)

```

Listing D.3: 80211CarDlg.h

```

1 // 80211CarDlg.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "80211Car.h"
6 #include "80211CarDlg.h"
7 #include "imageProcessing.h"
8
9 #ifdef _DEBUG
10 #define new DEBUG_NEW
11 #undef THIS_FILE
12 static char THIS_FILE[] = __FILE__;
13 #endif
14
15 // Variable used for weightingPicture in imageProcessing.cpp
16 // and for sending a datagram to our car
17 unsigned long counter = 1;
18
19
20 // This variable means, that every n-th datagram contains the
21 // speed information
22 static const unsigned short speedPacket = 5;
23
24 // Variable for the speed of the car in percent
25 static unsigned short speed = 13;
26
27 // Variable for adjusting the servos
28 extern double setAngle;
29
30
31 extern bool WRITE_TEST_PICTURES;
32
33 ////////////////////////////////////////////////////////////////////
34 // CMy80211CarDlg dialog
35
36 CMy80211CarDlg::CMy80211CarDlg(CWnd* pParent /*=NULL*/)
37 : CDialog(CMy80211CarDlg::IDD, pParent)
38 {
39 //{{AFX_DATA_INIT(CMy80211CarDlg)
40 // NOTE: the ClassWizard will add member initialization here
41 //}}AFX_DATA_INIT
42 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
43 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
44 }
45
46 void CMy80211CarDlg::DoDataExchange(CDataExchange* pDX)
47 {
48 CDialog::DoDataExchange(pDX);
49 //{{AFX_DATA_MAP(CMy80211CarDlg)
50 DDX_Control(pDX, IDC_GIF891, m_xplug);
51 //}}AFX_DATA_MAP
52 }
53
54 BEGIN_MESSAGE_MAP(CMy80211CarDlg, CDialog)
55 //{{AFX_MSG_MAP(CMy80211CarDlg)
56 ON_WM_PAINT()
57 ON_WM_QUERYDRAGICON()
58 ON_BN_CLICKED(IDSTART, OnStart)
59 ON_BN_CLICKED(IDSTOP, OnStop)
60 //}}AFX_MSG_MAP
61 END_MESSAGE_MAP()
62
63 ////////////////////////////////////////////////////////////////////
64 // CMy80211CarDlg message handlers
65
66 BOOL CMy80211CarDlg::OnInitDialog()
67 {
68 CDialog::OnInitDialog();
69
70 // Set the icon for this dialog. The framework does this automatically
71 // when the application's main window is not a dialog

```

```

72  SetIcon(m_hIcon, TRUE);           // Set big icon
73  SetIcon(m_hIcon, FALSE);        // Set small icon
74
75  // Our dialog is always on top, why?
76  // Because we extract the picture from the screen, and assume that there
77  // is another window partly covering our picture from the camera, then
78  // the edge picture will contain the edge of the covering window!
79  SetWindowPos(&this->wndTopMost, 0, 0, 0, 0, SWP_NOMOVE|SWP_NOSIZE);
80
81  // Run the ActiveX component
82  m_xplug.SetRemoteHost("192.168.10.200");
83  m_xplug.SetRemotePort(8481);
84  m_xplug.SetTimeout(5);
85  m_xplug.SetAuthType(1);
86  m_xplug.SetPreviewFrameRate(1);
87  m_xplug.SetPreviewWidth(320);
88  m_xplug.SetPreviewHeight(240);
89  m_xplug.SetDeviceSerialNo("");
90  m_xplug.Play();
91
92  // Initialize our socket
93  WSADATA wsaData;
94  WSAStartup(MAKEWORD(1,1), &wsaData);
95
96  mySocket = socket(AF_INET, SOCK_DGRAM, 0);
97
98  addrSrv.sin_family = AF_INET;
99  addrSrv.sin_addr.s_addr = inet_addr("192.168.10.20");
100  addrSrv.sin_port = htons(3000);
101
102  return TRUE; // return TRUE unless you set the focus to a control
103 }
104
105 // If you add a minimize button to your dialog, you will need the code below
106 // to draw the icon. For MFC applications using the document/view model,
107 // this is automatically done for you by the framework.
108
109 void CMy80211CarDlg::OnPaint()
110 {
111     if (IsIconic())
112     {
113         CPaintDC dc(this); // device context for painting
114
115         SendMessage(WM_ICONERASEBKGD, (WPARAM) dc.GetSafeHdc(), 0);
116
117         // Center icon in client rectangle
118         int cxIcon = GetSystemMetrics(SM_CXICON);
119         int cyIcon = GetSystemMetrics(SM_CYICON);
120         CRect rect;
121         GetClientRect(&rect);
122         int x = (rect.Width() - cxIcon + 1) / 2;
123         int y = (rect.Height() - cyIcon + 1) / 2;
124
125         // Draw the icon
126         dc.DrawIcon(x, y, m_hIcon);
127     }
128     else
129     {
130         CDialog::OnPaint();
131     }
132 }
133
134 // The system calls this to obtain the cursor to display while the user drags
135 // the minimized window.
136 HCURSOR CMy80211CarDlg::OnQueryDragIcon()
137 {
138     return (HCURSOR) m_hIcon;
139 }
140
141 DWORD WINAPI extractPictureThread( LPVOID lpParam )
142 {
143     CMy80211CarDlg* pDlg = (CMy80211CarDlg*)lpParam;
144     BOOLEAN localStopButton = FALSE;
145
146     while( !localStopButton ){
147         pDlg->extractPicture( "rawPicture.bmp", pDlg );
148         findTheLine();
149         pDlg->sendUDP();
150         pDlg->displayEdgePicture( "edgePicture.bmp", pDlg );
151     }

```

```

152     WaitForSingleObject( pDlg->m_hMSB, INFINITE );
153     localStopButton = pDlg->m_SB;
154     ReleaseMutex( pDlg->m_hMSB );
155
156     if( WRITE_TEST_PICTURES ){
157         localStopButton = true;
158         Beep(500,100);
159     }
160 }
161 CloseHandle( pDlg->m_hT );
162 CloseHandle( pDlg->m_hMSB );
163
164     return 0;
165 }
166
167 void CMy80211CarDlg::OnStart()
168 {
169     m_hMSB = CreateMutex( NULL, 0, NULL );
170     WaitForSingleObject( m_hMSB, INFINITE );
171     m_SB = FALSE;
172     ReleaseMutex( m_hMSB );
173
174     DWORD dwThreadID;
175     m_hT = CreateThread(
176         NULL,
177         0,
178         extractPictureThread,
179         this,
180         0,
181         &dwThreadID);
182 }
183
184 void CMy80211CarDlg::OnStop()
185 {
186     WaitForSingleObject( m_hMSB, INFINITE );
187     m_SB = TRUE;
188     ReleaseMutex( m_hMSB );
189
190     // Car should stop when the Stop-Button is clicked
191     char myMessage[4] = {'V','+', '0','0'};
192     sendto( mySocket, myMessage, 4, 0, (sockaddr*)&addrSrv, sizeof(addrSrv) );
193 }
194
195 void CMy80211CarDlg::OnOK()
196 {
197     WaitForSingleObject( m_hMSB, INFINITE );
198     m_SB = TRUE;
199     ReleaseMutex( m_hMSB );
200
201     // Car should stop when the OK-Button is clicked
202     char myMessage[4] = {'V','+', '0','0'};
203     sendto( mySocket, myMessage, 4, 0, (sockaddr*)&addrSrv, sizeof(addrSrv) );
204
205     closesocket(mySocket); WSACleanup();
206
207     CDialog::OnOK();
208 }
209
210 void CMy80211CarDlg::extractPicture( LPTSTR pFile, CMy80211CarDlg* pDlg )
211 {
212     CClientDC hScreenDC( pDlg );
213     CDC hmemDC;
214     hmemDC.CreateCompatibleDC( &hScreenDC );
215
216     HBITMAP hmemBM = CreateCompatibleBitmap( hScreenDC, 320, 240 );
217     SelectObject( hmemDC, hmemBM );
218
219     BitBlt( hmemDC, 0, 0, 320, 240, hScreenDC, 11, 11, SRCCOPY );
220
221     HGLOBAL hpixldata = GlobalAlloc( GMEM_FIXED, 320*240*3 );
222     void FAR* lpvpixldata = GlobalLock( hpixldata );
223
224     BITMAPINFO bmInfo;
225     bmInfo.bmiHeader.biSize = 40;
226     bmInfo.bmiHeader.biWidth = 320;
227     bmInfo.bmiHeader.biHeight = 240;
228     bmInfo.bmiHeader.biPlanes = 1;
229     bmInfo.bmiHeader.biBitCount = 24;
230     bmInfo.bmiHeader.biCompression = 0;
231     bmInfo.bmiHeader.biSizeImage = 0;

```

```

232 |   bmInfo.bmiHeader.biXPelsPerMeter = 0;
233 |   bmInfo.bmiHeader.biYPelsPerMeter = 0;
234 |   bmInfo.bmiHeader.biClrUsed = 0;
235 |   bmInfo.bmiHeader.biClrImportant = 0;
236 |
237 |   BITMAPFILEHEADER bmFileHeader;
238 |   bmFileHeader.bfType = 19778;
239 |   bmFileHeader.bfSize = (320*240*3)+40+14;
240 |   bmFileHeader.bfReserved1 = 0;
241 |   bmFileHeader.bfReserved2 = 0;
242 |   bmFileHeader.bfOffBits = 54;
243 |
244 |   GetDIBits( hmemDC, hmemBM, 0, 240, lpvpxldata, &bmInfo, DIB_RGB_COLORS );
245 |
246 |   CFile file;
247 |   file.Open( pFile, CFile::modeWrite|CFile::modeCreate );
248 |
249 |   file.Write( &bmFileHeader, sizeof(bmFileHeader) );
250 |   file.Write( &bmInfo, sizeof(bmInfo) );
251 |   file.Write( lpvpxldata, 320*240*3 );
252 |
253 |   file.Close();
254 |   GlobalUnlock( hpxldata );
255 |   GlobalFree( hpxldata );
256 |   DeleteObject( hmemBM );
257 |   DeleteDC( hmemDC );
258 |   ReleaseDC( &hScreenDC );
259 | }
260 |
261 | void CMy80211CarDlg::displayEdgePicture( LPTSTR pFile, CMy80211CarDlg* pDlg )
262 | {
263 |     CClientDC hScreenDC( pDlg );
264 |     CDC dc;
265 |     dc.CreateCompatibleDC( &hScreenDC );
266 |
267 |     HANDLE hPicture = LoadImage( NULL, pFile, IMAGE_BITMAP, 320, 240, LR_LOADFROMFILE );
268 |
269 |     HBITMAP* myPicture = ( HBITMAP* ) hPicture;
270 |     dc.SelectObject( myPicture );
271 |     hScreenDC.BitBlt( 11, 260, 320, 240, &dc, 0, 0, SRCCOPY );
272 |
273 |     DeleteObject( hPicture );
274 | }
275 |
276 | void CMy80211CarDlg::sendUDP()
277 | {
278 |     char myMessage[4];
279 |     char temp[3];
280 |
281 |     if( !(counter++ % speedPacket) ){
282 |         // prepare to send the speed information
283 |         // Format: V+xx
284 |         myMessage[0] = 'V';
285 |         myMessage[1] = '+';
286 |         if( speed<10 ){
287 |             itoa(speed,temp,10);
288 |             myMessage[2] = '0';
289 |             myMessage[3] = temp[0];
290 |         }
291 |         else{
292 |             itoa(speed,temp,10);
293 |             myMessage[2] = temp[0];
294 |             myMessage[3] = temp[1];
295 |         }
296 |     }
297 |     else{
298 |         // prepare to send the direction information
299 |         // Format: H+xx or H-xx
300 |         myMessage[0] = 'H';
301 |         if( setAngle < 0){
302 |             myMessage[1] = '-';
303 |         }
304 |         else{
305 |             myMessage[1] = '+';
306 |         }
307 |
308 |         if( abs(setAngle)<10 ){
309 |             itoa(abs((int)(setAngle)),temp,10);
310 |             myMessage[2] = '0';
311 |             myMessage[3] = temp[0];

```

```
312     }
313     else{
314         itoa(abs((int)(setAngle)),temp,10);
315         myMessage[2] = temp[1];
316         myMessage[3] = temp[0];
317     }
318 }
319
320 sendto( mySocket, myMessage, 4, 0, (sockaddr*)&addrSrv, sizeof(addrSrv) );
321 }
```

Listing D.4: 80211CarDlg.cpp