



FACHHOCHSCHUL-BACHELORSTUDIENGANG

Automatisierungstechnik / Industrielle Informatik

**Entwurf einer Benutzeroberfläche zur Steuerung der Aktoren und
Auslesen der Sensoren des Eurobot-Roboters**

ALS BACHELORARBEIT EINGEREICHT

zur Erlangung des akademischen Grades

Bachelor of Science in Engineering

von

Christian Mitterndorfer

März 2010

Betreuung der Bachelorarbeit durch

Prof.(FH) DI Walter Rokitansky

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt, die den benutzten Quellen entnommenen Stellen als solche kenntlich gemacht habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

.....
<<Vorname Name>>

<<Wohnort>>, <<Datum>>

KURZFASSUNG

Seit dem Jahr 2007 nimmt das RoboRacingTeam der Fachhochschule Wels an dem internationalen Roboterwettbewerb EUROBOT^{open} teil.

Diese Arbeit beschäftigt sich mit dem Erstellen einer Benutzeroberfläche zur Steuerung der Aktoren und Auslesen der Sensoren des Eurobot – Roboters des Jahres 2010. Aufgaben dieser Software sind die Unterstützung der Inbetriebnahme und Testphase des Roboters sowie die Konfiguration der Kamera beim Bewerb.

Die Arbeit gibt einen kurzen Überblick über das Regelwerk der EUROBOT^{open} sowie die Aufgabenstellung in diesem Jahr. Weiters beschäftigt sie sich mit der Kommunikation zwischen PC und Roboter bzw. Kamera. Sie gibt Auskunft darüber wie die Befehle zur Steuerung der Aktoren implementiert wurden und wie die Kamera einfach und schnell parametrisiert wird. Auch das Empfangen von Daten vom Roboter oder von der Kamera ist Teil dieser Arbeit.

Zum Schluss werden weiterführende Möglichkeiten dieses Projektes vorgestellt um auch in Zukunft konkurrenzfähig zu bleiben.

INHALTSVERZEICHNIS

1	MOTIVATION	1
1.1	EUROBOT ^{open}	1
1.1.1	EUROBOT ^{open} Thema 2010.....	1
2	BENUTZEROBERFLÄCHE	3
2.1	Konzept	3
3	Die serielle Schnittstelle	4
3.1	Grundlagen	4
3.2	Datenübertragung	5
3.3	Datenrate	5
3.4	Spannungspegel.....	7
3.5	Anschlussbelegung.....	7
3.6	Implementierung.....	8
3.6.1	Einstellungen	8
3.6.2	Daten senden	11
3.6.3	Daten empfangen.....	11
3.6.3.1	Daten sind für Kamerakonfiguration bestimmt:.....	11
3.6.3.2	Daten sind für Kameraauswertung bestimmt:	11
3.6.3.3	Daten für keine bestimmte Stelle bestimmt:	11
4	Befehle.....	13
4.1	Konzept	13
4.1.1	Befehl „Vorwärts fahren“.....	13
4.1.2	Befehl „Rückwärts fahren“	13
4.1.3	Befehl „Drehung links“	13
4.1.4	Befehl „Drehung rechts“	14
4.1.5	Befehl „Einzelnen Motor ansteuern“	14
4.1.6	Befehl „Servo ansteuern“	14
4.2	Implementierung der Befehle	14
4.2.1	Konstruktor für Vorwärts – bzw. Rückwärts fahren	15
4.2.2	Konstruktor für Drehung links bzw. rechts	15

4.2.3	Konstruktor für einzelnen Motor ansteuern	16
4.2.4	Konstruktor für Servo ansteuern	16
4.2.5	Konstruktor für Befehle die schon bekannt sind	16
4.2.6	Konstruktor für eine Sendepause	17
4.3	Eingabe der Befehle	17
4.3.1	Eingabe der Maximalwerte	18
4.3.1.1	Änderung der TickFrequency – Eigenschaft	18
4.3.1.2	Änderung der LargeChange – Eigenschaft.....	19
4.3.1.3	Maximalwerte überprüfen	19
4.3.2	Eingabe der Daten	20
4.3.2.1	Überprüfung ob eingegebener Wert nicht größer Maximalwert ist	20
4.3.2.2	Überprüfung ob eingegebener Wert eine Zahl ist	21
4.3.3	Senden der Befehle.....	22
4.4	Vergabe von Motornamen.....	23
5	Kamera.....	24
5.1	Grundkonzept	24
5.2	Kamera parametrieren	24
5.2.1	Daten anfordern.....	26
5.2.2	Modus auf binär umstellen	26
5.2.3	Modus auf Werte umstellen	27
5.2.4	Threshold umstellen	27
5.3	Auswertung	28
6	Sendeliste senden / Daten empfangen	30
6.1	Sendeliste senden	30
6.1.1	Senden	30
6.1.2	Speichern / Laden einer Sendeliste.....	32
6.1.3	Einzelne Befehle aus der Sendeliste löschen.....	34
6.2	Daten empfangen.....	34
7	ZUSAMMENFASSUNG UND AUSBLICK.....	35
8	LITERATUR	36
8.1	Bücherliste.....	36
8.2	Internet.....	36

ABBILDUNGSVERZEICHNIS

Abbildung 1-1: Spielfeld EUROBOT ^{open} Thema 2010.....	2
Abbildung 2-1: Robot comunication 2009.....	3
Abbildung 3-1: Übertragung des Zeichens G über die serielle Schnittstelle	5
Abbildung 3-2: D-Sub Stecker 25 polig.....	7
Abbildung 3-3: D-Sub Buchse 25 polig.....	7
Abbildung 3-4: D-Sub Stecker 9 polig.....	7
Abbildung 3-5: D-Sub Buchse 9polig	7
Abbildung 3-6: Unterfenster für die Schnittstelleneinstellungen.....	8
Abbildung 4-1: Befehl „Vorwärts fahren“	13
Abbildung 4-2: Befehl „Rückwärts fahren“	13
Abbildung 4-3: Befehl „Drehung links“.....	13
Abbildung 4-4: Befehl „Drehung rechts“.....	14
Abbildung 4-5: Befehl „Einzelnen Motor ansteuern“	14
Abbildung 4-6: Befehl „Servo ansteuern“	14
Abbildung 4-7: Klassendesign der Klasse „Befehle“.....	14
Abbildung 4-8: Robot Remote Control 2010, Eingabe von Befehlen.....	17
Abbildung 4-9: Unterfenster zur Vergabe von Namen für Motoren und Servos	23
Abbildung 5-1: Kamera – Parametrierungsfenster.....	25
Abbildung 5-2: Unterfenster für die Eingabe des neuen Threshold.....	28
Abbildung 5-3: Auswertung der Kameradaten.....	29
Abbildung 6-1: Fenster für Sendelisten senden und Daten empfangen	30

TABELLENVERZEICHNIS

Tabelle 1-1: Spielelemente und ihre Punkte.....	1
Tabelle 3-1: ASCII – Tabelle Hex - codiert	4
Tabelle 3-2: Baudraten und ihr zugehörige Bitlänge.....	6
Tabelle 3-3: Kontaktbelegung der D-Sub Stecker.....	7
Tabelle 4-1: ASCII – Tabelle Dezimal - codiert	21

FORMELVERZEICHNIS

Formel 3-1: Benötigte Zeit um ein ASCII – Zeichen zu übertragen	6
Formel 3-2: Benötigte Zeit um einen Befehl zu übertragen	6
Formel 3-3: Anzahl übertragbarer Befehle pro Sekunde.....	6
Formel 4-1: TickFrequency – Berechnung ohne Runden.....	18
Formel 4-2: TickFrequency – Berechnung mit Runden	18
Formel 4-3: Berechnung der LargeChange – Eigenschaft.....	19

1 MOTIVATION

Seit dem Jahr 2007 nimmt das RoboRacingTeam der Fachhochschule Wels an dem internationalen Roboterwettbewerb EUROBOT^{open} teil.

In den letzten Jahren konnten immer beachtliche Erfolge verbucht werden. Um an diese Erfolge auch dieses Jahr anzuknüpfen, ist es von großer Bedeutung den Roboter in der Inbetriebnahme und Testphase so gut wie möglich abzustimmen. Für diesen Zweck wurde ein Programm entwickelt, welches das Testen des Roboters vereinfachen soll.

1.1 EUROBOT^{open}¹

Die EUROBOT^{open} findet jedes Jahr Ende Mai statt. Teilnahmeberechtigt sind Teams aus allen Ländern.

Charakteristisch für die EUROBOT^{open} ist, dass sich die Aufgabenstellungen und die Regeln Jahr für Jahr ändern. Die Aufgabenstellung und die Regeln werden jedes Jahr Ende September bekannt gegeben, wodurch sich eine Entwicklungszeit von 8 Monaten ergibt. Grundlegend ist, dass die Entwicklungszeit für alle Teams gleich ist. Lediglich die Größe des Spieltisches mit 210cm x 300cm und die Spieldauer mit 90sec. bleiben jedes Jahr gleich.

1.1.1 EUROBOT^{open} Thema 2010²

Das Thema der EUROBOT^{open} 2010 ist „We feed the world“. Aufgabe des Roboters ist es, drei verschiedene Spielelemente, welche drei verschiedene Nahrungsmittel darstellen, aufzunehmen und zu einer Entladestelle zu bringen. Jedes Spielelement bringt verschiedene Punkte. Je mehr Gewicht das Element hat, und desto schwerer es aufzunehmen ist, desto mehr Punkte bringt es. Tabelle 1-1 zeigt eine Übersicht der Spielelemente, welche Nahrungsmittel sie darstellen und wie viele Punkte sie bringen.

Nahrungsmittel	Spielelement	Gewicht (g) / Punkte
Tomaten	Bälle / Rot	je 150
Maiskolben	Zylinder	je 250
Orangen	Bälle / Orange	je 300

Tabelle 1-1: Spielelemente und ihre Punkte

¹ [eurobot, 2010]

² [Rules E2010, 2009]

Der Roboter, der in 90 sec. die meisten Punkte sammelt und auch bei der Entladestelle auslehrt, gewinnt. Abbildung 1-1 zeigt die Verteilung der Spielelemente auf dem Spielfeld.

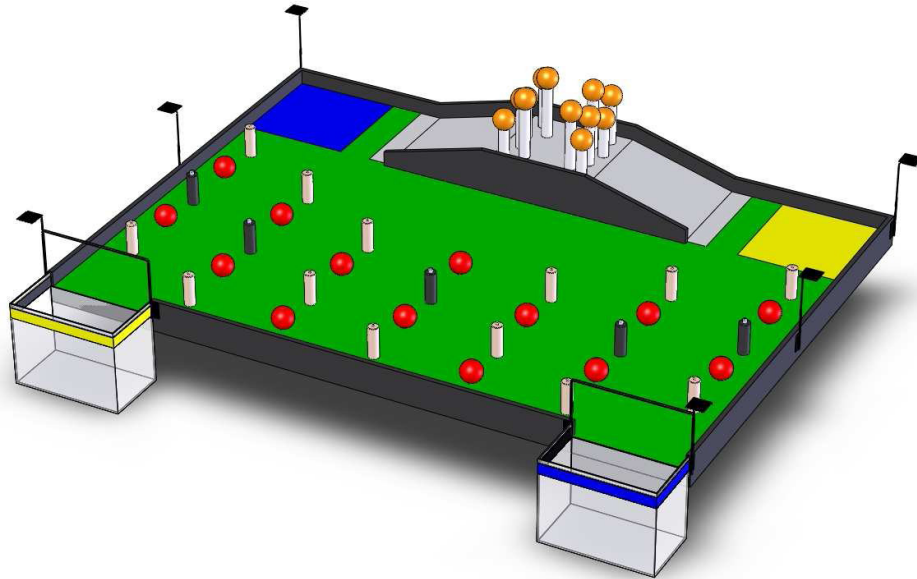


Abbildung 1-1: Spielfeld EUROBOT^{open} Thema 2010

Wie aus Abbildung 1-1 ersichtlich gibt es zwei Spielfarben, Gelb und Blau. Die Entladestellen sind jeweils gegenüber der Startzone.

Eine Schwierigkeit besteht darin, dass die Orangen, welche die meisten Punkte bringen, auf einem erhöhten Podest auf „Bäumen“ platziert sind. Der Roboter muss einerseits die Steigung der Rampe überwinden und andererseits einen Mechanismus besitzen die Orangen von den Bäumen zu holen. Eine weitere Schwierigkeit besteht darin, dass es schwarze und weiße Zylinder am Spielfeld gibt. Während die weißen Zylinder zum Einsammeln sind und dadurch Punkte bringen, sind die schwarzen am Tisch festgeschraubt und bringen keine Punkte. Viel mehr besteht die Gefahr, dass der Roboter an den schwarzen Zylindern hängen bleibt oder durch einen hängen gebliebenen gegnerischen Roboter blockiert wird. Weiters ist die Aufnahme der Tomaten relativ schwierig, da sie frei herumrollen können und ihre Position dadurch nicht fix definiert ist.

2 BENUTZEROBERFLÄCHE

2.1 Konzept

In Anlehnung an das bereits bestehende Programm entwickelt in der grafischen Programmiersprache LAB-View wurden für die Benutzeroberfläche folgende relevanten Objekte festgelegt.

- ✓ Einfache Eingabe der Befehle für die Aktoren mittels Schieberegler und händische Eingabe.
- ✓ Maximalwert für Befehle soll zur Programmlaufzeit einstellbar sein.
- ✓ Möglichkeit mehrere Aktorenbefehle zu einer Ablaufkette zu vereinen und diese auch abzuspeichern.
- ✓ Daten, welche vom Roboter geschickt werden, ausgeben.
- ✓ Kameras zur Bestimmung der Spieltischkonfiguration einlesen und auswerten.
- ✓ Einfache, und vor allem schnelle Parametrierung der Kameras sollte möglich sein.
- ✓ Kommunikation über die serielle Schnittstelle.

Abbildung 2.1 zeigt einen Ausschnitt aus dem bestehenden Programm „Robot communication 2009“, an dessen Anlehnung das neue Programm erstellt werden soll.

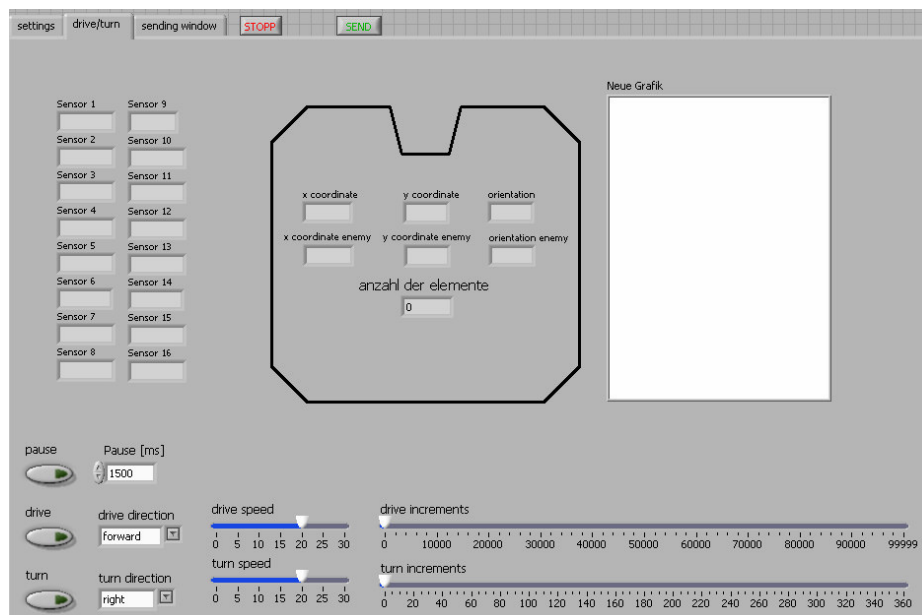


Abbildung 2-1: Robot communication 2009

3 Die serielle Schnittstelle³

Wie in den Anforderungen beschrieben, erfolgt die Kommunikation über ein Funkmodul⁴ welches an der seriellen Schnittstelle angeschlossen ist. Im Folgenden werden die serielle Schnittstelle und ihre Einbindung in das Programm beschrieben.

3.1 Grundlagen⁵

Die serielle Schnittstelle wurde 1962 für die Datenübertragung über die Telefonleitung entwickelt. Wie der Name sagt, werden die Daten seriell, d.h. nacheinander übertragen. Dabei wird zu Beginn der Datenübertragung ein Start-Bit, danach bis zu 8 Daten-Bits gefolgt von 1, 1.5 oder 2 Stopp-Bits gesendet. Zur Kontrolle der Datenübertragung kann vor dem Stopp – Bit noch ein Paritätsbit eingeschoben werden. Dieses kann entweder Even (gerade) oder Odd (ungerade) sein. Bei einer Even – Parity wird das Paritätsbit gesetzt wenn die Anzahl der in den Daten übertragenen High – Bits ungerade ist.

Durch 8 Datenbits lässt sich bei jeder Übertragung ein ASCII – Zeichen Hex – codiert übertragen.

HEX		HEX		HEX		HEX		HEX		HEX		HEX		HEX	
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	\$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(38	8	48	H	58	X	68	h	78	x
09	TAB	19	EM	29)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Tabelle 3-1: ASCII – Tabelle Hex - codiert⁶

³ [Tietze & Schenk, 1999]

⁴ [Zauner, 2009]

⁵ [rn-wissen, 2010]

⁶ [Tietze - Schenk, 1999]

3.2 Datenübertragung

Am Anfang der Übertragung ist die Übertragungsstrecke logisch auf 1. Durch das Start – Bit (logisch 0) wird die Übertragung gestartet. Nach den Datenbits folgt je nach Einstellung das Paritätsbit. Beendet wird die Übertragung mit dem Stopp – Bit, welches die Übertragungsstrecke wieder auf logisch 1 setzt.

Bsp.:

Übertragenes Zeichen: G → Hex: 47
Parity: odd → ungerade
Stopp – Bit 1

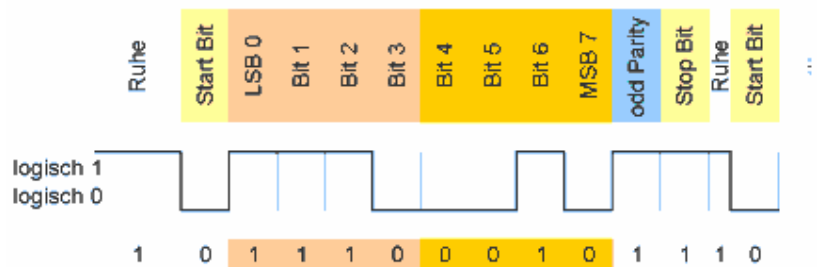


Abbildung 3-1: Übertragung des Zeichens G über die serielle Schnittstelle⁷

Abbildung 3-1 zeigt die Übertragung des Zeichens „G“. Die Übertragung beginnt mit dem Start – Bit. Mit den ersten 4 Datenbits wird „7“ übertragen, mit den letzten 4 Datenbits „4“. Das übertragene Zeichen ist HEX = 47 was die ASCII – Codierung für das Zeichen „G“ ist. Da insgesamt 4 Datenbits High sind, ist die Summe der High – Datenbits gerade. Wenn die Parität ungerade gesetzt ist wird auch das Paritätsbit gesetzt. Beendet wird die Übertragung mit dem Stopp – Bit.

3.3 Datenrate⁸

Die Übertragung funktioniert nur, wenn Sender und Empfänger die Bits mit derselben Geschwindigkeit lesen bzw. schreiben. Um dies zu gewährleisten ist es notwendig auf beiden Seiten dieselbe Datenrate einzustellen.

⁷ [rn-wissen, 2010]

⁸ [sprut, 2010]

Die Datenrate wird in Baud (= Bits pro Sekunde) angegeben. Gezählt werden alle Bits inklusive Start- und Stoppbit, dadurch berechnet sich die Bitlänge mit 1/Baud.

Tabelle 3-2 zeigt gängige Baudraten mit ihren Bitlängen.

Baudrate (Bits/sec)	2400	9600	19200	38400	57600	115200
Bitlänge (sec)	417 μ	104 μ	52,08 μ	26,04 μ	17,36 μ	8,68 μ

Tabelle 3-2: Baudraten und ihr zugehörige Bitlänge

Bsp.:

Baudrate: 19200
Parity: None
Stoppbits: 1
Datenbits: 8

Obige Anforderungen stellt die Situation, wie Sie im Normalfall beim Roboter gegeben ist, dar. Durch diese Anforderungen ergibt sich, dass pro übertragenen Zeichen 10 Bits notwendig sind (1 Startbit + 8 Datenbits + 1 Stoppbit). Der längste Befehl für den Roboter hat 13 Zeichen (Einzeln Motor ansteuern).

Bei einer Übertragungsrate von 19200 Bits/sec ergibt sich eine Zeit von

$$\begin{aligned} \text{Baud} * \text{Anzahl Bits} &= \text{Zeit pro Zeichen} \\ 52,08\mu s * 10 &= 520,8\mu s \end{aligned}$$

Formel 3-1: Benötigte Zeit um ein ASCII – Zeichen zu übertragen

um ein Zeichen zu übertragen.

Daraus ergibt sich für den längsten Befehl (worst case) eine Übertragungszeit von:

$$\begin{aligned} \text{Zeit pro Zeichen} * \text{Anzahl Zeichen} &= \text{Zeit pro Befehl} \\ 520,8\mu s * 13 &= 6,77ms \end{aligned}$$

Formel 3-2: Benötigte Zeit um einen Befehl zu übertragen

Dadurch berechnen sich die pro Sekunde übertragbaren Befehle zu:

$$\begin{aligned} \frac{1}{\text{Zeit pro Befehl}} &= \text{Anzahl Befehle pro Sekunde} \\ \frac{1s}{6,77ms} &\approx 147 \end{aligned}$$

Formel 3-3: Anzahl übertragbarer Befehle pro Sekunde

3.4 Spannungspegel

Die serielle Schnittstelle arbeitet in einem Bereich von -15V bis +15V. Der Bereich von -15V bis -3V als logisch low (0) und der Bereich von +3V bis +15V als logisch high (1) interpretiert wird. Jedoch werden die Daten mit negativer Logik gesendet. Dadurch werden Daten beim Senden normalerweise mit -12V (für high) und +12V (für low) ausgegeben. Der Empfänger interpretiert alles unter -3V als logisch high und alles über +3V als logisch low.

3.5 Anschlussbelegung⁹

Abbildung 3-2 bis 3-5 zeigen die unterschiedlichen Stecker mit ihren Pinbelegungen.

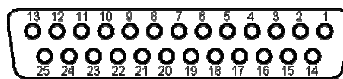


Abbildung 3-2: D-Sub Stecker 25 polig

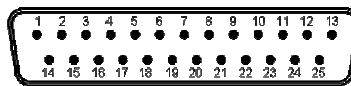


Abbildung 3-3: D-Sub Buchse 25 polig

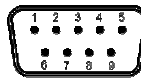


Abbildung 3-4: D-Sub Stecker 9 polig

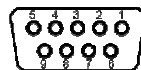


Abbildung 3-5: D-Sub Buchse 9 polig

Aus Tabelle 3-3 ist ersichtlich welcher Pin mit welchem Signal belegt ist.

Signalname		D-Sub 9 polig	D-Sub 25 polig
DCD	Trägerpegel (Data Carrier Detect)	1	8
RxD	Empfangsdaten (Receive Data)	2	3
TxD	Sendedaten	3	2
DTR	Terminal bereit (Data Terminal Ready)	4	20
GND	Signalmasse	5	7
DSR	Datenübertragung bereit (Data Set Ready)	6	6
RTS	Senden einschalten (Request to Send)	7	4
CTS	Senden bereit (Clear to Send)	8	5
RI	Ankommender Anruf	9	22
GND	Gehäusemasse	-	1

Tabelle 3-3: Kontaktbelegung der D-Sub Stecker

⁹ [rn-wissen, 2010]

3.6 Implementierung

3.6.1 Einstellungen

Stoppbits und Datenbits werden im Programm fest vorgegeben. Das Stoppbit wurde auf 1 festgelegt, die Datenbits auf 8. Die Einstellungen des Ports, der Baudrate und der Parity können in einem Unterfenster vorgenommen werden.

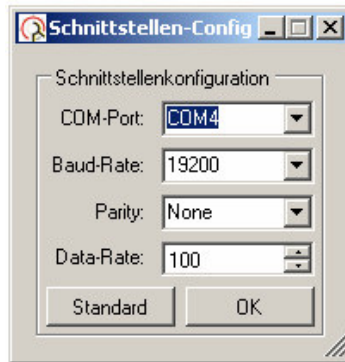


Abbildung 3-6: Unterfenster für die Schnittstelleneinstellungen

Die Einstellmöglichkeit der Data-Rate gibt an, wie schnell hintereinander die einzelnen Befehle, in ms, gesendet werden.

Das abspeichern der Einstellungen geschieht über ein User Setting¹⁰. Dadurch ist es auf einfache Weise möglich die Daten auch nach dem Beenden des Programms dauerhaft abzuspeichern. Weiters kann an verschiedenen Stellen des Programms auf das User Setting zugegriffen werden. Dadurch ist auch der Datenaustausch zwischen den Fenstern realisiert. Beim öffnen des Fensters werden die Daten aus dem User Setting gelesen. Zuvor, beim Initialisieren des Fensters, werden die auf den PC verfügbaren COM – Ports in die Combobox geladen.

```

/// <summary>
/// Erforderliche Methode für die Designerunterstützung
/// MC 19.01.2010
/// </summary>
public frm_Config()
{
    InitializeComponent();
    // Alle verfügbaren seriellen Schnittstellen laden
    string[] portName = SerialPort.GetPortNames();
    // Verfügbare Schnittstellen der Auswahl hinzufügen
    foreach (string name in portName)
    {
        this.cbx_Com.Items.Add(name);
    }
    this.SetDefault();
}

```

¹⁰ [codeproject, 2010]

Nachdem die COM – Ports geladen sind werden die gespeicherten Einstellungen geladen.

```
/// <summary>
/// Setzt die Standardwerte für die Schnittstelleneinstellung
/// MC 19.01.2010
/// </summary>
private void SetDefault()
{
    string com = COM_Setting.Default.PortNr;
    string baud = COM_Setting.Default.Baudrate;
    string par = COM_Setting.Default.Parity;
    // COM-Daten Einstellungen
    for (int i = 0; i < this.cbx_Com.Items.Count; i++)
    {
        this.cbx_Com.SelectedIndex = i;
        if (this.cbx_Com.Text.ToString() == com)
        {
            break;
        }
        // Falls gespeicherte Einstellung nicht gefunden wird
        // wird Standardeinstellung angezeigt
        if (i + 1 == this.cbx_Com.Items.Count)
        {
            this.cbx_Com.SelectedIndex = 0;
            break;
        }
    }
    // Baudrate Einstellungen
    for (int i = 0; i < this.cbx_Baud.Items.Count; i++)
    {
        this.cbx_Baud.SelectedIndex = i;
        if (this.cbx_Baud.Text.ToString() == baud)
        {
            break;
        }
        // Falls gespeicherte Einstellung nicht gefunden wird
        // wird Standardeinstellung angezeigt
        if (i + 1 == this.cbx_Baud.Items.Count)
        {
            this.cbx_Baud.SelectedIndex = 6;
            break;
        }
    }
    // Parity-Bit Einstellungen
    for (int i = 0; i < this.cbx_Parity.Items.Count; i++)
    {
        this.cbx_Parity.SelectedIndex = i;
        if (this.cbx_Parity.Text.ToString() == par)
        {
            break;
        }
        // Falls gespeicherte Einstellung nicht gefunden wird
        // wird Standardeinstellung angezeigt
        if (i + 1 == this.cbx_Parity.Items.Count)
        {
            this.cbx_Parity.SelectedIndex = 2;
            break;
        }
    }
    this.nud_DataRate.Value = COM_Setting.Default.DataRate;
}
```

Beim Beenden des Programms werden die Einstellungen gespeichert und das Unterfenster geschlossen. Nach dem Schließen des Unterfensters werden im Hauptfenster die Einstellungen für die Schnittstelle, neben einigen anderen Einstellungen, übernommen.

```
/// <summary>
/// Methode um die Schnittstellenkonfiguration zuzuweisen
/// MC 19.01.2010
/// </summary>
private void SetComPort()
{
    // Variablen für die COM-Port Einstellungen
    string comPort = COM_Setting.Default.PortNr;
    int baudRate = Convert.ToInt32(COM_Setting.Default.Baudrate);
    Parity par = Parity.None;
    StopBits stopBit = StopBits.One;
    int anzahlBits = 8;
    string parity = "None";
    sp_Connection.Encoding = System.Text.Encoding.Default;
    sp_Connection.Handshake = Handshake.None;
    // Den Timeout auf 1000ms stellen damit Daten nicht ewig gelesen bzw.
    // gesendet werden
    sp_Connection.ReadTimeout = 1000;
    sp_Connection.WriteTimeout = 1000;
    // Wenn der TAB Kamera gewählt ist wird die Baudrate höher gestellt
    // ToDo_MC: Nur für Testzwecke. Im Bewerb wird 19200 eingestellt.
    if (this.tcon_Main.SelectedIndex == TAB_KAMERA ||
        this.tcon_Main.SelectedIndex == TAB_GUI)
    {
        //baudRate = 115200;
    }
    // Parity auslesen
    switch (COM_Setting.Default.Parity)
    {
        case "Even":
        {
            par = Parity.Even;
            break;
        }
        case "Odd":
        {
            par = Parity.Odd;
            break;
        }
        default:
        {
            par = Parity.None;
            break;
        }
    }
    parity = COM_Setting.Default.Parity;
    // Schnittstelleneinstellungen zuweisen
    this.sp_Connection.PortName = comPort;
    this.sp_Connection.Parity = par;
    this.sp_Connection.BaudRate = baudRate;
    this.sp_Connection.DataBits = anzahlBits;
    this.sp_Connection.StopBits = stopBit;
    // Gewählte Einstellungen anzeigen
    this.tssl_COM.Text = comPort.ToString();
    this.tssl_Baud.Text = baudRate.ToString();
    this.tssl_Parity.Text = parity.ToString();
    this.sp_Connection.Open();
}
}
```

3.6.2 Daten senden

Daten senden über die serielle Schnittstelle erfolgt an mehreren Stellen im Programm.

Deshalb wurde für das Senden eine eigene Methode geschrieben.

```

/// <summary>
/// Befehl an die Schnittstelle schreiben
/// MC 19.01.2010
/// </summary>
/// <param name="message">Zu schreibender Befehl</param>
private void WriteCOM(string message)
{
    try
    {
        this.sp_Connection.Write(message);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fehler beim Senden!!! Fehlertext: " +
            ex.ToString());
    }
}

```

3.6.3 Daten empfangen

Sobald Daten an der seriellen Schnittstelle anstehen wird das Ereignis DataReceived ausgelöst. Da die empfangenen Daten je nach Programmstatus an verschiedenen Stellen des Programms verarbeitet werden, wurde eine eigene Methode für das Empfangen der Daten geschrieben. In dieser wird ein Flag abgefragt, das festlegt wie die Daten verarbeitet werden müssen.

3.6.3.1 Daten sind für Kamerakonfiguration bestimmt:

In diesem Fall werden die empfangenen Daten auf das Output – Fenster des Kamera – Konfigurationsfenster (siehe 5.2) geschrieben. Es wurde festgelegt, dass die gesendeten Daten der Kamera mit # beginnen und mit * enden. Alle anderen Daten werden in das Empfangfenster (siehe 6.2) geschrieben. Da dies an verschiedenen Stellen des Programms geschieht wurde hierfür eine eigene Methode (WriteToTerminal) geschrieben.

3.6.3.2 Daten sind für Kameraauswertung bestimmt:

In diesem Fall werden die empfangenen Daten sofort ausgewertet und das Bild mit der richtigen Spieltischkonfiguration wird geladen. Auch hier werden Daten welche vor dem # - Zeichen oder nach dem * - Zeichen kommen wieder ins Empfangfenster geschrieben.

3.6.3.3 Daten für keine bestimmte Stelle bestimmt:

In allen anderen Fällen werden die Daten in das Empfangfenster geschrieben.

```
/// <summary>
/// Methode zum einlesen der COM-Schnittstelle
/// MC 27.01.2010
/// </summary>
private void ReadCOM()
{
    string result = string.Empty;
    try
    {
        switch (this.spSender)
        {
            // Sender ist TAB_Kamera
            case (1):
            {
                result = this.sp_Connection.ReadTo("#");
                this.WriteToTerminal(result);
                result = this.sp_Connection.ReadTo("*");
                if (result != string.Empty)
                {
                    this.tbx_KamOutput.AppendText(result +
                        "\r\n");
                    this.tbx_KamOutput.ScrollToCaret();
                }
                result = this.sp_Connection.ReadExisting();
                this.WriteToTerminal(result);
                break;
            }
            // Sender ist TAB_GUI
            case (2):
            {
                result = this.sp_Connection.ReadTo("#");
                this.WriteToTerminal(result);
                result = this.sp_Connection.ReadTo("*");
                if (result != string.Empty)
                {
                    string[] split = result.Split(new char[]
                    { ' ' });
                    string load = split[1] + split[2] +
                    split[3] + split[4] + "01";
                    this.pbx_Spielfeld.ImageLocation =
                    ("CornConfig\\" + load + ".emf");
                }
                result = this.sp_Connection.ReadExisting();
                this.WriteToTerminal(result);
                break;
            }
            // Kein Sender vorhanden
            default:
            {
                result = this.sp_Connection.ReadExisting();
                this.WriteToTerminal(result);
                break;
            }
        }
        this.spSender = 0
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fehler beim lesen! Fehlertext: " +
            ex.Message);
        this.spSender = 0;
    }
}
```

4 Befehle

4.1 Konzept

Der Aufbau der Befehle wurde von dem bereits bestehenden System¹¹ des Vorjahres übernommen. Nach diesen Vorgaben besteht ein Befehl aus:

- ✓ einem Start – Byte (# → ASCII 0x23)
- ✓ das Modus – Byte
- ✓ die Daten
- ✓ ein Stopp – Byte (* → ASCII 0x2A)

Die Daten der Befehle werden einheitenlos übertragen. Es muss am Roboter festgelegt werden um welche Einheiten es sich handelt.

4.1.1 Befehl „Vorwärts fahren“

Start Byte	Modus Byte	Daten: Geschwindigkeit		Daten: Distanz					Stopp Byte
#	F	9	9	9	9	9	9	9	*

Abbildung 4-1: Befehl „Vorwärts fahren“

4.1.2 Befehl „Rückwärts fahren“

Start Byte	Modus Byte	Daten: Geschwindigkeit		Daten: Distanz					Stopp Byte
#	B	9	9	9	9	9	9	9	*

Abbildung 4-2: Befehl „Rückwärts fahren“

4.1.3 Befehl „Drehung links“

Start Byte	Modus Byte		Daten: Geschwindigkeit		Daten: Winkel					Stopp Byte
#	T	l	9	9	9	9	9	9	9	*

Abbildung 4-3: Befehl „Drehung links“

¹¹ [Zauner, 2009]

4.1.4 Befehl „Drehung rechts“

Start Byte	Modus Byte		Daten: Geschwindigkeit		Daten: Winkel					Stopp Byte
#	T	r	9	9	9	9	9	9	9	*

Abbildung 4-4: Befehl „Drehung rechts“

4.1.5 Befehl „Einzelnen Motor ansteuern“

Dieser Befehl wurde im Vergleich zum Vorjahr etwas verändert. Die Daten für die Distanz wurden neu hinzugefügt. Wenn der anzusteuern Motor einen Inkrementalgeber hat, dann kann man diesen Motor eine bestimmte Distanz fahren lassen. Im Falle, dass er keinen Inkrementalgeber hat, werden die Daten für die Distanz einfach ignoriert.

Start Byte	Modus Byte	Daten: Motor Nr.		Daten: Richtung	Daten: Geschwindigkeit		Daten: Distanz					Stopp Byte
#	R	1	6	r/l	9	9	9	9	9	9	9	*

Abbildung 4-5: Befehl „Einzelnen Motor ansteuern“

4.1.6 Befehl „Servo ansteuern“

Start Byte	Modus Byte	Daten: Servo Nr.		Daten: Winkel			Stopp Byte
#	S	1	6	9	9	9	*

Abbildung 4-6: Befehl „Servo ansteuern“

4.2 Implementierung der Befehle

Um die einzelnen Befehle im Programm besser handhaben zu können wurde hierfür eine Klasse geschrieben. Damit ist es möglich, mehrere Befehle in einer Liste zusammenzuführen, was das Verwalten einer Befehlskette erleichtert. Weiters ist eine Erweiterung um einen Befehlssatz damit leicht zu realisieren da alle Befehle an einem zentralen Ort verwaltet werden. Abbildung 4-7 zeigt das einfache Klassendesign.

Befehle
-bez string
-bef string
+get Bezeichnung() : string
+get Befeh() : string

Abbildung 4-7: Klassendesign der Klasse „Befehle“

Die Klasse besteht aus zwei Membervariablen „bez“ und „bef“ jeweils vom Typ string. Auf der Variable „bez“ steht die Bezeichnung des Befehls um die Befehlskette für den Benutzer übersichtlicher zu gestalten. Auf der Variable „bef“ steht der Befehl in Form wie oben beschrieben. Beide Variablen können mittels get - Eigenschaft gelesen werden. Um die Handhabung der einzelnen Befehle so einfach wie möglich zu halten wurde für jeden Befehl ein Konstruktor geschrieben.

4.2.1 Konstruktor für Vorwärts – bzw. Rückwärts fahren

```
/// <summary>
/// Konstruktor für Befehl Vorwärts- bzw. Rückwärtsfahren
/// 30.12.2009 MC
/// </summary>
/// <param name="vorwaerts">True für Vorwärts
///                               False für Rückwärts</param>
/// <param name="geschwindigkeit">Daten für Geschwindigkeit</param>
/// <param name="distanz">Daten für Distanz</param>
public Befehle(bool vorwaerts, int geschwindigkeit, int distanz)
{
    string mod = "R";
    this.bez = "Rückwärts fahren";
    if (vorwaerts)
    {
        mod = "F";
        this.bez = "Vorwärts fahren";
    }
    this.bef = "#" + mod + geschwindigkeit.ToString("00") +
    distanz.ToString("00000") + "*";
}
```

4.2.2 Konstruktor für Drehung links bzw. rechts

```
/// <summary>
/// Konstruktor für Befehl links- bzw. rechts drehen
/// 30.12.2009 MC
/// </summary>
/// <param name="richtung">True für rechts drehen
///                               False für links drehen</param>
/// <param name="geschwindigkeit">Daten für Geschwindigkeit</param>
/// <param name="winkel">Daten für Drehwinkel</param>
public Befehle(int geschwindigkeit, int winkel, bool richtung)
{
    string mod = "Tl";
    this.bez = "Drehung nach links";
    if (richtung)
    {
        mod = "Tr";
        this.bez = "Drehung nach rechts";
    }
    this.bef = "#" + mod + geschwindigkeit.ToString("00") +
    winkel.ToString("00000") + "*";
}
```

4.2.3 Konstruktor für einzelnen Motor ansteuern

```

/// <summary>
/// Konstruktor für den Befehl zum Ansteuern eines Motors
/// 30.12.2009 MC
/// </summary>
/// <param name="motorNr">Nummer des anzusteuernenden Motors</param>
/// <param name="richtung">True für Rechtslauf
///     False für Linkslauf</param>
/// <param name="geschwindigkeit">Daten für Geschwindigkeit</param>
/// <param name="distanz">Daten für Distanz</param>
/// <param name="motorbezeichnung">Bezeichnung des Motors</param>
public Befehle(int motorNr, bool richtung, int geschwindigkeit, int
distanz, string motorbezeichnung)
{
    string mod = "R";
    this.bez = motorbezeichnung + "-Motor ansteuern";
    if (richtung)
    {
        this.bef = "#" + mod + motorNr.ToString("00") + "r" +
geschwindigkeit.ToString("00") + distanz.ToString("00000") +
"*";
    }
    else
    {
        this.bef = "#" + mod + motorNr.ToString("00") + "l" +
geschwindigkeit.ToString("00") + distanz.ToString("00000") +
"*";
    }
}

```

4.2.4 Konstruktor für Servo ansteuern

```

/// <summary>
/// Konstruktor für den Befehl zum Ansteuern eines Servos
/// 02.01.2010 MC
/// </summary>
/// <param name="servoNr">Nummer des anzusteuernenden Servos</param>
/// <param name="winkel">Wert auf welche Position der Servo fahren
/// soll</param>
/// <param name="servobezeichnung">Bezeichnung des Servos</param>
public Befehle(int servoNr, int winkel, string servobezeichnung)
{
    string mod = "S";
    this.bez = servobezeichnung + "-Servo ansteuern";
    this.bef = "#" + mod + servoNr.ToString("00") +
winkel.ToString("000") + "*";
}

```

4.2.5 Konstruktor für Befehle die schon bekannt sind

Dieser Konstruktor wurde für den Fall gemacht, dass der Befehl schon in der Form wie unter 4.1 beschrieben vorliegt (das heißt, dass der Befehl schon einmal erzeugt wurde und z.B.: von einer gespeicherten Befehlsliste ausgelesen wird).

```
/// <summary>  
/// Konstruktor für Befehle die schon bekannt sind  
/// Wenn die Befehle von einer Datei gelesen werden  
/// 22.01.2010 MC  
/// </summary>  
/// <param name="befehl">Der Befehl in der richtigen Codierung</param>  
/// <param name="bezeichnung">Die zugehörige Bezeichnung</param>  
public Befehle(string befehl, string bezeichnung)  
{  
    this.bef = befehl;  
    this.bez = bezeichnung;  
}
```

4.2.6 Konstruktor für eine Sendepause

Um den Roboter Zeit zur Abarbeitung der Befehle zu geben ist es notwendig Pausen zwischen dem Senden der Befehle zu haben. Für diesen Fall wurde ein Befehl Pause angelegt. Auf diesen Fall wird unter 6.1 noch genauer eingegangen.

```
/// <summary>  
/// Konstruktor für eine Pause  
/// 22.01.2010 MC  
/// </summary>  
/// <param name="pause">Zeit in ms der Pause</param>  
public Befehle(decimal pause)  
{  
    this.bef = pause.ToString();  
    this.bez = this.bez + "ms Pause";  
}
```

4.3 Eingabe der Befehle

Abbildung 4-8 zeigt den Entwurf der Oberfläche für die Befehlseingabe die alle unter 2 vorgegebenen Eigenschaften erfüllt.

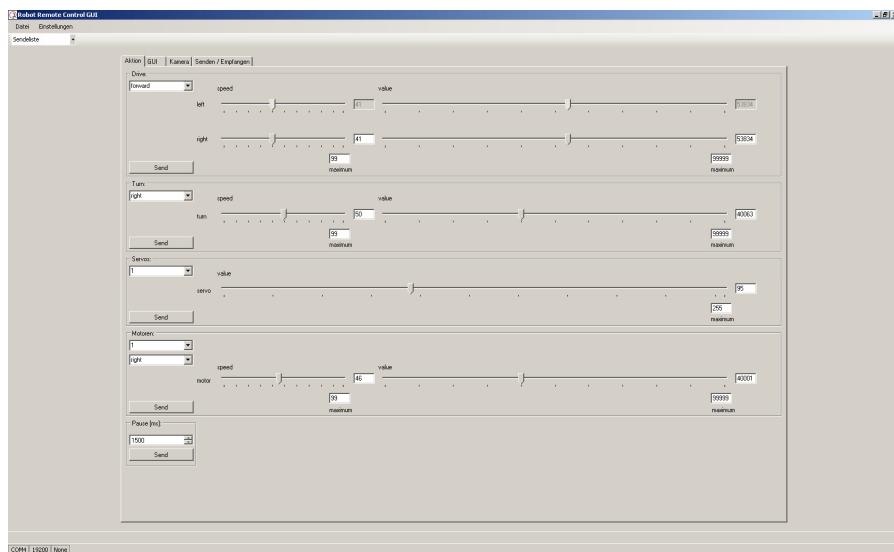


Abbildung 4-8: Robot Remote Control 2010, Eingabe von Befehlen

Zur Eingabe der Daten wurden Schiebepalken (Track-Bars) und Textboxen verwendet. Die Eingabe der Maximalwerte erfolgt nur über Textboxen. Als absolutes Maximum wurden für Geschwindigkeit 99(m/s), für Distanz 99999(m) und für Winkel 999(°) genommen. Dies war notwendig da bei den einzelnen Befehlen nicht mehr Platz für diese Daten vorgesehen ist (Überprüfung ob Eingabe kleiner/gleich Maximum ist siehe 4.3.2.1).

4.3.1 Eingabe der Maximalwerte

Da die Maximalwerte einen relativ großen Einstellbereich haben (im Fall Distanz 1 – 99999), sollte bei den Track-Bars die LargeChange – und TickFrequency – Eigenschaften individuell angepasst werden. Ändert man den Maximalwert wird die LargeChange – und TickFrequency – Eigenschaft neu berechnet. Weiters wird kontrolliert ob der aktuell eingegebene Wert kleiner ist als der neue Maximalwert und bei Bedarf ebenfalls angepasst. Weiters muss überprüft werden ob nur Zahlenwerte eingegeben wurden. Diese Prüfung wird unter 4.3.2.1 behandelt.

4.3.1.1 Änderung der TickFrequency – Eigenschaft

Bei der TickFrequency wurde 1/10 des Maximalwertes genommen. In diesem Bereich kann noch gut abschätzen werden wo man sich beim verstellen des Schiebepalkens befindet, andererseits wird die Anzeige nicht durch zu viele Unterteilungen unübersichtlich.

Da der Maximalwert und die TickFrequency vom Typ Integer sind ist eine Integer-Division unumgänglich. Um ein kaufmännisches Runden zu gewährleisten wird der Maximalwert vor der Division um 5 erhöht.

Bsp.:

Maximum = 48

ohne Runden:

$$TickFrequency = \frac{Maximum}{10} = \frac{48}{10} = 4,8 = 4$$

Formel 4-1: TickFrequency – Berechnung ohne Runden

mit Runden:

$$TickFrequency = \frac{Maximum + 5}{10} = \frac{48 + 5}{10} = \frac{53}{10} = 5,3 = 5$$

Formel 4-2: TickFrequency – Berechnung mit Runden

```
// Werte für drehen setzten  
// 29.12.2009 MC  
this.tbr_turnSpeed.TickFrequency = (this.tbr_turnSpeed.Maximum + 5) / 10;
```

4.3.1.2 Änderung der LargeChange – Eigenschaft

Die LargeChange – Eigenschaft ist jener Wert, der zum aktuellen Wert hinzugefügt oder abgezogen wird wenn mit der Maus auf den Schiebebalken geklickt wird, und beträgt Standardmäßig die Hälfte der TickFrequency. Auch diese Eigenschaft muss beim Verstellen des Maximalwertes dynamisch eingestellt werden. Hier handelt es sich ebenso um eine Integer – Division, jedoch wurde bei dieser Eigenschaft der Rundungsfehler vernachlässigt.

Bsp.:

Maximum = 48

$$\begin{aligned} \text{LargeChange} &= \frac{\text{TickFrequency}}{2} \\ \text{TickFrequency} &= \frac{\text{Maximum} + 5}{10} = \frac{48 + 5}{10} = \frac{53}{10} = 5,3 = 5 \\ \text{LargeChange} &= \frac{5}{2} = 2,5 = 2 \end{aligned}$$

Formel 4-3: Berechnung der LargeChange – Eigenschaft

```
// Bei der LargeChange-Eigenschaft ist es mir egal wenn bei einer ungeraden
// Zahl die Kommastellen weggeschnitten werden und die kleiner Zahl
// genommen wird.
// MC 29.12.2009
this.tbr_turnSpeed.LargeChange = this.tbr_turnSpeed.TickFrequency / 2;
```

4.3.1.3 Maximalwerte überprüfen

Weiters wird kontrolliert ob der aktuell eingegeben Wert größer ist als der neue Maximalwert. In diesem Fall wird der eingegebene Wert auf den Maximalwert korrigiert.

```
/// <summary>
/// Nach einer Eingabe wird überprüft ob der Maximumwert nicht
/// überschritten wird
/// 30.12.2009 MC
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void tbr_turnSpeedMax_TextChanged(object sender, EventArgs e)
{
    int zahl = this.CheckTextBoxValue((TextBox)sender, TURN_SPEED_MAX);
    // Falls die eingestellte Zahl größer ist als die neue maximale Zahl
    // wird der Wert angepasst
    if (tbr_turnSpeed.Value > zahl)
    {
        tbr_turnSpeed.Value = zahl;
        tbr_turnSpeed.Text = zahl.ToString();
    }
    this.tbr_turnSpeed.Maximum = zahl;
    // TickFrequency und LargeChange – Eigenschaft neu berechnen
    this.SetTrackBarValues();
}
```

4.3.2 Eingabe der Daten

Die Eingabe der Daten kann auf zwei Arten erfolgen. Einerseits über die TrackBars oder als Zahlenwert in der Textbox. Beide Eingabearten sind miteinander gekoppelt, das heißt, die TrackBars werden automatisch an den neuen Wert angepasst wenn die Textbox geändert wird, und umgekehrt. Weiters muss bei der Textboxeingabe darauf geachtet werden das nur Zahlen eingegeben werden und das der Maximalwert nicht überschritten wird.

4.3.2.1 Überprüfung ob eingegebener Wert nicht größer Maximalwert ist

Da die Überprüfung, ob eine Eingabe kleiner/gleich dem Maximalwert ist, relativ häufig gemacht werden muss, wurde für diese Aufgabe eine Methode geschrieben.

```
/// <summary>
/// Überprüft ob die Eingabe in eine Textbox den Maximalwert nicht
/// überschreitet
/// MC 30.12.2009
/// </summary>
/// <param name="tbx"></param>
/// <param name="maxValue"></param>
/// <returns></returns>
private int CheckTextBoxValue(TextBox tbx, int maxValue)
{
    int value = 1;
    // Nur überprüfen wenn eine Eingabe vorhanden ist
    if (tbx.Text.Length != 0)
    {
        value = Convert.ToInt32(tbx.Text);
        // Wenn die Eingabe größer dem maximalen Wert ist
        // wird die Eingabe automatisch auf den Maximumwert korrigiert.
        if (value > maxValue)
        {
            tbx.Text = maxValue.ToString();
            value = maxValue;
        }
    }
    return value;
}
```

Die Konvertierung der Eingabe in eine Integerzahl würde, wenn es sich bei der Eingabe um keine Zahl handelt, zu einer Exception führen. Dieser Fall wird aber von vorn herein ausgeschlossen da bereits bei der Eingabe kontrolliert wird ob es sich um eine Zahl handelt (siehe 4.3.2.2).

4.3.2.2 Überprüfung ob eingegebener Wert eine Zahl ist¹²

Um dem Benutzer bei der Eingabe in eine TextBox nur Zahleneingaben zu erlauben wird jeder Tastendruck überprüft. Handelt es sich bei der Eingabe um eine Zahl oder um die Taste BackSpace(=Löschen – Taste) wird das Event ganz normal bearbeitet. In jedem anderen Fall jedoch wird e.Handled auf true gesetzt was dem Programm sagt, dass der Event schon behandelt wurde, ohne jedoch auf den Tastendruck zu reagieren. Da diese Überprüfung auch sehr häufig gemacht werden muss, wurde auch hierfür eine Methode geschrieben. Die Überprüfung ob die Eingabe eine Zahl ist, erfolgt indem überprüft wird, ob sich der ASCII – Wert der eingegebenen Zeichen zwischen 48 und 57 dezimal befindet. Dies ist, wie Tabelle 4-1 zeigt, der Bereich der numerischen Zeichen. ASCII 8 ist die Taste BackSpace.

```

/// <summary>
/// Überprüft ob die Eingabe eine Zahl ist
/// 30.12.2009 MC
/// </summary>
/// <param name="e"></param>
private void CheckTextBoxNumber(KeyPressEventArgs e)
{
    // Wenn die Eingabe keine Zahl oder BackSpace ist, wird das Event
    // nicht bearbeitet
    if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar != 8)
    {
        // e.Handled auf true um Programm zu sagen das Event schon
        // behandelt wurde.
        e.Handled = true;
    }
}

```

Dez		Dez		Dez		Dez		Dez		Dez		Dez		Dez	
0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	TAB	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Tabelle 4-1: ASCII – Tabelle Dezimal - codiert¹³

¹² [Microsoft, 2010]

4.3.3 Senden der Befehle

Laut Anforderungen sollen die Befehle entweder direkt gesendet werden können oder in einer Sendeliste zu einer Befehlskette zusammengefügt werden können. Um dies zu ermöglichen wurde eine ComboBox mit den Einträgen „Sendeliste“ und „Direkt senden“ in das Programm eingefügt. Mit einem Click auf den Button „Send“ des jeweiligen Befehls wird zuerst der Befehl erzeugt und dieser dann entweder direkt an die serielle Schnittstelle gesendet oder zu einer Liste mit Befehlen hinzugefügt. Da man im Vorfeld nicht sagen kann wie viele Befehle der Benutzer anlegen will, wurde auf den Einsatz eines Arrays verzichtet und stattdessen der Datentyp List verwendet. Dadurch braucht man sich über die Anzahl der Elemente keine Gedanken machen.

```
/// <summary>
/// Liste für die Befehle die in die Befehlsliste geschrieben werden
/// MC 29.12.2009
/// </summary>
private List<Befehle> Commands = new List<Befehle> ();

/// <summary>
/// Befehl für drehen bestimmen
/// 02.01.2010 MC
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_turn_Click(object sender, EventArgs e)
{
    // Richtung bestimmen
    bool richtung = false;
    if (cbx_turn.SelectedIndex == 0)
    {
        richtung = true;
    }
    Befehle turn = new Befehle(tbr_turnSpeed.Value, tbr_turnValue.Value,
    richtung);
    // Entweder sofort senden oder in Sendeliste schreiben
    if (tscbx_Sendeauswahl.SelectedIndex == 0)
    {
        // Zur Sendeliste hinzufügen
        this.Commands.Add(turn);
    }
    else
    {
        // Direkt senden
        this.WriteCOM(turn.Befehl);
    }
}
```

¹³ [Tietze - Schenk, 1999]

4.4 Vergabe von Motornamen

Da mit diesem Programm bis zu 16 verschiedene einzelne Motoren sowie 16 verschiedene Servos anzusteuern sind, ist es für die Übersicht von Vorteil, wenn man diesen Motoren Namen wie z.B.: Schieber geben kann. Die Bedienung des Programms und somit auch des Roboters wird dadurch erheblich erleichtert, da man sich nicht mehr die Nummern der einzelnen Motoren bzw. Servos merken muss.

Um dies zu ermöglichen wurde ein Unterfenster für die Vergabe von Namen für die einzelnen Motoren und Servos angelegt. Abbildung 4-9 zeigt dieses Unterfenster.

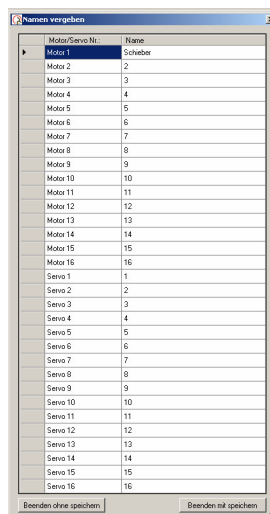


Abbildung 4-9: Unterfenster zur Vergabe von Namen für Motoren und Servos

Die Liste der Motoren und Servos wird in einem DataGridView angezeigt da sich dieses Element sehr gut für die Anzeige von Tabellen eignet.

Da die Daten auch nach Beenden des Programms nicht verloren gehen dürfen, müssen die Einstellungen in irgendeiner Form abgespeichert werden. Um das Problem der Serialisierung¹⁴ zu lösen wurde wie bei den Einstellungen für die serielle Schnittstelle mit Einstellungsdateien (UserSettings¹⁵), gearbeitet. Dadurch ist das Abspeichern und Laden der Daten einfach zu realisieren. Auch der Datenaustausch zwischen den Fenstern erfolgt über diese UserSettings. Beim Schließen des Unterfensters werden die Daten, falls dies vom Benutzer gewünscht, im UserSettings abgespeichert und nach dem Schließen des Unterfensters werden die Daten aus dem UserSettings im Hauptfenster neu ausgelesen und dadurch eventuelle Änderungen übernommen.

¹⁴ [Kühnel, 2009]

¹⁵ [codeproject, 2010]

5 Kamera

In Zusammenarbeit mit einem anderen Projekt¹⁶ wurde ein einfaches System zur Auswertung und Parametrierung der Kamera entwickelt. Einerseits ist es wichtig, gewisse Parameter der Kamera, wie z.B.: Threshold, schnell umzustellen, andererseits sollte die Funktion der Kamera schnellstmöglich getestet werden können.

5.1 Grundkonzept

Die Kameras werden eingesetzt um die Zylinder auf dem Spielfeld zu detektieren. Dabei sind weiße Zylinder von schwarzen zu unterscheiden. Während weiße Zylinder Punkte bringen sind schwarze Zylinder am Tisch angeschraubt und bringen keine Punkte. Viel mehr besteht die Gefahr dass der Roboter an einen schwarzen Zylinder hängen bleibt. Daher ist es essenziell, schwarze von weißen Zylindern zu unterscheiden. Die Kameras schneiden bei jedem Bild die Position der Zylinder aus und bestimmen die Farbe. Die Information über die Farbe wird dann als Binärwert (1 für Schwarz, 0 für Weiß) übergeben.

Um das gesamte Spielfeld zu erkennen sind zwei Kameras nötig, wobei jede Kamera einen Teil des Tisches analysiert. Die dritte Kamera ist auf die Orangen des Gegners gerichtet um zu erkennen ob der Gegner diese aufgesammelt hat. Welche Kamera welche Zylinder erkennt, wird in der anderen Projektarbeit ausführlich erläutert.

5.2 Kamera parametrieren

Zum Abgleich der Kamera auf verschiedene Lichtverhältnisse und verschiedene Definitionen von Weiß bzw. Schwarz ist es nötig den Threshold umzustellen. Der Threshold ist ein Wert der angibt ab welchen Wert die Kamera den Zylinder als Weiß bzw. Schwarz erkennt. Dazu wird der blaue Farbwert des Bildes, welcher zwischen 0 und 255 liegen kann, mit dem Thresholdwert verglichen. Ein kleiner Wert (z.B.: < 40) bedeutet, dass es sich um einen schwarzen Zylinder handelt, ein hoher Wert, dass es sich um einen weißen Zylinder handelt. Dieser Wert, in diesem Beispiel 40, muss leicht umzustellen sein.

Um den gemessenen Wert auslesen zu können ist es notwendig den Rückgabeparameter umzustellen. Im Normalbetrieb wird 0 für Weiß, 1 für Schwarz zurückgegeben. Um den

¹⁶ [Muckenhumer, 2010]

Threshold abgleichen zu können ist es notwendig den Rückgabewert umzustellen um die gemessenen Werte zu bekommen. Die Rückgabewerte sind dann Werte zwischen 0 und 255.

Folgende Befehle sind für die Parametrierung der Kamera vereinbart worden:

- ✓ a, b, c: Anhalten der Messung von Kamera 1, 2 oder 3
- ✓ rs: Reset aller Kameras
- ✓ ca 1, cb 1, cc 1: Kamera 1, 2 oder 3 auf binäre Datenübertragung umstellen
- ✓ ca 0, cb 0, cc 0: Kamera 1, 2 oder 3 soll Werte übertragen
- ✓ sc: Messung der Kameras starten
- ✓ ga, gb, gc: Daten von Kamera 1, 2 oder 3 anfordern
- ✓ sa xxx, sb xxx, sc xxx: Threshold von Kamera 1, 2 oder 3 auf den Wert xxx umstellen

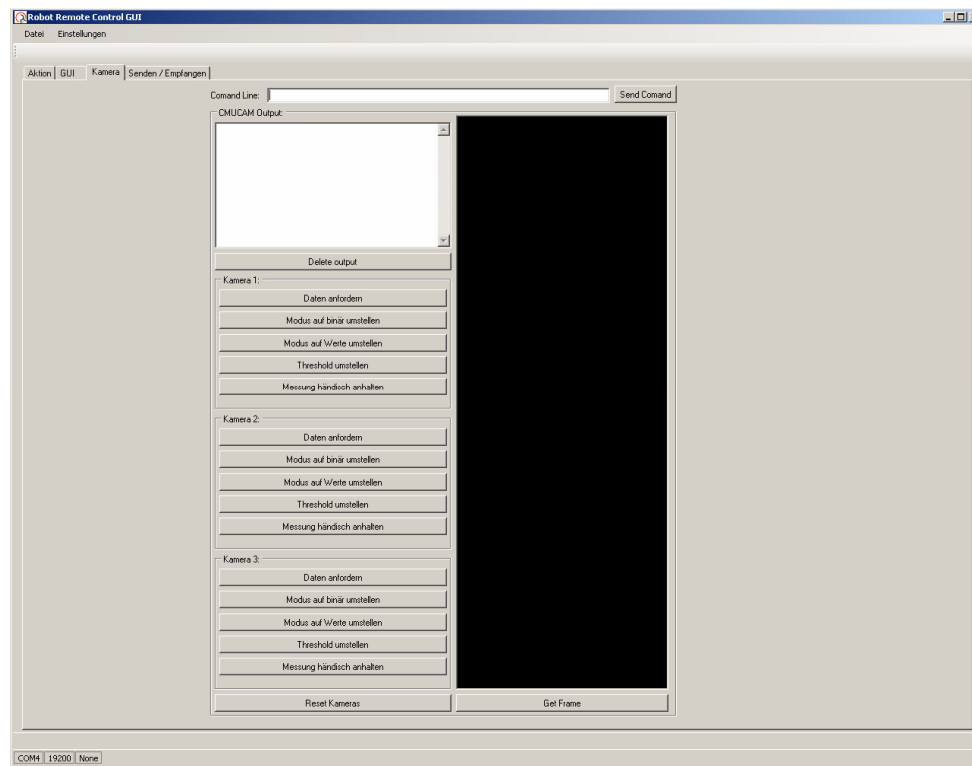


Abbildung 5-1: Kamera – Parametrierungsfenster

Wie aus Abbildung 5-2 ersichtlich, ist es möglich alle drei Kameras einzeln anzusprechen. Dadurch sind Einstellungen auf einer Kamera schnell durchzuführen. Weiters ist es möglich, durch eine Command Line jeden beliebigen Befehl zu senden. Daten, welche die Kamera sendet werden im Output – Textfeld angezeigt. Um die Kamera richtig einrichten zu können ist es möglich den aktuellen Bildbereich mit Get Frame zu laden.

5.2.1 Daten anfordern

Mit diesem Befehl werden die aktuellen Messdaten der Kamera angefordert. Die Kamera sendet dann die Messdaten, je nach Einstellung, entweder binär oder in Werten von 0-255 (siehe 5.2.2 und 5.2.3). Um die Daten anzufordern wird folgende Befehlskette gesendet.

- ✓ Anhalten der Messung a, b oder c
- ✓ Daten von Kamera anfordern ga, gb oder gc

```

/// <summary>
/// Corn-Config von Kamera 1 anfordern
/// 22.01.2010 MC
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_Cam1_Click(object sender, EventArgs e)
{
    this.WriteCOM("a\r");
    this.WriteCOM("ga\r");
    this.spSender = 1;
}

```

Damit die Kamera Daten sendet, muss die aktuelle Messung gestoppt werden. Diese speichert dann die letzte Messung. Wenn dann die Daten angefordert werden schickt die Kamera die Daten der letzten Messung und beginnt wieder automatisch mit der Messung.

5.2.2 Modus auf binär umstellen

Mit diesem Befehl wird die Kamera angewiesen, die gemessenen Daten binär zu übertragen wenn diese angefordert werden. Wie bereits erläutert werden im Normalbetrieb die Daten immer binär übertragen, jedoch ist es zum Einstellen der Threshold notwendig die gemessenen Werte einzusehen. Um den Modus nach dem Einstellen wieder in den binären Modus zu stellen wird folgende Befehlskette gesendet.

- ✓ Anhalten der Messung a, b, oder c
- ✓ Modus auf binär umstellen ca 1, cb 1 oder cc 1
- ✓ Messung wieder starten sc

```

/// <summary>
/// Der Kamera anweisen die Daten ab jetzt binär zu schicken
/// 1 --> schwarz , 0 --> weiß
/// MC 29.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_Cam1Bin_Click(object sender, EventArgs e)
{
    this.WriteCOM("a\r");
    this.WriteCOM("ca 1\r");
    this.WriteCOM("sc\r");
}

```

Um den Modus umzustellen, muss die Messung vorher angehalten werden. Danach wird der Modus umgestellt und zum Schluss wird die Messung wieder gestartet. Werden danach Daten von der Kamera angefordert, werden diese Daten binär übertragen.

5.2.3 Modus auf Werte umstellen

Wie bereits erläutert ist es vor allem zum Einstellen des Threshold nötig, die gemessene Werte zu analysieren. Diese können zwischen 0 und 255 liegen. Um die tatsächlichen Farbintensitätswerte auszulesen muss folgende Befehlskette gesendet werden.

- ✓ Anhalten der Messung a, b oder c
- ✓ Modus auf Werte umstellen ca 0, cb 0 oder cc 0
- ✓ Messung wieder starten sc

```
/// <summary>
/// Die Kamera anweisen die Datenwerte zu schicken
/// 0 - 255
/// MC 29.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_Cam1Data_Click(object sender, EventArgs e)
{
    this.WriteCOM("a\r");
    this.WriteCOM("ca 0\r");
    this.WriteCOM("sc\r");
}
```

Um den Modus umzustellen, muss die Messung vorher angehalten werden. Danach wird der Modus umgestellt und zum Schluss wird die Messung wieder gestartet. Werden danach Daten von der Kamera angefordert, werden die Intensitätswerte übertragen.

5.2.4 Threshold umstellen

Bei verschiedenen Lichtverhältnissen kann sich die Grenze zwischen Schwarz und Weiß leicht verschieben. Standardmäßig ist ein Wert von 40 eingestellt, jedoch muss ein schnelles Umstellen dieses Wertes möglich sein. Um dies zu bewerkstelligen muss folgende Befehlskette gesendet werden.

- ✓ Anhalten der Messung a, b, oder c
- ✓ Neuen Threshold senden sa xxx, sb xxx oder sc xxx
- ✓ Messung wieder starten sc

```
/// <summary>
/// Den Threshold für Schwarz - Weiß unterscheidung umstellen
/// MC 29.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_Cam1Threshold_Click(object sender, EventArgs e)
{
```

```

frm_SetThreshold diag = new frm_SetThreshold();
if (diag.ShowDialog() == DialogResult.OK)
{
    decimal threshold = diag.nud_Threshold.Value;
    this.WriteCOM("a\r");
    this.WriteCOM("sa " + threshold.ToString() + "\r");
    this.WriteCOM("sc\r");
}
}

```

Bevor diese Befehlskette gesendet wird öffnet sich noch ein Unterfenster für die Eingabe des neuen Threshold – Wertes.



Abbildung 5-2: Unterfenster für die Eingabe des neuen Threshold

Wenn dieses Unterfenster mit OK geschlossen wird, wird die Messung der Kamera gestoppt, der neue Thresholdwert gesendet und danach die Messung wieder gestartet.

5.3 Auswertung

Um die aktuelle Spieltischkonfiguration graphisch auszuwerten wurde noch ein weiteres Fenster für die Kameraauswertung programmiert. In diesem können die Daten der Kamera abgerufen werden und je nach Spieltischkonfiguration wird das richtige Bild geladen. Um auch ohne Kameras ein Bild laden zu können wird vor dem Abfragen der Kameradaten der Benutzer gefragt ob die Kameras verfügbar sind.

```

/// <summary>
/// Hintergrundbild des Spielfeldes laden
/// 28.12.2009 MC
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void tsb_LoadCornConfig_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Bild automatisch laden? (Kameras müssen
    verfügbar sein)", "Automatisch laden?", MessageBoxButtons.YesNo) ==
    DialogResult.Yes)
    {
        // Daten von Kamera laden
        this.sp_Connection.DiscardInBuffer();
        this.WriteCOM("a\r");
        this.WriteCOM("ga\r");
        this.spSender = 2;
    }
    else

```

```
{
    OpenFileDialog diag = new OpenFileDialog();
    diag.InitialDirectory = "CornConfig";
    diag.Filter = ("CornConfigFiles (*.emf)|*.emf|All files (*.*)|*.*");
    if (diag.ShowDialog() == DialogResult.OK)
    {
        this.pbx_Spielfeld.ImageLocation = diag.FileName;
    }
}
}
```

Als diese Funktion geschrieben wurde, war nur eine Kamera verfügbar, weshalb auch nur eine Kamera abgefragt wird. In Zukunft müssen aber zwei Kameras abgefragt werden, das heißt diese Methode muss noch etwas abgeändert werden.

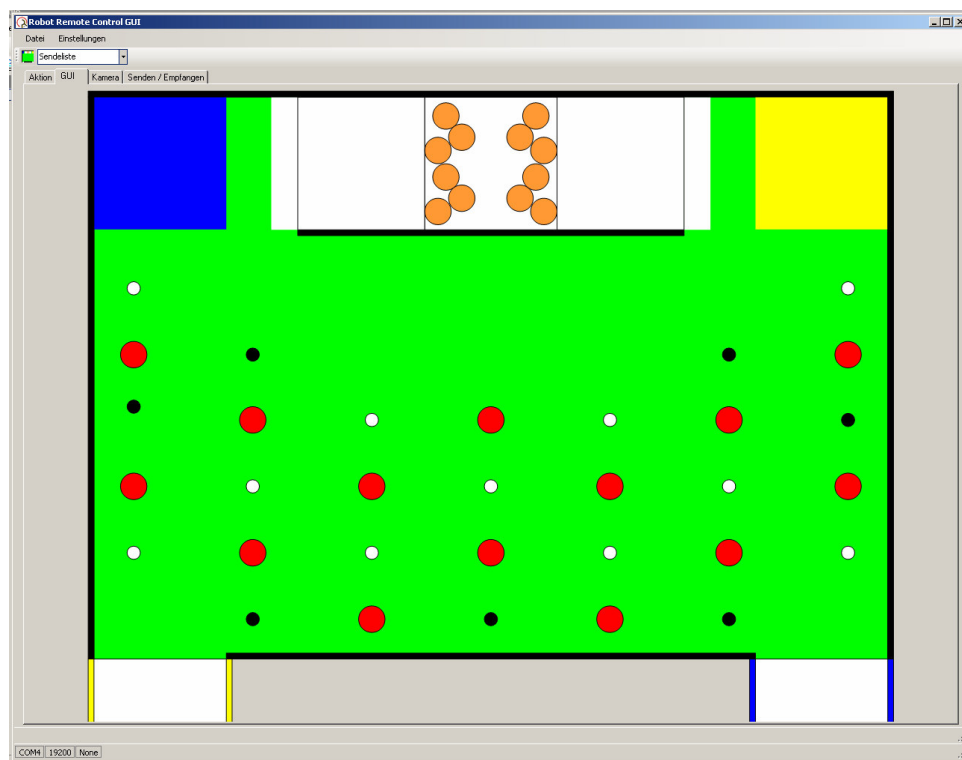


Abbildung 5-3: Auswertung der Kameradaten

6 Sendeliste senden / Daten empfangen

Wie in den Anforderungen beschrieben ist es notwendig mehrere Befehle zu einer Sendeliste zusammenzuführen und diese dann zum Roboter zu senden. Weiters müssen Daten, welche der Roboter sendet, angezeigt werden. Diese beiden Anforderungen werden in dem Fenster, welches Abbildung 6-1 zeigt, erfüllt.

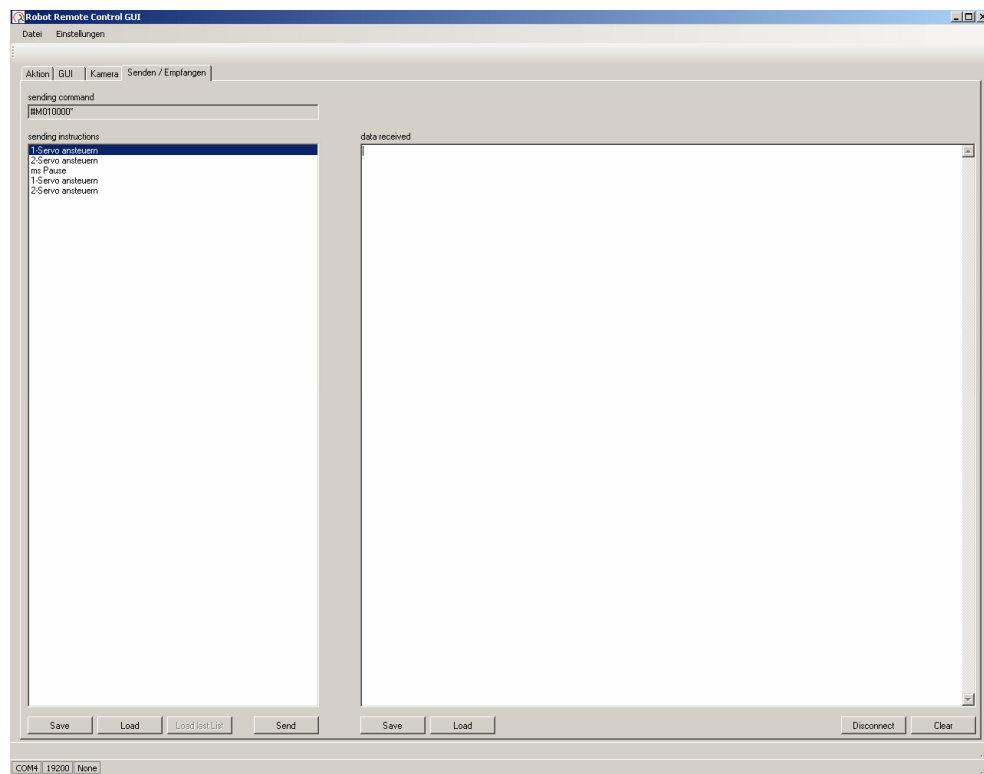


Abbildung 6-1: Fenster für Sendelisten senden und Daten empfangen

6.1 Sendeliste senden

6.1.1 Senden

Um eine Sendeliste zu senden wurde ein Timer eingeführt der in einem gewissen Zeitintervall die einzelnen Befehle an die serielle Schnittstelle sendet. Um die als letztes gesendete Liste wieder zu laden, um sie gegebenenfalls abzuspeichern oder nochmals zu senden, wird vor dem Senden die gesamte Liste in einem temporären Verzeichnis gespeichert. Diese kann dann nach dem Senden mit dem Button „Load Last List“ wieder geladen werden.

Da in der Sendeliste auch Pausen vorkommen können wurde ein zweiter Timer programmiert, der die Pausenfunktion erfüllt.

```

/// <summary>
/// Die Daten der Befehlliste werden an die serielle Schnittstelle
/// geschickt
/// MC 21.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_send_Click(object sender, EventArgs e)
{
    // Aktuelle Befehlsliste speichern
    StreamWriter sw = new
    StreamWriter(Path.Combine(Application.StartupPath, "temp.csv"));
    foreach (Befehle b in this.Commands)
    {
        sw.Write(b.Befehl + ";" + b.Bezeichnung + "\n");
    }
    sw.Close();
    btn_rueck.Enabled = true;
    // Daten Schreiben
    this.timer_send.Interval = COM_Setting.Default.DataRate;
    this.timer_send.Start();
}

```

Wie bereits beschrieben wird die Sendeliste zuerst temporär abgespeichert und danach der
Sende – Timer gestartet. Die Befehle werden dann nacheinander in einen gewissen
Zeitintervall gesendet.

```

/// <summary>
/// Wenn der Timer gestartet wurde(wird gemacht wenn auf Senden geklickt
/// wird), wird bei jedem Tick ein Befehl gesendet
/// MC 21.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void timer_send_Tick(object sender, EventArgs e)
{
    if (this.Commands.Count > 0)
    {
        if (this.Commands[0].Bezeichnung.Contains("Pause"))
        {
            this.timer_send.Stop();
            this.timer_pause.Interval =
            Convert.ToInt32(this.Commands[0].Befehl);
            this.timer_pause.Start();
        }
        else
        {
            this.WriteCOM(Commands[0].Befehl);
            this.Commands.RemoveAt(0);
            this.SetSendeliste();
        }
    }
    else
    {
        this.timer_send.Stop();
    }
}

```

Solange die Anzahl der Befehle in der Sendeliste größer 0 ist werden die Befehle an die
serielle Schnittstelle gesendet. Außer es handelt sich bei dem Befehl um eine Pause. In diesem

Fall wird der Sende – Timer angehalten und der Pause – Timer mit der Pausezeit als

Intervallzeit gestartet.

```

/// <summary>
/// Wenn der aktuelle Befehl eine Pause ist wird die Pause angewartet und
/// dann wieder weitergesendet
/// MC 26.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void timer_pause_Tick(object sender, EventArgs e)
{
    timer_pause.Stop();
    this.Commands.RemoveAt(0);
    this.SetSendeliste();
    timer_send.Start();
}

```

Da die Intervallzeit des Pausetimers genau der gewünschten Pause entspricht wird das Tick – Ereignis genau nach der Pausezeit zum ersten Mal ausgelöst. Der Pause – Timer wird wieder angehalten und der Sende – Timer wieder gestartet.

6.1.2 Speichern / Laden einer Sendeliste

Um ein und die selbe Aktion öfters an den Roboter senden zu können ist es von Nöten eine Sendeliste abzuspeichern um sie später wieder laden zu können.

```

/// <summary>
/// Befehlsliste speichern
/// MC 22.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_save_Click(object sender, EventArgs e)
{
    SaveFileDialog save = new SaveFileDialog();
    save.Filter = "CSV-Dateien (*.csv) |*.csv";
    if (save.ShowDialog() == DialogResult.OK)
    {
        StreamWriter sw = new StreamWriter(save.FileName);
        foreach (Befehle b in this.Commands)
        {
            sw.Write(b.Befehl + ";" + b.Bezeichnung + "\n");
        }
        sw.Close();
    }
}

```

Die Sendeliste kann als csv – Datei auf dem Rechner abgespeichert werden. Der Speicherpfad wird vor dem Speichern abgefragt. Danach werden die Befehle selbst und ihre Bezeichnungen jeweils in einer Zeile geschrieben.

Zum Laden einer Sendeliste muss der Benutzer den Pfad der zu ladenden Sendeliste angeben. Falls sich im Programmspeicher noch eine andere Sendeliste befindet wird abgefragt ob diese vor dem Laden der neuen Sendeliste abgespeichert werden soll. Nach Auswahl einer gewünschten Aktion wird die neue Sendeliste in den Programmspeicher geladen.

```

/// <summary>
/// Lest eine vorhandene Befehlliste
/// Wenn sich noch eine Befehlliste im Speicher befindet wird abgefragt ob
/// diese gespeichert werden soll
/// MC 22.01.2010
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btn_read_Click(object sender, EventArgs e)
{
    OpenFileDialog load = new OpenFileDialog();
    load.Filter = "CSV-Dateien (*.csv) |*.csv";
    if (load.ShowDialog() == DialogResult.OK)
    {
        // Wenn sich noch Befehle im Speicher befinden abfragen ob
        // diese gespeichert werden sollen
        if (Commands.Count != 0)
        {
            DialogResult result = MessageBox.Show("Soll die aktuelle
            Befehlsliste gespeichert werden?", "Speichern?",
            MessageBoxButtons.YesNoCancel);
            if (result == DialogResult.Yes)
            {
                btn_save_Click(sender, e);
                Commands.Clear();
                this.ReadFile(load.FileName);
            }
            else if (result == DialogResult.No)
            {
                Commands.Clear();
                this.ReadFile(load.FileName);
            }
        }
        else
        {
            this.ReadFile(load.FileName);
        }
    }
}

```

Da, abhängig von der Benutzereingabe und Programmspeicherstatus, das Laden der Liste an verschiedenen Programmstellen stattfinden kann, wurde das Laden der Liste in eine eigene Methode ausgelagert.

```

/// <summary>
/// Methode zum lesen einer Befehlsliste
/// MC 22.01.2010
/// </summary>
/// <param name="file">Speicherort der Befehlsliste</param>
private void ReadFile(string file)
{
    try
    {
        StreamReader sr = new StreamReader(file);
        string zeile = string.Empty;
        string[] split = new string[2];
        while (sr.Peek() != -1)
        {
            zeile = sr.ReadLine();
            split = zeile.Split(new char[] { ';' });
            Befehle b = new Befehle(split[0], split[1]);
            Commands.Add(b);
        }
        sr.Close();
    }
}

```

```
        this.SetSendeliste();  
    }  
    catch (Exception ex)  
    {  
        MessageBox.Show("Fehler beim lesen! Fehlertext: " +  
            ex.Message);  
    }  
}
```

Voraussetzung für das Lesen einer Sendeliste ist, dass es sich um ein File handelt das die richtige Struktur hat. Ansonsten kann es beim Lesen zu einem Fehler kommen, welcher im catch – Block abgefangen wird. Um den Fehler zu lokalisieren wird in diesem Fall der Fehlertext ausgegeben. Im Normalfall liest diese Methode Zeile für Zeile des Files ein, trennt Befehl von Bezeichnung und erstellt mit den geladenen Befehlen eine neue Sendeliste.

6.1.3 Einzelne Befehle aus der Sendeliste löschen

Nachdem eine Sendeliste erstellt worden ist kann es nötig sein einzelne Befehle aus dieser Sendeliste wieder zu löschen. Für diesen Zweck wurde die Funktion der Entfernen – Taste programmiert. Ist ein Befehl markiert und wird dann die Entfernen – Taste gedrückt, wird der markierte Befehl gelöscht.

```
/// <summary>  
/// In der Listbox der Befehle auf die Entfernentaste reagieren  
/// MC 19.01.2010  
/// </summary>  
/// <param name="sender"></param>  
/// <param name="e"></param>  
private void lbx_Befehle_KeyDown(object sender, KeyEventArgs e)  
{  
    if (e.KeyCode == Keys.Delete && this.Commands.Count > 0)  
    {  
        int index = this.lbx_Befehle.SelectedIndex;  
        this.Commands.RemoveAt(index);  
        this.SetSendeliste();  
    }  
}
```

6.2 Daten empfangen

Wie in den Anforderungen beschrieben ist es notwendig, Daten, welche vom Roboter gesendet werden, auszugeben. Dies soll vor allem die Inbetriebnahme und Testphase erleichtern, weil es dadurch möglich ist, Statusmeldungen abzurufen. Für diesen Zweck wurde eine Textbox programmiert in welcher alle empfangenen Daten angezeigt werden. Weiters ist es möglich die empfangenen Daten abzuspeichern oder eine abgespeicherte Liste zu laden um die Daten wiederholt einzusehen.

7 ZUSAMMENFASSUNG UND AUSBLICK

Durch die neu erstellte Benutzeroberfläche ist es möglich, den Roboter der Fachhochschule Wels für die EUROBOT^{open} in Betrieb zu nehmen und zu Testen. Wichtige Faktoren, wie die einfache Bedienung wurden in der Entwicklung berücksichtigt.

Dadurch, dass die Befehle in eine eigene Klasse ausgelagert wurden ist es möglich, weitere Befehle einfach zu implementieren. Auch die Möglichkeit, für die Motoren und Servos Namen zu vergeben, erleichtert die Bedienung des Programms. Ein weiterer wichtiger Faktor, verschiedene Befehlslisten abzuspeichern und wieder zu lesen, wurde realisiert.

Um ein erfolgreiches Antreten beim Bewerb sicherzustellen wurde eine Möglichkeit geschaffen, die Kameras rund um den Spieltisch einfach und schnell zu konfigurieren. Auch das war ein Schlüsselpunkt dieser Projektarbeit.

Obwohl die Benutzeroberfläche stark für die EUROBOT^{open} 2010 entworfen wurde ist es durch nicht allzu großen Aufwand möglich, dieses Programm für die nachfolgenden Bewerbe zu adaptieren. Weiters bestehen Möglichkeiten, das Programm weiters zu verbessern. So ist es denkbar, die Befehle für das Fahren des Roboters graphisch, auf der geladenen Spielfeld – Graphik, mit der Maus einzugeben. Auch das Einlesen eines Joysticks oder einer Game – Controllers ist denkbar. Die Erweiterung um einen Befehl zur Kurvenfahrt ist eine weitere denkbare Erweiterung.

8 LITERATUR

8.1 Bücherliste

- [Rules E2010, 2009]: E2010_rules_and_drawing_EN.pdf
Stand 24.09.2009
- [Tietze - Schenk, 1999] Ulrich, Tietze - Christoph, Schenk:
Halbleiter – Schaltungstechnik
11. Auflage
- [Zauner, 2009] Zauner, Michael:
Bachelorarbeit 1 - Modulares und skalierbares elektronisches System für autonome Roboter
- [Kühnel, 2009] Kühnel, Andreas:
Visual C# 2008. Das umfassende Handbuch.
- [Muckenhumer, 2010] Muckenhumer, Bernhard:
Bachelorarbeit 1 – Anbindung und Adaptierung einer mikrokontroller gesteuerten Kamera an einen autonomen Roboter

8.2 Internet

- [eurobot, 2010] <http://www.eurobot.org/> Stand 14.03.2010
- [Microsoft, 2010] <http://msdn.microsoft.com/> Stand 12.02.2010
- [codeproject, 2010] <http://www.codeproject.com/> Stand 16.02.2010
- [rn-wissen, 2010] <http://www.rn-wissen.de/> Stand 16.02.2010
- [sprut, 2010] <http://www.sprut.de/> Stand 04.03.2010