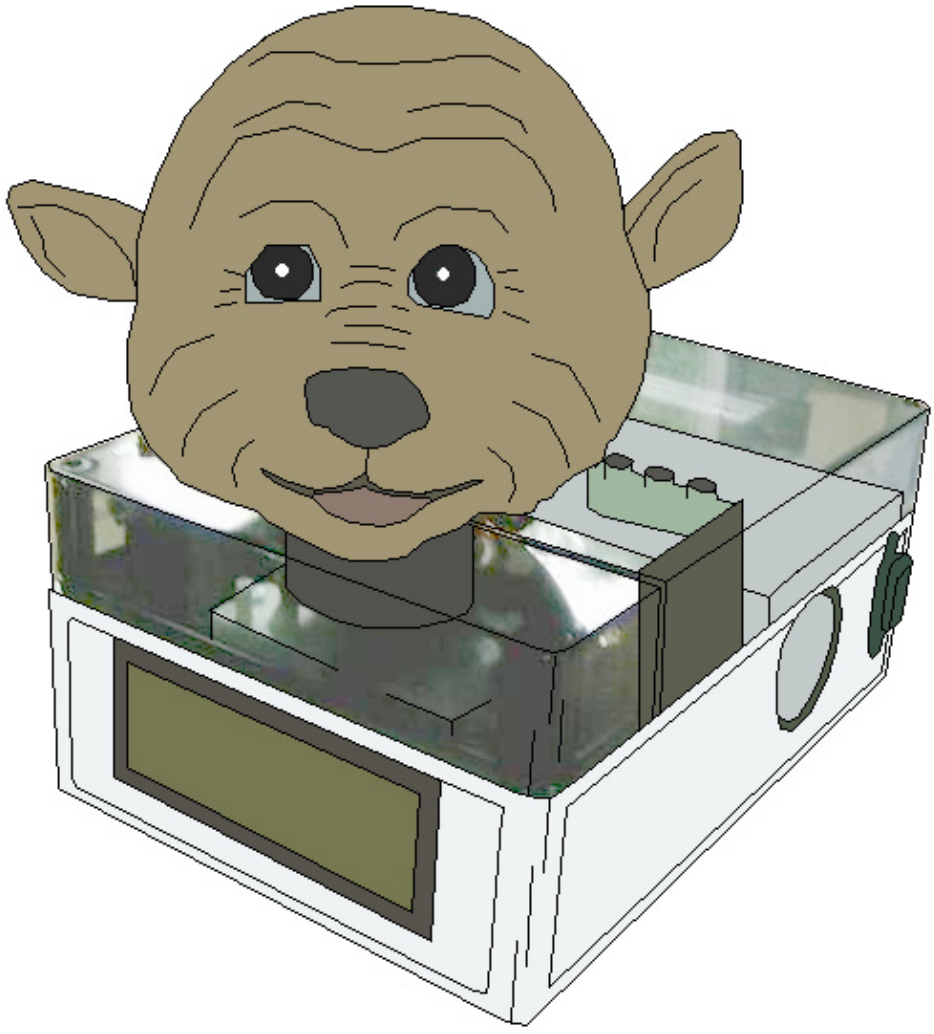
Technikerarbeit 2006/07

Interaktiver Roboter
„SUPER-YANO”



... eine Abschlussarbeit mit Köpfchen ...

von Philipp Bank

Technikerarbeit 2006/07

Informationstechnik

Interaktiver Roboter

Verfasser: Philipp Bank

Betreuer: Jürgen Schnaiter

Gewerblich Technische Schule Offenburg
Moltkestraße 23 • 77654 Offenburg
www.gs-offenburg.de

Erklärung

Ich versichere, dass die Technikerarbeit von mir selbstständig angefertigt und nur die angegebenen Hilfsmittel benutzt wurden.
Alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, habe ich durch Angabe der Quellen kenntlich gemacht.

11.06.2007	Philipp Bank
Datum	Unterschrift

Wenn man Roboter baut, fällt einem auf, wie komplex die Dinge sind.
Prof. Cynthia Breazeal, MIT Media Lab

Prolog

Bei dieser Technikerarbeit, dem Abschlussprojekt der Ausbildung, geht es um einen Roboter, dessen Zweck es ist, Kindern von 3 bis 8 Jahren Märchen zu erzählen.
Auf den Verlauf der Geschichten können die Kinder dabei interaktiv einwirken.
Der Roboter kommuniziert mit Sprachausgabe und Spracheingabe, hält mit Kameras Blickkontakt zu seinen Zuhörern und simuliert menschliche, bzw. tierische Verhaltensweisen.

Als mechanische Basis wurde der Kopf einer elektronischen Spielzeugpuppe „Yano“ verwendet, welcher massiv umgebaut und erweitert wurde.
Da es sich in der Internetszene der Bastler und Tüftler mittlerweile fast zum Standard etabliert hat, modifizierte, gehackte und verbesserte Geräten mit einem vorangestellten „Super-“ zu titulieren, wurde bei der Namensgebung dieses Projekts ebenso verfahren, was den Namen *Super-Yano* erklärt.

Bis Anfang Februar wurde von mir jedoch noch ein anderes Roboterprojekt mit einem Industrieroboter der Gewerbeschule Offenburg als Technikerarbeit verfolgt. Hier ging es darum eine veraltete DOS-Software zur Ansteuerung des Roboters für Windows nachzubilden. Leider waren aber keine Informationen mehr zum Protokoll der Ansteuerung aufzutreiben und auch mit Methoden des reverse-engineering konnten die Geheimnisse des Kommunikationsformats nicht entschlüsselt werden. Darum musste das Projekt aufgegeben werden. An dessen Stelle wurde ab diesem Zeitpunkt Super-Yano als Abschlussarbeit entwickelt.

Somit stand zu Beginn dieses Projekts nur noch ein knapp begrenzter Zeitraum von drei Monaten zu Verfügung.
Dennoch ist es ein sehr umfangreiches und komplexes Projekt mit einer Vielzahl von Features geworden.
Alle Funktionen, Eigenschaften, theoretischen Hintergründe und Schritte der Umsetzung zu beschreiben, würde ein ganzes Buch füllen.
Um den Rahmen dieser Dokumentation nicht zu sprengen kann hier daher nicht auf sämtliche Einzelheiten detailliert eingegangen werden.
So habe ich vielmehr versucht die Herangehensweisen und konkreten Schritte der Realisierung beim Bau des Roboters zu erklären, sowie die Abläufe und Wirkungsweise der Software anhand der daran wesentlich beteiligten Funktionen.

Eine vollständige Softwaredokumentation mit Illustrationen der Aufrufabhängigkeiten zwischen allen Funktionen befindet sich zudem auf der beiliegenden CD im HTML-Format.

Inhalt

Was sind interaktive Roboter?

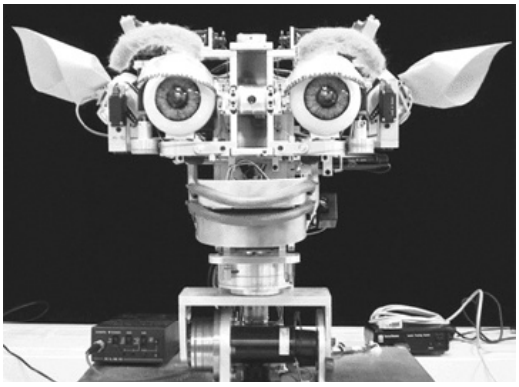


Abb. 1: „Kismet“

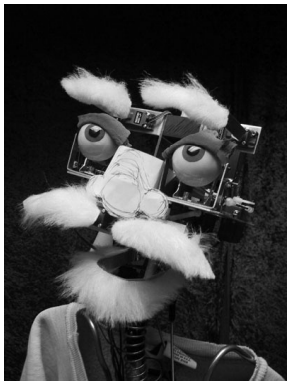


Abb. 2: „Doc Beardsley”

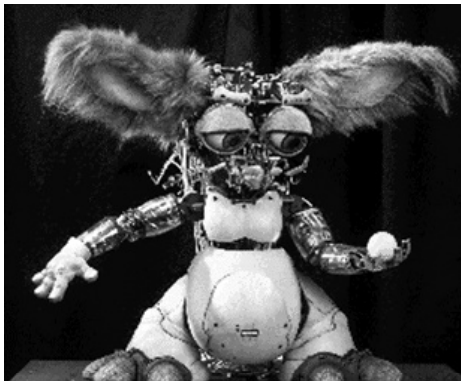


Abb. 3: „Leonardo“

Interaktive Roboter, auch „Social Machines“ genannt, sind Roboter zur „Mensch-Technik-Interaktion“.
Die grundlegende These besagt, dass immer komplexer werdende Maschinen einfache und an den Menschen angepasste Schnittstellen erfordern.
Herausragende Eigenschaft solcher Schnittstellen ist die Kommunikation mit Menschen über Sprache, Mimik und Gestik.
Um dies zu ermöglichen sind sie menschenähnlich konzipiert und simulieren menschliche Verhaltensweisen.
Ein bekannter Vertreter dieser Art von Robotern ist beispielsweise „Kismet“ (Abb. 1) vom M.I.T. (Massachusetts Institute of Technology, USA).

Kismet [1] ist ein K.I.-Projekt, zur Erforschung der Mensch-Maschine-Kommunikation.
Der Roboter besteht aus einem beweglichen Kopf und kann ein breites Spektrum verschiedener Gesichtsausdrücke darstellen. Zur räumlichen Erfassung von Gesichtern und Bewegungen dienen vier Kameras, davon zwei als „Augen“.
Über Mikrofon und Sprachausgabe ist er in der Lage mit Personen zu sprechen.

Soziale Maschinen sind noch ein vergleichsweise junges Forschungsgebiet und wenig verbreitet.
Die steigende Nachfrage nach intuitiv bedienbaren Servicerobotern und deren zunehmendes Vordringen in den Alltag wird in den nächsten Jahren jedoch sicher für eine rasante Weiterentwicklung sorgen.

Exemplarisch sei an dieser Stelle der Roboter „Paro“ (Abb. 4) erwähnt [2].
Dieser Therapieroboter in Gestalt eines Seehundes wurde für 10 Millionen Euro entwickelt und wird in japanischen Altenheimen zur Unterhaltung und als Ersatz für Haustiere eingesetzt.
In Japan, dem Land der stärksten Bevölkerungsvergreisung, hofft man eines Tages mit Pflege- und Servicerobotern den Mangel an Betreuungspersonal kompensieren zu können.


Abb. 4: Therapieroboter „Paro“

Wie es zu dieser Technikerarbeit kam

Das Vorbild

Anfang August 2006 bin ich zufällig auf das „Robotic F.A.C.E.“ [3], ein Projekt des MIT Media Lab, gestoßen. Hier wurde der Kopf einer Spielzeugpuppe „Yano“ modifiziert um die Gesichtsmimik von einem PC aus zu steuern.

Abb. 5: Robotic F.A.C.E. der Speech Interface Group des MIT Media Lab mit modifizierter Yano-Puppe

Bei Yano handelt es sich um eine ca. 30cm hohe elektronische Puppe, die Kindern Geschichten erzählt und dabei den Kopf bewegt, mit den Ohren wackelt und beim Sprechen Mundbewegungen ausführt (näheres dazu im Kapitel „Mechanik“).

Ursprünglich sollte der Roboter ähnlich wie Kismet agieren. Allerdings stellte sich heraus, das aufgrund unzureichender Bewegungsfreiheitsgrade der Mechanik (beispielsweise sind die Bewegungen der Ohren, Augenlider und Augenbrauen gekoppelt), die Emotionsalgorithmen von Kismet nicht verwendet werden konnten.

Daher wird der Roboter heute zusammen mit einem anderen Projekt zur Überwachung entfernter Orte eingesetzt und reagiert dabei mit unterschiedlichen Mimiken auf verschiedene Ereignisse am Überwachungsort.

Die Idee

Angeregt durch das Robotik F.A.C.E. war meine Idee nun, ebenfalls auf Basis der Yano-Spielzeugpuppe, einen interaktiven Roboter zu bauen und zu programmieren. Dank des MIT-Projekts waren die mechanischen Beschränkungen bereits bekannt, so dass auch gleich geplant war, nicht nur die elektronische Ansteuerung zu ersetzen, sondern auch die komplette Mechanik und somit von der Yano-Puppe nur die „Hülle“ zu verwenden.

Es war bei diesem Projekt jedoch nicht geplant ein intelligentes System nach dem Vorbild Kismets zu schaffen, sondern vielmehr zunächst die Funktionalität der Yano-Puppe als Geschichtenerzähler beizuhalten und zu erweitern, sowie eine Plattform für mögliche weiterführende Projekte in diesem Bereich der Robotik aufzubauen.

Das Projekt begann somit mit dem Erwerb einer gebrauchten Yano-Spielzeugpuppe und dem „Ausschlachten“ selbiger.

Zielsetzungen

Konzeption

Ziel dieses Projekts war, aufbauend auf der bereits mechanisch modifizierten Yano-Puppe, die Konstruktion eines computergesteuerten Roboterkopfes, welcher zur Kommunikation über Mimik und Sprache fähig ist und als interaktiver Geschichtenerzähler für Kinder agieren soll.

Da jedoch zu Beginn dieses Projektes, aufgrund der eingangs geschilderten Umstände, nur noch ein sehr begrenzter Zeitrahmen zur Verfügung stand, musste unter hohem Druck gearbeitet werden und auf die Realisierung mancher gewünschten Features, wie das Wiedererkennen von Personen über Gesichtserkennung und individuelle Gesichtsmerkmale zur namentlichen Anrede, leider verzichtet werden.

Im der Konzeption als interaktiver Geschichtenerzähler wurde der Roboterkopf dabei als physikalische Schnittstelle zur Außenwelt entworfen, mit der Möglichkeit zur Aufnahme der Kommunikation mit Benutzern über ein gesichtsähnliches Medium. Im wahrsten Sinn des Wortes: ein „Inter-Face“.

Die Interaktivität besteht darin, dass dem Benutzer – Zielgruppe sind Kinder von 3 bis 8 Jahren – ein Teil einer Geschichte erzählt wird, deren weiterer Verlauf dann beeinflusst werden kann.
Dies entspricht der Funktion des originalen Yano.
Da, wie bereits erwähnt, eine Verbesserung dessen angestrebt war, sollte dieses Projekt den Geschichtenerzähler um einige Features erweitern, die nachfolgend genannt werden.

Anforderungen und gewünschte Features

Um als Geschichtenerzähler, insbesondere von jüngeren Kindern, akzeptiert zu werden muss der Roboter meiner Ansicht nach über ein Grundmaß sozialer Fähigkeiten verfügen. Dazu bedarf er zunächst verschiedener Wahrnehmungs- und Ausdrucksmöglichkeiten.
Diese sollten über entsprechende Sensoren und Aktoren realisiert werden.
Verbale Kommunikation sollte über Standardkomponenten wie Soundkarte, Mikrofon und Lautsprecher umgesetzt werden.

Zur nonverbalen Kommunikation ist eine ausgeprägte Gesichtsmimik wünschenswert. Von der Mechanik erfordert dies möglichst viele Bewegungen unabhängig voneinander ausführen zu können.

Zugleich ist die Fähigkeit Blickkontakt aufzunehmen und zu halten sicherlich der bedeutenste Faktor bezüglich des sozialen Verhaltens und sollte ebenso implementiert werden.
Hierzu benötigt der Roboter Kameras, die ihm einen visuellen Eindruck seiner Umgebung verschaffen und es ihm ermöglichen Personen wahrzunehmen.

Die Elektronik sollte so entwickelt werden, dass ein einfacher Anschluss an den PC möglich ist.
Sie muss in der Lage sein die erforderlichen Aktoren präzise, kraftvoll und mit unterschiedlichen Geschwindigkeiten anzusteuern.
Ferner ist ein Display zur Textanzeige wünschenswert, welches beispielsweise als Statusanzeige dienen kann oder zur Entwicklungszeit als Debugtool.
Und selbstverständlich ist auch eine geeignete Stromversorgung notwendig, die genug Kapazitäten für eventuelle spätere Erweiterungen aufweist.

Zu den Aufgaben der Software gehörte die Programmierung einer Ansteuerung der Roboterkopfbewegungen, die Realisierung des Roboterverhaltens und die Entwicklung einer übersichtlichen grafischen Benutzeroberfläche.

Zu den zu realisierenden Funktionen der Software gehörten neben der Sprachausgabe auch das Erkennen von Gesichtern, welcher der Roboter mit seinen Kameras erfasst.
Weiterhin war neben den korrekten Mundbewegungen zur Sprachausgabe auch die Umsetzung der Simulation menschen- oder tierähnlicher Verhaltensweisen wie z.B. Augenblinzeln vorgesehen.

Bei allen Arbeiten waren nach Möglichkeit fertige Komponenten oder Verfahren anzuwenden um Zeit und Kosten zu sparen.

Herangehensweise

Zur Umsetzung der Ziele wurde das Projekt in drei Aufgabengebiete getrennt: Mechanik, Elektronik und Softwareentwicklung.
Folgende Herangehensweise wurde für diese einzelnen Bereiche entworfen:

Mechanik

Die originale Mechanik des Roboterkopfes war Anfang Februar bereits entfernt und fast vollständig durch eine eigene Konstruktion mit Servomotoren ersetzt worden. Zum Abschluss dieses Bereichs gehört jedoch auch der Einbau aller Komponenten in ein entsprechend zu bearbeitendes Gehäuse, was natürlich erst nach Abschluss aller elektronischen Arbeiten sinnvoll war.

Elektronik

Insgesamt 11 Servos werden zur Bewegung des Roboterkopfes verwendet.
Hierbei handelt es sich um handelsübliche Modellbauservos mit integrierter Elektronik, welche über ein PWM-Signal (Pulsweitenmodulation) angesteuert werden.
PWM-Signale können sehr einfach mit dem PWM-Generator eines Mikrocontroller erzeugt werden. Ein Beispiel dazu ist auf meiner Website [4] zu finden.

Für dieses Projekt fand sich jedoch eine elegantere Lösung in Form einer USB-Servocontrollerkarte von Parallax.
Sie verfügt über einen integrierten USB-Nach-Seriell-Wandler und ist in der Lage bis zu 16 Servos gleichzeitig in verschiedenen Geschwindigkeitsstufen zu betreiben.
Das serielle Signal ist an der Karte abgreifbar um es einer weiteren Servocontrollerkarte zuzuführen, was sich später als sehr nützlich erweisen sollte.

Komplett auf einen eigenen Mikrocontroller konnte jedoch nicht verzichtet werden, da ja auch ein Display für den Roboter vorgesehen war.
Dieser sollte nicht nur ein 27x4-Zeichen-Display ansteuern, sondern auch das Umschalten des Videosignals auf die Ausgänge der zwei Kameras übernehmen, sowie das Ein-/Ausschalten der Servostromversorgung.

Die Befehle dazu sollte er über eine serielle Schnittstelle erhalten.
Und da die Parallax-Karte ja bereits eine integrierte USB<->Seriell-Schnittstelle besaß und das Signal abgreifbar war, konnte der Mikrokontroller somit einfach an die Servocontrollerkarte angehängt werden.

Da zu Projektbeginn noch nicht absehbar war, wie groß die maximale Leistungsaufnahme letztlich sein würde, wurde zur Stromversorgung ein AT-Netzteil gewählt.
Diese bieten geregelte Spannungen von 5V und 12V und sind bis zu einigen Ampere belastbar.

Software

Zur Gesichtserkennung sollte die Bildverarbeitungsbibliothek „OpenCV“ verwendet werden, mit welcher ich zuvor bereits erste Erfahrungen gesammelt hatte.
OpenCV ist eine freie API von Intel [5] und in C++ geschrieben.
Als Programmiersprache wurde daher Visual C++ gewählt und als Entwicklungsumgebung Microsoft Visual Studio 2005.
Grafische Benutzeroberflächen lassen sich hiermit sehr einfach entwerfen und auch die Kommunikation mit der Parallax-Karte wurde durch fertige Komponenten für serielle Schnittstellen erleichtert.
Da das Projekt sehr modular strukturiert ist, wurde vorgesehen die Gesichtserkennung in eine separate DLL auszulagern, um sie bei Bedarf auch in anderen Anwendungen verwenden zu können.

Zur Sprachausgabe sollte die Microsoft Speech API (SAPI) [6] zum Einsatz kommen.
Die SAPI-Engine ist ebenfalls frei verfügbar.
Ihr größter Vorteil im Vergleich zu anderen Sprachausgabeprodukten ist die große Verbreitung und die Erweiterbarkeit durch zahlreiche Sprachpakete zur Installation weiterer Stimmen und Sprachen.
So wird in diesem Projekt ein Sprachpaket zur Sprachausgabe in Deutsch von einem Drittanbieter verwendet.

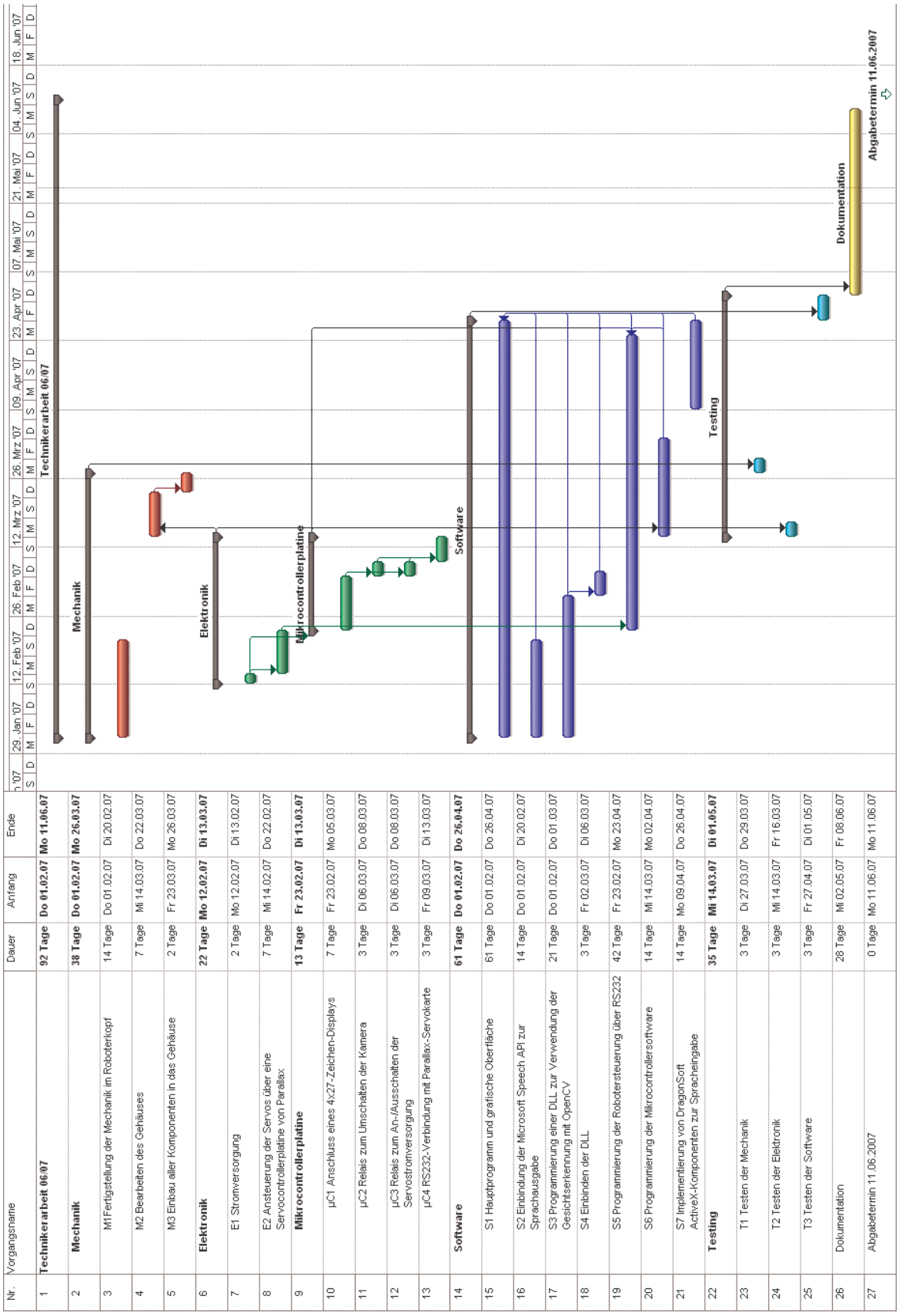
Auch Spracheingabe ist mit der SAPI möglich. Derzeit werden jedoch nur Sprachpakete zur englischen und japanischen Spracherkennung angeboten.
Da der Wunsch bestand, auch Spracheingabe auf Deutsch zu realisieren, musste eine andere Lösung gefunden werden.
Diese fand sich im Laufe der Entwicklung in den ActiveX-Komponenten der Software „Naturally Speaking“ [7] von DragonSoft.

Alle Funktionen, die zum Steuern des Roboterkopfes notwendig sind, sollten in einer Klasse untergebracht werden, auf welche vom Hauptprogramm, bzw. vom Benutzerinterface zugegriffen wird.

Zeitplanung

Basierend auf den Überlegungen zur Herangehensweise des Projektes wurde folgende Zeitplanung aufgestellt:

Nr.	Vorgangsname	Dauer	Anfang	Ende
1	**Technikerarbeit 06/07**	**92 Tage**	**Do 01.02.07**	**Mo 11.06.07**
2	**Mechanik**	**38 Tage**	**Do 01.02.07**	**Mo 26.03.07**
3	M1 Fertigstellung der Mechanik im Roboterkopf	14 Tage	Do 01.02.07	Di 20.02.07
4	M2 Bearbeiten des Gehäuses	7 Tage	Mi 14.03.07	Do 22.03.07
5	M3 Einbau aller Komponenten in das Gehäuse	2 Tage	Fr 23.03.07	Mo 26.03.07
6	**Elektronik**	**22 Tage**	**Mo 12.02.07**	**Di 13.03.07**
7	E1 Stromversorgung	2 Tage	Mo 12.02.07	Di 13.02.07
8	E2 Ansteuerung der Servos über eine Servocontrollerplatine von Parallax	7 Tage	Mi 14.02.07	Do 22.02.07
9	**Mikrocontrollerplatine**	**13 Tage**	**Fr 23.02.07**	**Di 13.03.07**
10	µC1 Anschluss eines 4x27-Zeichen-Displays	7 Tage	Fr 23.02.07	Mo 05.03.07
11	µC2 Relais zum Umschalten der Kamera	3 Tage	Di 06.03.07	Do 08.03.07
12	µC3 Relais zum An-/Ausschalten der Servostromversorgung	3 Tage	Di 06.03.07	Do 08.03.07
13	µC4 RS232-Verbindung mit Parallax-Servokarte	3 Tage	Fr 09.03.07	Di 13.03.07
14	**Software**	**61 Tage**	**Do 01.02.07**	**Do 26.04.07**
15	S1 Hauptprogramm und grafische Oberfläche	61 Tage	Do 01.02.07	Do 26.04.07
16	S2 Einbindung der Microsoft Speech API zur Sprachausgabe	14 Tage	Do 01.02.07	Di 20.02.07
17	S3 Programmierung einer DLL zur Verwendung der Gesichtserkennung mit OpenCV	21 Tage	Do 01.02.07	Do 01.03.07
18	S4 Einbinden der DLL	3 Tage	Fr 02.03.07	Di 06.03.07
19	S5 Programmierung der Robotersteuerung über RS232	42 Tage	Fr 23.02.07	Mo 23.04.07
20	S6 Programmierung der Mikrocontrollersoftware	14 Tage	Mi 14.03.07	Mo 02.04.07
21	S7 Implementierung von DragonSoft ActiveX-Komponenten zur Spracheingabe	14 Tage	Mo 09.04.07	Do 26.04.07
22	**Testing**	**35 Tage**	**Mi 14.03.07**	**Di 01.05.07**
23	T1 Testen der Mechanik	3 Tage	Di 27.03.07	Do 29.03.07
24	T2 Testen der Elektronik	3 Tage	Mi 14.03.07	Fr 16.03.07
25	T3 Testen der Software	3 Tage	Fr 27.04.07	Di 01.05.07
26	Dokumentation	28 Tage	Mi 02.05.07	Fr 08.06.07
27	Abgabetermin 11.06.2007	0 Tage	Mo 11.06.07	Mo 11.06.07



Komplettübersicht

Bevor auf die einzelnen Teile näher eingegangen wird, sei an dieser Stelle zum Zweck der Orientierung vorweg ein Gesamtplan des Projekts mit kurzen Beschreibungen gegeben:

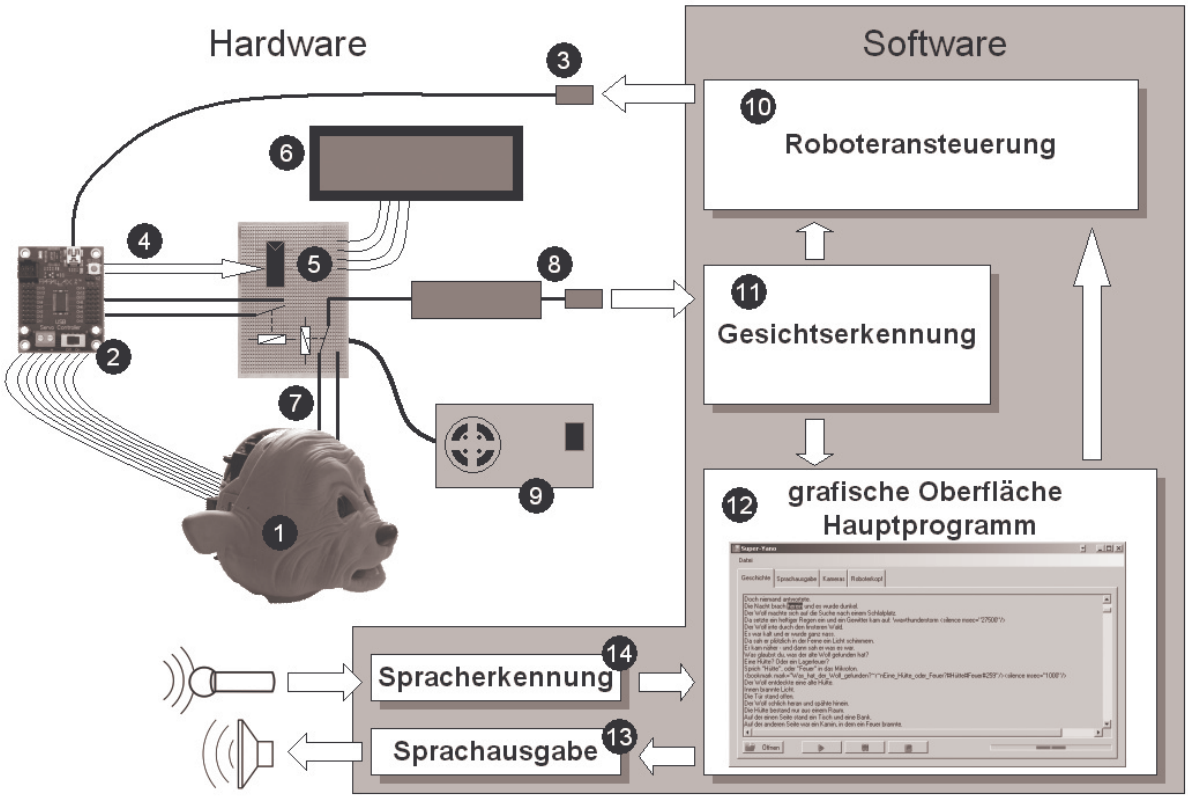


Abb. 6: Schematische Gesamtansicht des Projektes

Im Kopf **(1)** bewegen 11 Servos die Mechanik des Roboters.
Die Servos erhalten ihre PWM-Signale von einer Servocontrollerkarte **(2)**, die über einen USB-Nach-Seriell-Wandler mit dem PC verbunden ist **(3)**.
Das RS232-Signal ist an der Servokarte abgreifbar **(4)** und wird einer Mikrocontrollerplatine **(5)** zugeführt.
Der Mikrocontroller steuert ein 27x4-Zeichen-Display **(6)** und schaltet über Relais die Servostromversorgung an/aus sowie zwischen den Kamerasignalen **(7)** um.
Das Videosignal der gewählten Kamera gelangt über eine USB-Videograbberkarte zum PC **(8)**. Im Gerätemanager erscheint die Grabberkarte als Bildbearbeitungsgerät.
Zum Bereich der Hardware gehört natürlich auch die Stromversorgung, wozu ein AT-Netzteil **(9)** Verwendung findet.
Zum Softwarebereich gehört die Klasse RobotControl, welche die Roboteransteuerung **(10)** regelt.
Die Gesichtserkennung **(11)** benutzt das Videosignal der Kameras, wertet es aus und stellt die Ergebnisse anderen Modulen zur Verfügung.
Das Hauptprogramm **(12)** zeigt eine grafische Benutzeroberfläche und greift über die Roboteransteuerung auf den Roboter zu.
Zudem liest es über Sprachausgabe Texte vor **(13)** und nimmt verbale Benutzereingaben über Spracheingabe entgegen **(14)**.

Mechanik

Yano

Yano ist der Name einer von „The Original San Francisco Toymakers“ hergestellten elektronischen Spielzeugpuppe [8].

Abb. 7: Yano

Die Puppe ist ein interaktiver Geschichtenerzähler für Kinder von 3 bis 6 Jahren. In einen Slot können spezielle Karten gesteckt werden, die den Yano veranlassen eine Geschichte zu erzählen.
Währenddessen fragt er gelegentlich nach, wie der weitere Verlauf der Geschichte ablaufen soll. Über eine IR-Fernbedienung lässt sich dann eine Variante auswählen.

Während des Erzählens bewegt er seinen Kopf, einen Arm, blinzelt mit seinen Augenlider, wackelt mit den Ohren und bewegt seine Augenbrauen.

Yano kommt in Gestalt eines etwa 30cm hohen Kobolds daher und besitzt ein ausdrucksstarkes Gesicht aus Gummi.
Somit erwies er sich für die Anforderungen dieses Projekts als sehr zweckmäßige Basis und wurde – zum Bedauern meiner Tochter – demontiert und umgebaut.

Demontage


Abb. 8: Einblick in Yano

Erster Schritt war das Zerlegen der Yano-Puppe. Von Interesse war nur der Kopf - die Elektronik und Mechanik aus dem Rumpf wurden nicht mehr gebraucht, da ohnehin eine eigene Ansteuerung der Gesichtsmimik vorgesehen war.
Nach dem Abtrennen des Kopfes konnte von diesem das aufgeklebte Kunstfell entfernt und das Gesicht abgenommen werden.

Das Gesicht des Yano besteht aus Gummi und ist mit fünf Druckknöpfen an den Gestängen für Kiefer, Mundwinkel und Augenbrauen befestigt.


Abb. 9: Enthauptet


Abb. 10: Nach Entfernen des Gesichtes

Die Mechanik im Kopf

Im Kopf des Yano werden nur drei Motoren für die Bewegung der Gesichtszüge benutzt. Die Steuerungen für Ohren, Augenlider und Augenbrauen sind mechanisch gekoppelt.

Aufgrund dieser Beschränkungen wäre jedoch nur eine limitierte Vielfalt an Gesichtsausdrücken möglich.

Abb. 11: Kopplung der Gesichtsbewegungen

Darum wurde die originale Mechanik komplett entfernt, um sie durch eine eigene Lösung mit Servos zu ersetzen.
Lediglich die Gestänge zum Bewegen der Gummihaut wurden belassen und weiterverwendet.
Somit ergab sich ausreichend Raum zur Umsetzung eines eigenen Systems.
Abbildung 13 zeigt die Ausgangsposition vor dem Einbau der Servos.

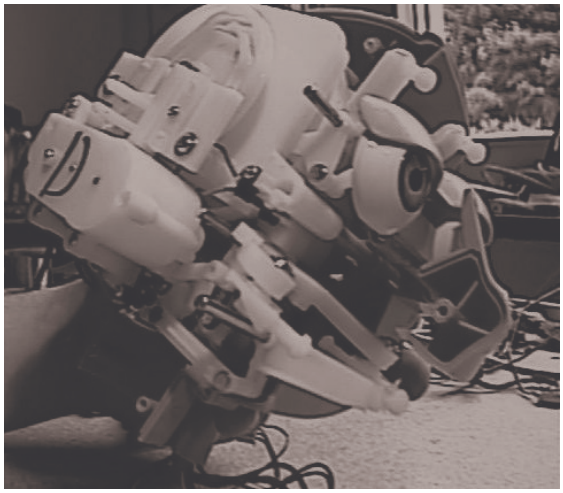


Abb. 12: Die Originalmechanik

Abb. 13: Nur die Gestänge wurden belassen

Einbau von Kameraaugen

Um dem Roboter das Sehen zu ermöglichen, waren zwei drehbare Kameraaugen vorgesehen.
Farbkamera-Module in passender Größe fanden sich bei Conrad Electronic.
Hierbei handelt es sich um analoge CCD-Module mit 1/3“ Bildsensor und den Abmessungen von 22 x 26 x 22 mm.

Abb. 14: Miniatur-Kameramodul

Abb. 15: Kameramodule eingebaut

Die Kameramodule wurden, drehbar gelagert, auf Zahnräder montiert.
Je ein Servo ist für die Drehbewegung zuständig, was es dem Roboter ermöglicht mit den Augen in unterschiedliche Richtungen zu blicken.
Ein Video hierzu findet sich unter [9].

Um ein möglichst natürliches Aussehen zu erzielen, wurden die Augapfelhalbschalen der ursprünglichen Yano-Augen um die Kameralinsen herum geklebt.
Neben dem natürlichen Aussehen wurde somit auch das Justieren der Bildschärfe bzw. des Fokus erleichtert, da dies durch Drehen der Linse eingestellt wird.

Die Augenlider des Yano waren fest miteinander verbunden.
Um sie unabhängig öffnen, bzw. schließen zu können wurden sie mittig getrennt und werden nun über Gestänge jeweils von einem Servo bewegt.
Dabei stellte sich heraus, dass die Gummihaut des Gesichtes der Rückbewegung zuviel Haftreibung entgegensetzt, was zum Hängenbleiben der Lider führte.
Gelöst wurde dieses Problem mit Rückzugfedern, welche die Servomechaniken nun beim Schließen der Lider unterstützen.

Abb. 16: Kameramodule auf Zahnräder montiert

Weitere Einbauten

Nach der Montage der Kameraaugen wurden die weiteren Servos eingebaut.
Die Vorgehensweise war dabei nicht gerichtet, sondern erfogte nach und nach.
So wurden zunächst die Servos zum Bewegen der Mundwinkel, des Kiefers und der Augenbrauen integriert, welche noch problemlos im der linken Kopfhälfte Platz fanden und über die originalen Gestänge die Gummihaut bewegen.

Ab diesem Zeitpunkt traten dann jedoch vermehrt Platzprobleme auf, die oft ausgeklügelter Lösungen bedurften.
So wurde beispielsweise zum Bewegen der Augenlider ein Gestänge zum Ansteuern von Modell-Hubschrauberrotoren in Kombination mit Rückholfedern angewendet.

Um die Wartbarkeit zu erhöhen und um den Kopf möglichst unkompliziert öffnen zu können, sind fast alle Servos in und an die linke Kopfhälfte montiert.
Lediglich das Servo zum Bewegen des rechten Ohres befindet sich in der rechten Kopfschale und das Servo zur Kopfdrehung unterhalb des Kopfes im Elektronikgehäuse.

Bei beiden Ohren, war aufgrund der begrenzten Platzverhältnissen nur eine Lösung über Seilzüge möglich.

Die Rückbewegung erfolgt dabei über Zugfedern, während das Servo über ein Seil (beim linken Ohr zusätzlich über eine Seilrolle) die Bewegung steuert.
Gesucht wurde also ein Seil, bzw. eine Schnur, welche enge Biegeradien zulässt, sowie reiß- und abriebfest ist.

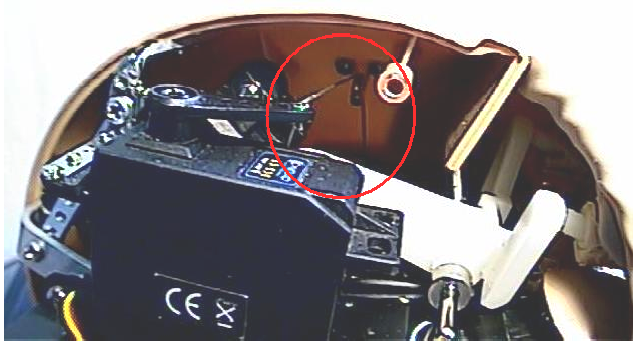


Abb. 17: Seilzug und Umlenkrolle für das linke Ohr

Nylonseil erwies sich bei Testversuchen als zu brüchig und nicht ausreichend haltbar. Erst ein 0,6mm Drahtseil aus einem alten Nadeldrucker hielt den Anforderungen stand.

Um alle Servos sicher zu befestigen, wurden dazu angepasste Halterungen aus Aluminium- und Stahlprofilen angefertigt.

Sämtliche Verbindungen sind, um den Roboter zu Reparaturzwecken demontieren zu können, geschraubt.

Abbildung 18 zeigt einen Einblick in die linke Kopfhälfte im fertigen Zustand des Roboters und verdeutlicht die räumlichen Verhältnisse:

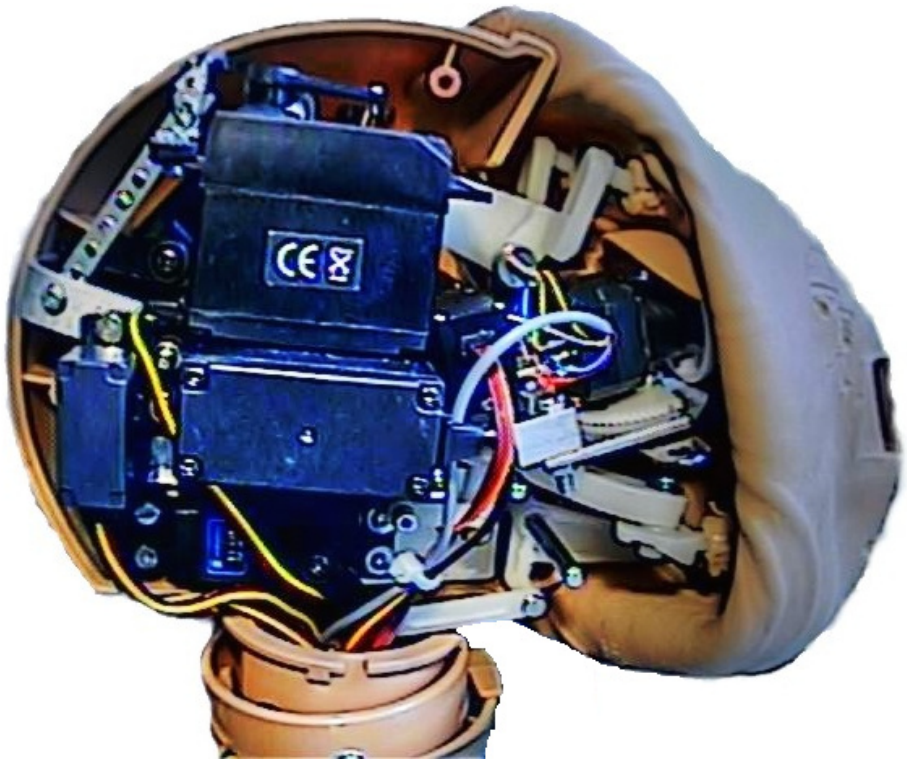


Abb. 18: Einblick in den fertigen Roboterkopf mit weggeklappter Gummihaut.
Nicht mit im Bild: rechte Kopfhälfte mit zweitem Ohr-Servo

Gehäuse

Nach die Arbeiten an der Elektronik abgeschlossen waren, konnte ein adäquates Gehäuse beschafft und bearbeitet werden, um die Komponenten einzubauen. Dabei war es schwieriger als man glauben könnte, bei den gängigen Elektroniksortimentern ein Plastikgehäuse (Metallgehäuse schieden aufgrund der vielen Bohrungen und Durchbrüche aus) in passender Größe zu finden.

Schließlich fand sich bei Conrad Electronic ein Klemmengehäuse mit den Abmessungen 254 x 180 x 111mm und transparentem Deckel, welcher Einblick auf die Elektronik im Innern gewährt (Abbildung 19).

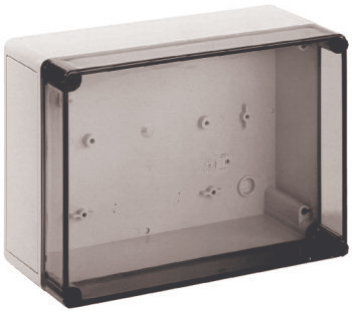


Abb. 19: Gehäuse für die Elektronik

Um einen einfachen Zugang zu den Komponenten zu gewähren und den Roboterkopf einbauen zu können, wurde der Deckel 5cm von der Vorderkante quer durchgesägt. So kann der hintere Bereich problemlos geöffnet werden.
Weitere große Durchbrüche wurden an der Front zum Einlassen des Displays und an der rechten Seitenwand zur Belüftung des Netzteiles angebracht.

Elektronik

Servoansteuerung

Für den Roboter wurden handelsübliche Servos verwendet, wie man sie aus dem Modellbau kennt.

Solche Servos (Abbildung 21 zeigt ein Exemplar) bestehen im Wesentlichen aus einem Elektromotor, einem Getriebe, an welchem ein Servoarm befestigt ist, einem Potentiometer zur Messung des aktuellen Drehwinkels und einer Servoelektronik.
Die Elektronik arbeitet dabei als Regler; sie erfasst mit dem Poti den aktuellen Drehwinkel und vergleicht diesen Wert mit der extern zugeführten Sollvorgabe.

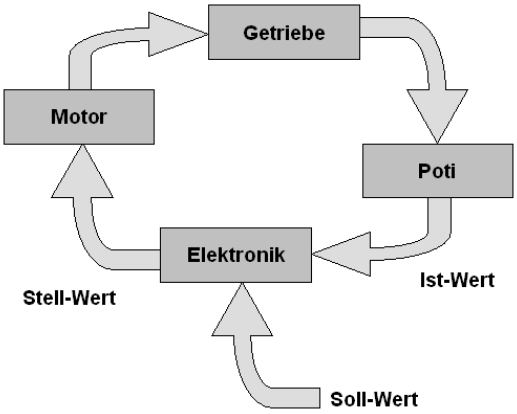


Abb. 20: Schematische Darstellung eines Servo-Regelkreises

Abbildung 20 verdeutlicht den Regelkreis.

Der Sollwert wird Servos als PWM-Signal (Pulsweitenmodulation) zugeführt.
Die Information über den Drehwinkel ist dabei als Pulsdauer kodiert.

Servostellung	Pulsweite
neutral	1.5 ms
ganz links	0.9 ms
ganz rechts	2.1 ms

Dies bedeutet, dass sich der Drehbereich des Servoarmes (mechanisch begrenzt auf einen Bereich von ca. 180°) auf einen Zeitbereich von 1.2 ms skaliert.
Die hier verwendeten Servos arbeiten in 8bit-Auflösung. D.h. sie können den 1.2 ms-Zeitbereich in 256 Schritte (2^8 = 256) unterteilen.
Somit bewirkt eine Pulsdaueränderung von 4.7 µs bereits eine Servoarmbewegung um ca. 0.7°.
Dies ist nur ein Rechenbeispiel, soll aber die zeitlichen Dimensionen verdeutlichen.

Da gängige Mikrocontroller nur bis zu zwei PWM-Kanäle verfügen, hätten zur Erzeugung der erforderlichen PWM-Signale entweder 6 Mikrocontroller parallel arbeiten müssen, oder die Signale hätten durch Software emuliert werden müssen. Beides wäre hier kein zufriedenstellendes Verfahren.

Praktischerweise fand sich dann jedoch eine ebenso einfache wie zweckdienliche Lösung in Form einer Servocontrollerkarte von Parallax Inc. (Abbildung 22).

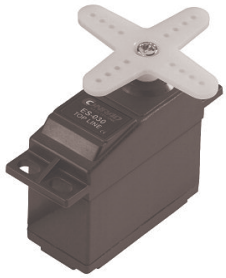


Abb. 21: Modellbauservo

Parallax Servocontrollerkarte

Die Parallax Servocontrollerkarte [10] kann 16 Servos unabhängig mit unterschiedlichen Geschwindigkeiten ansteuern.
Auf PC-Seite erscheint die Karte als normaler serieller COM-Port.

Abb. 21: Parallax Servocontrollerkarte

Über ASCII-Befehle auf diese Schnittstelle erfolgt die Ansteuerung der Servos (dazu später mehr im Kapitel zur Softwaredokumentation).
Auf der Karte befindet sich ein USB-Nach-Seriell-Wandler. Dieser und der Chip zur PWM-Erzeugung beziehen ihren Strom über USB.

Die Stromversorgung der Servos muss extern zugeführt werden und kann mit einem Schalter an/aus geschaltet werden.

Weiterhin befindet sich auf der Karte eine Steckerleiste, an der das serielle Signal abgegriffen werden kann.
Dies dient dem Zweck die Steuerbefehle an eine zweite Servocontrollerkarte weiterzuleiten um somit bis zu 32 Servos anzusteuern.
Die Adressierung der Karten erfolgt dabei über einen Jumper.

Mehr als eine Karte war bei diesem Projekt zwar nicht nötig, aber der offene Zugang zum seriellen Signal erwies sich dennoch als sehr dienlich, da es somit einfach einem Mikrocontroller zugeführt werden konnte.

Dieser sollte über Relais zwischen den beiden Kameras umschalten können und ein 4x27-Zeichen Display ansteuern.

So entstand der nachfolgende Schaltplan:

Schaltplan

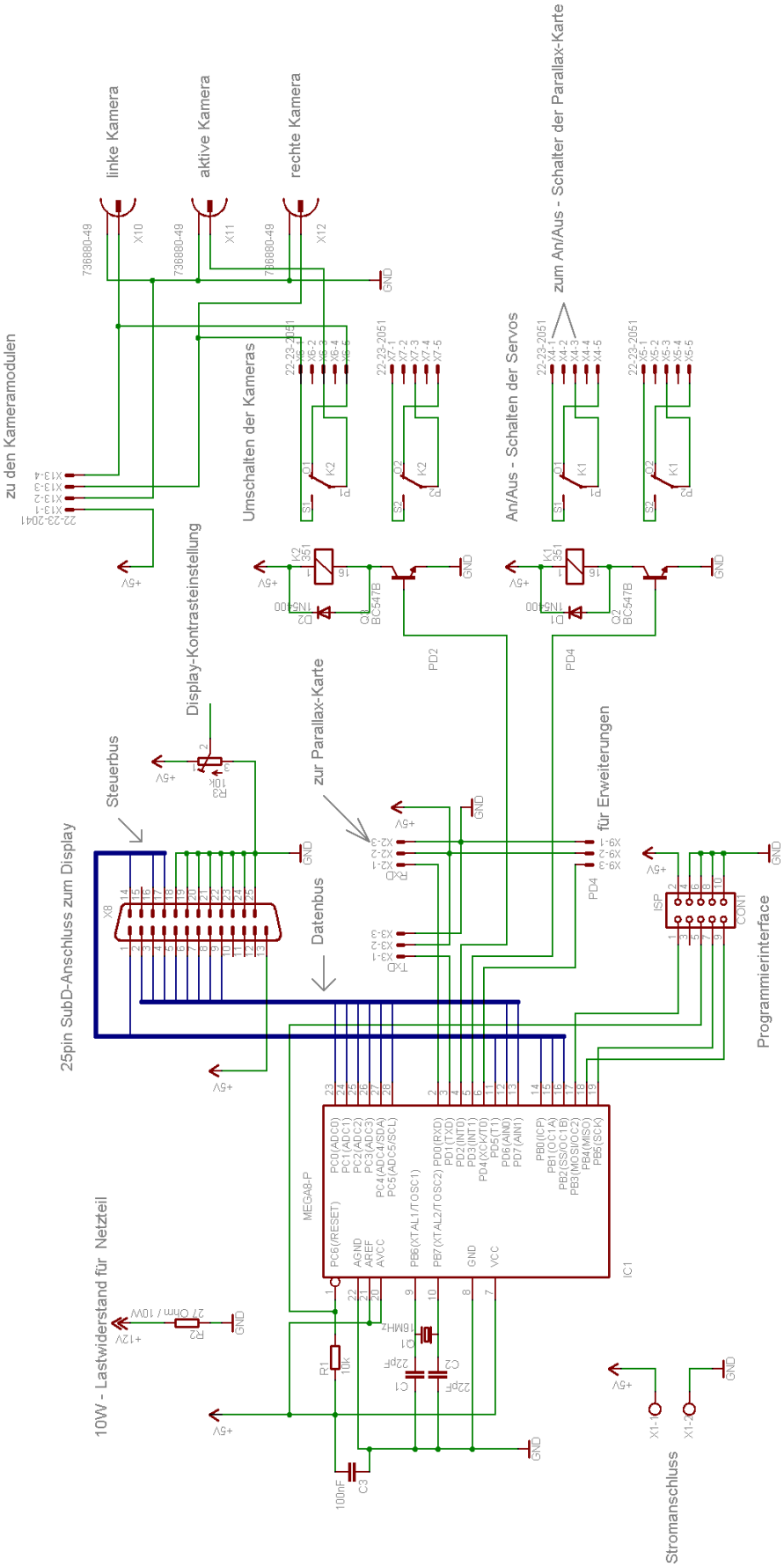


Abb. 23: Schaltplan der Mikrocontroller-Platine

Mikrocontroller-Platine

Herzstück der Platine ist ein ATmega8 Mikrocontroller von Atmel, der hier extern mit 16MHz getaktet ist.
Die hier zu bearbeitenden Aufgaben stellen keine hohen Anforderungen, daher wären auch andere Controller denkbar gewesen. Die Wahl fiel auf diesen Typ, da er gerade vorhanden war und ein passendes Programmiergerät (Flasher) vorlag.
Zum Programmieren des Controllers wurde ein ISP-Anschluss auf der Platine angebracht.

Wie bereits zuvor beschrieben, konnte das serielle Signal von der Parallax-Karte einfach an den Mikrocontroller weitergeleitet werden.
Dazu wurde auf dessen Platine ein entsprechender Stecker an RxD (Pin 2) angebracht.
Dass der Controller über die serielle Schnittstelle auch sendet, war hier nicht nötig. Für eventuelle zukünftige Erweiterungen wurde jedoch auch TxD (Pin 3) über eine Steckerleiste nach außen geführt.

Das in diesem Projekt verwendete LC-Display stammt von Pollin Electronic [11] und kann 4x27 Zeichen darstellen. Dieses Display wurde zuvor bereits bei einem früheren Delphi-Projekt an einem Druckerport verwendet.
Daher war bereits ein 25pin SubD-Stecker angelötet. Um diesen nicht wieder entfernen zu müssen wurde die Mikrocontroller-Platine mit einer entsprechenden 25pin SubD-Buchse versehen.
Zum Ansteuern werden acht Datenleitungen (D0 bis D7) benötigt, sowie vier Steuersignale.
Somit verbleiben nur noch drei nutzbare Pins am Controller.
Davon wurde eines (Pin 6) für eventuelle Erweiterungen über eine Stiftleiste herausgeführt, die anderen beiden werden zum Ansteuern der Relais verwendet.
Da ein Controllerausgang maximal 20mA Strom aufnehmen kann, können Relais nicht direkt über ein Controllerpin geschaltet werden.
Darum wurden für die Relais Treibertransistoren vom Typ BC547 vorangestellt.
Die Transistoren agieren hier als reine Schalter ohne Vorwiderstände mit sofortiger Übersteuerung. Da die Relais über ihre Impedanz selbst ausreichend den Kollektorstrom begrenzen und die Controllerausgänge über eine interne Strombegrenzung verfügen, mag dieses, zugegeben minimalistische, Vorgehen zwar „unkonform“ sein, dient hier aber völlig hinreichend dem Zweck.

Um Spannungsspitzen beim Schalten der Relais zu unterdrücken wurden an ihnen Dioden entgegen der Betriebsspannung angebracht.

Ferner befindet sich auf der Platine eine 4pin-Steckerleiste zur Stromversorgung der Kameras und zur Aufnahme der Videosignale von selbigen.
Die Videosignale werden an drei Cinch-Buchsen herausgeführt.
An der Mittleren ist dabei das über Relais umschaltbare Kamerasignal abgreifbar.

Aus zeitlichen Gründen konnte keine Platine mehr entwickelt und geätzt werden, darum befindet sich im Elektronikgehäuse des Roboters derzeit noch die Prototypversion auf einer Lochrasterplatine.

Netzgerät

Zur Stromversorgung wurde ein AT-Netzteil aus dem PC-Bereich gewählt.
Diese Computernetzteile haben stabilisierte Spannungen von 5V und 12V und sind bis zu einigen Ampere belastbar.
Da Servos durchaus hohe Ströme benötigen können, war dies das Hauptargument zu dieser Wahl.
Des weiteren sind AT-Netzteile, im Gegensatz zum Nachfolgestandard ATX, direkt über einen Netzschalter an/aus-schaltbar – ein Signal wie bei ATX, um das Netzteil aus dem Standbymodus zu aktivieren entfällt.
Das bei diesem Projekt verwendete Netzteil stammt von Pollin Electronic.
Mit dem Abmessungen von 155x105x72 mm ist es ca. 5 cm schmaler, als typische AT-Netzteile, was hier bezüglich der Platzverhältnisse im Gehäuse vorteilhaft war.

Allerdings stellte sich heraus, dass sich das verwendete Netzteil erst bei einem Leistungsabfall von ungefähr 5W aktivieren ließ.
Diese Last muss direkt nach dem Einschalten anliegen, andernfalls schaltet sich das Netzteil umgehend selbst ab.
Da die Elektronik beim Stillstand der Servos keine 5W verbraucht, musste also ein Lastwiderstand angebracht werden - wenngleich auch mir diese Lösung aus wärmetechnischer Hinsicht nicht gefiel.

Der Widerstand wurde an die bislang ungenutzte 12V Spannung angeschlossen.
Bei 27Ω fließen 0.4 Ampere und somit entsteht eine Last von 5.3W.
Der verwendete Widerstand ist bis 10 Watt belastbar.
Da er eine hohe Abwärme verursacht, die bei längerem Kontakt in der Lage ist unvorsichtige Finger zu verbrennen, wurde er zur Kühlung an das Netzteilgehäuse geklemmt.
Wärmeleitpaste sorgt dabei für einen ausreichenden Wärmeübergang.
Der Widerstand befindet sich nun ca. 2 cm unterhalb der Gehäuseabdeckung.
Auch bei längerem Betrieb konnten bislang keine Wärmestaus im Gehäuse registriert werden.

Der Vollständigkeit halber wurde der Lastwiderstand nachträglich auch in den Schaltplan mit eingetragen (vgl. Abb. 23).

Software

Der langwierigste aber sicherlich auch interessanteste Part dieser Abschlussarbeit war die Softwareentwicklung, die aus diesem Grund ausführlicher geschildert werden soll.

Für diese Abschlussarbeit stand nur ein kurzer Zeitrahmen zur Verfügung, auf zeitintensive Entwicklungsprozesse wie Analysen oder Entwürfe musste daher weitgehend verzichtet werden. Zudem änderten sich im Laufe der Entwicklung immer wieder Prioritäten und anfangs gewünschte Funktionen entfielen oder neue kamen hinzu. Daher wurde die Software nach dem *Extreme Programming Prinzip* entwickelt.

Dennoch lässt sich auch die Software wieder in mehrere Teilaufgaben untergliedern (vgl. Zeitplanung), die überwiegend parallel bearbeitet wurden.
Zum besseren Verständnis werde ich mich daher nicht nach dem chronologischen Entwicklungsprozess halten, sondern zuerst auf die unabhängigsten Teilbereiche eingehen.

Überblick

Zuvor soll jedoch einmal ein grober Überblick über die Softwarearchitektur gegeben werden.
Ein vollständiges Klassendiagramm wäre jedoch zu unübersichtlich.
Daher hoffe ich, mit untenstehender Skizze besser verdeutlichen zu können, welche Komponenten welche Aufgaben beinhalten:

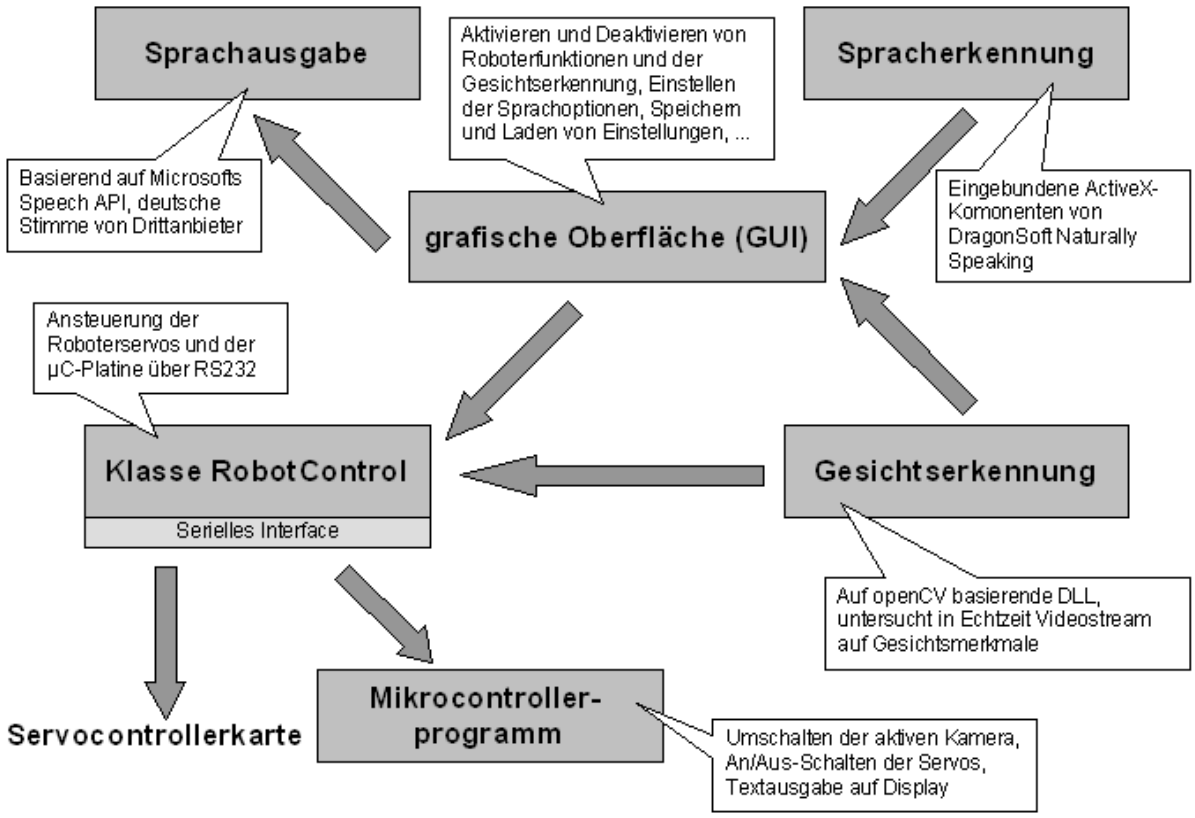


Abb. 24: vereinfachtes Modell der Software

Im Mittelpunkt befindet sich dabei die grafische Oberfläche, da hierhinter auch das Hauptprogramm steht. Dieses verwaltet u.a. sämtliche verbale Kommunikation über Sprachausgabe- und Spracheingabekomponenten.
Weiterhin sind alle Optionen über die grafische Oberfläche einstellbar.
Nicht minder bedeutend ist jedoch auch die Klasse *RobotControl*. Diese steuert über eine serielle Verbindung die Servocontrollerkarte, kommuniziert mit dem Mikrocontroller und regelt das Verhalten des Roboters (worauf später noch detailliert eingegangen werden soll).

Verhältnismäßig kleinere Bereiche sind dagegen das Mikrocontrollerprogramm und die DLL zur Gesichtserkennung, die nun genauer erläutert werden soll.

Gesichtserkennung

Das Sehen ist der wichtigste Sinn des Menschen.
Etwa 80% unserer Wahrnehmung kommt von den Augen.
Bereits Neugeborene reagieren besonders stark auf Gesichter.
Es ist für uns wichtig, unser Gegenüber einschätzen zu können, daher blicken wir, wenn wir einen anderen Menschen sehen, zuerst in sein Gesicht.
Das Nicht-Aufnehmen von Blickkontakt wird in allen Kulturkreisen als abweisendes oder unterwürfiges Verhalten gedeutet.
Blickkontakt ist also auch ein zentrales Kommunikationselement.

OpenCV

Für den Roboter wurde die Erkennung von Gesichtern mit der Softwarebibliothek OpenCV von Intel realisiert. OpenCV ist eine frei verfügbare, in C/C++ geschriebene Sammlung von Algorithmen und Filterfunktionen zur digitalen Bildverarbeitung.
Unter anderem ist in dem Paket eine bereits trainierte Gesichtserkennung integriert. Trainiert bedeutet hier dabei, dass eine XML-Datei vorliegt, in welcher typische Gesichtsmerkmale, die aus verschiedenen Gesichtern erlernt wurden, definiert sind.
Die Erkennungsrate erwies sich dabei, selbst bei ungünstigen Beleuchtungsverhältnissen, als extrem gut. Die Erklärung der dabei im Hintergrund ablaufenden Prozesse und mathematischen Grundlagen würde jedoch ein Buch für sich füllen.

Bei der Verwendung der Funktionen von OpenCV erleichtern zahlreiche Beispielprogramme den Einstieg.
Auch die Integration von OpenCV in ein Visual C++ Projekt lässt sich in wenigen Schritten vollziehen. Eine gute Anleitung hierzu findet sich unter [12].
Dabei ist allerdings ein wichtiger Punkt zu beachten: OpenCV kann nicht ohne Weiteres in ein Projekt mit *managed code* integriert werden!

Managed und unmanaged code

Zur Erklärung: beim „unmanaged code“, der klassischen C/C++ Programmierung, werden alle Objekte vom Programmierer verwaltet. Einmal erstellte Objekte müssen, sobald sie nicht mehr benötigt werden, explizit durch Befehle wieder gelöscht werden, um den belegten Speicher wieder frei zu geben.
Wird dies vergessen entsteht ein sog. *memory leak*, das bei Wiederholung (z.B. in Programmschleifen) letztlich den ganzen Arbeitsspeicher „frisst“ und den Computer lahm legt.
Anders sieht es aus, wenn „managed code“ verwendet wird. Hier müssen Objekte nicht mehr durch eigenen Code freigegeben werden, sondern eine „Speicher-Müllabfuhr“, der *Garbage Collector*, erkennt automatisch ob ein Objekt nicht mehr benötigt wird und entfernt es gegebenenfalls.

Die Vermischung von managed und unmanaged code ist aber nicht ohne Umwege möglich.
Da OpenCV unmanaged code ist, konnte die Gesichtserkennung also nicht direkt in das bestehende Projekt integriert werden, da hier nicht auf die Vorteile von gemanagtem Code verzichten werden sollte.

Lösung mit einer DLL

Wenn man die Gesichtserkennung in eine separate DLL auslagert, lässt sich dieses Problem jedoch einfach umschiffen und aus der Not eine Tugend machen.
Denn so kann die DLL, bzw. deren Funktionalität, zudem auch anderen Anwendungen zur Verfügung stehen und die Modularität der Software wird erhöht.

Es wurde also eine DLL erstellt, die folgende Aufgaben erfüllen sollte:

- Erfassen des Videobildes einer Webcam oder einer anderen Videoquelle
- Untersuchung des Bildes auf Gesichter
- Markierung gefundener Gesichter
- Grafische Darstellung des Videobildes mit Markierungen
- Rückgabe der Anzahl gefundener Gesichter
- Angabe über Position des größten Gesichtes

Um einen Videostream kontinuierlich auf Merkmale (hier Gesichter) zu untersuchen gibt es nun zwei Ansätze.
Das Bild wird in einer (Endlos-)Schleife innerhalb einer Funktion der DLL untersucht, oder die Funktion untersucht nur ein einzelnes Frame des Streams und endet dann. Bei der zweiten Variante, befindet sich die Schleife dann in einem externen Programm und ruft die Funktion der DLL von außerhalb immer wieder auf.

Der Vorteil letzterer Vorgehensweise ist, dass keine weiteren Start- und Stopfunktionen innerhalb der DLL nötig sind um die Schleife zu starten, zu beenden oder die Verzögerungszeit einzustellen.

Videoinitialisierung

Kommen wir nun also zu den Funktionen der DLL.
Bevor die Bildanalyse starten kann, muss ein Capture-Objekt initiiert werden und die XML-Datei mit den Erkennungsmustern geladen werden.
Schlägt dies fehl, so endet die Funktion umgehend und eine globale Variable *isInitiated* wird nicht auf „1“ gesetzt.

Andere Funktionen, welche die Erkennungsmuster erfordern, überprüfen bei ihrem Aufruf zuerst ob *isInitiated* gesetzt wurde und arbeiten nur in diesem Fall weiter.
Abbildung 25 zeigt die Programmlogik der Initialisierungsfunktion.
Der Parameter *camNumber* gibt dabei die Nummer (0, 1, 2, ...) des zu verwendenden Videogerätes an.

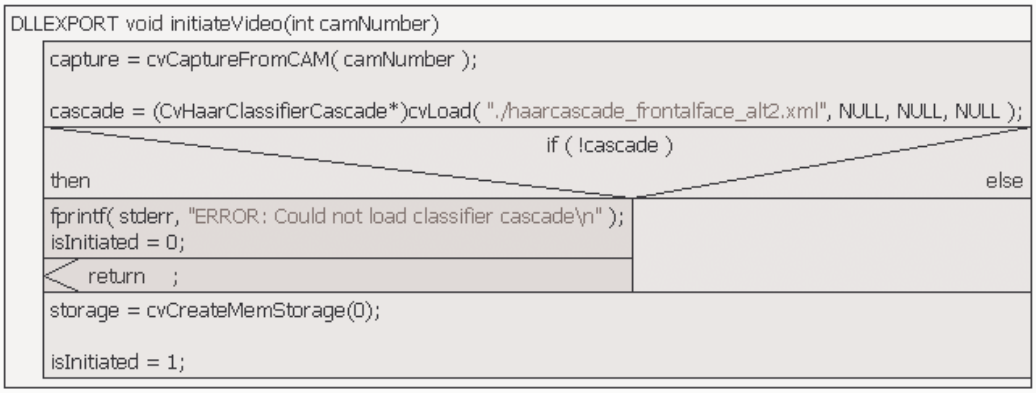


Abb. 25: Initialisierungsfunktion *initiateVideo*

Gesichtserkennung

Um nun ein Frame des Videostreams auf Gesichtsmerkmale zu untersuchen steht die Funktion *processVideoFrame* zur Verfügung (Abb. 26).

Diese überprüft zunächst ob die Initialisierung zuvor gestartet wurde und das Capture-Objekt existiert und endet falls nicht.

Andernfalls wird ein Frame vom Stream eingelesen und, wenn dies erfolgreich war, die Unterfunktion *detect_and_draw* mit dem eingelesenen Bild aufgerufen.

Diese ist die einzige Funktion, die nicht exportiert wird.
In ihr findet die eigentliche Gesichtssuche statt (s. Abb. 27).

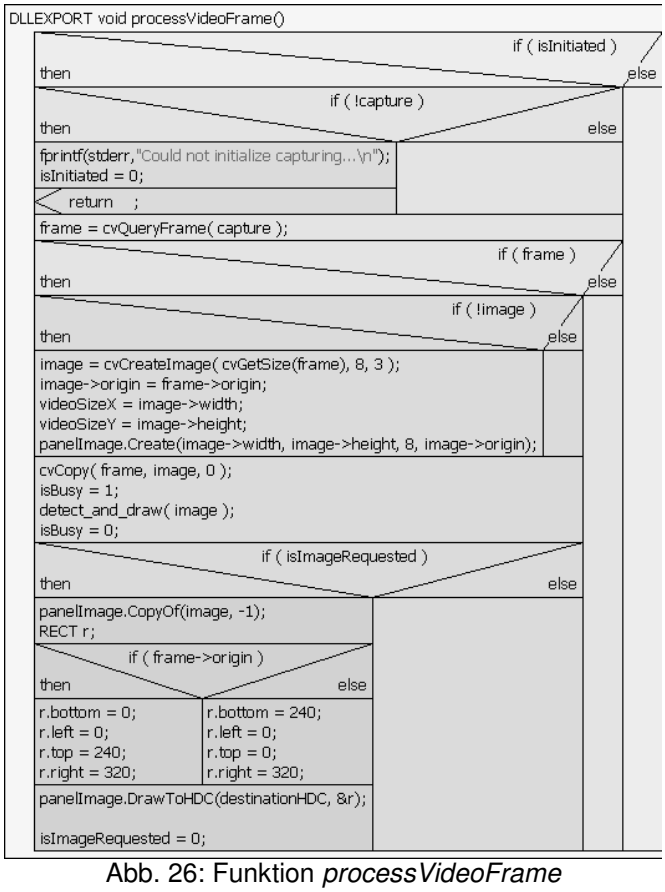


Abb. 26: Funktion *processVideoFrame*

Um die CPU-Belastung dabei in Grenzen zu halten, wird das Eingangsbild zuerst um den Faktor 4 verkleinert
(`double scale = 4.0`).

Bei jedem gefundenen Gesicht wird überprüft, ob es größer als das zuletzt gefundene Gesicht ist, um das größte Gesicht (nach Anzahl der Pixel) zu ermitteln.

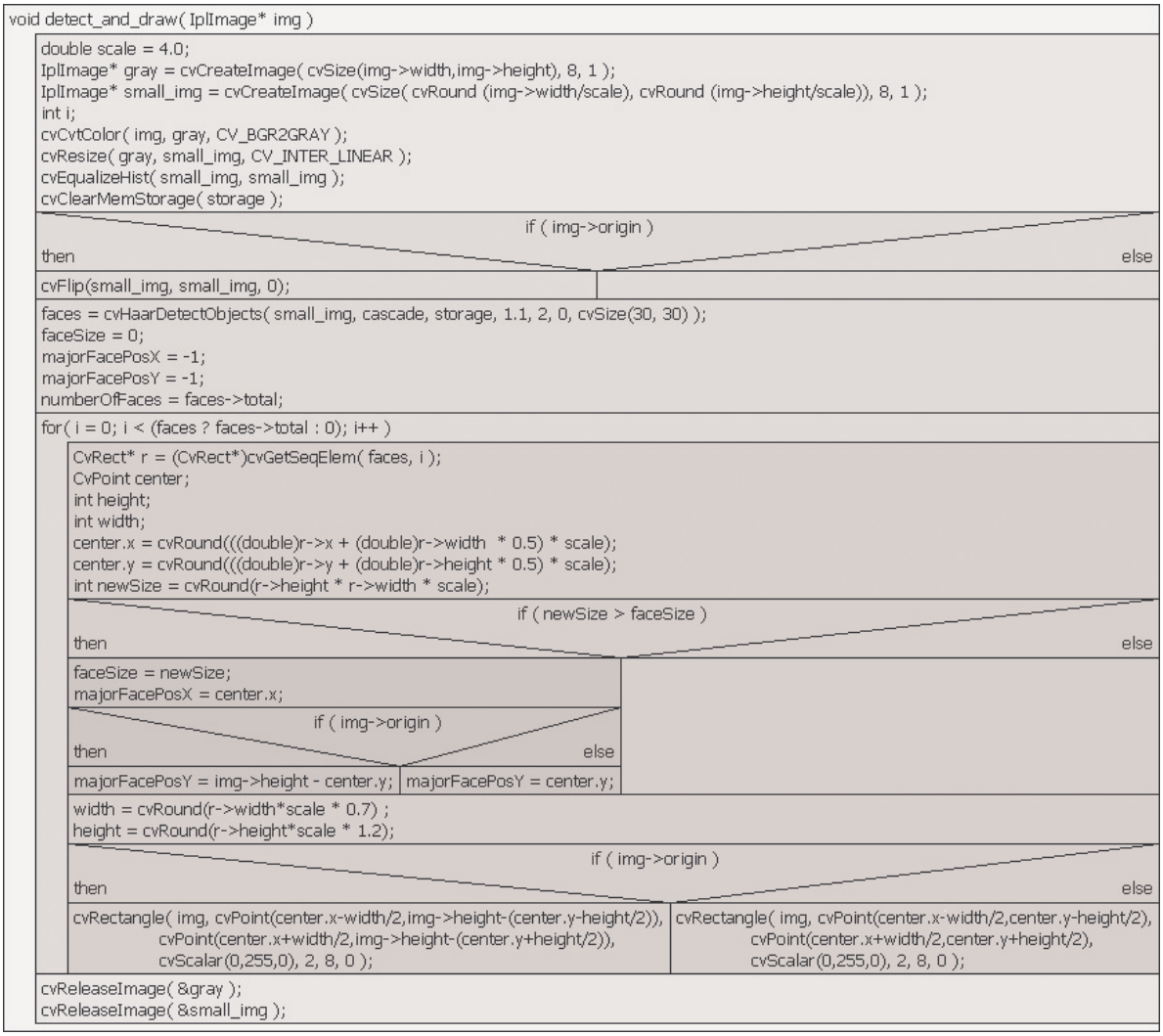


Abb. 27: Funktion *detect_and_draw* zur Gesichtssuche

Um gefundene Gesichter wird in das Eingangsbild schließlich noch ein grüner Rahmen gezeichnet.
Anschließend kehrt der Ablauf zurück zu *processVideoFrame*.

Grafische Ausgabe der Ergebnisse

Bis hierher verlief alles ohne visuelle Ausgabe ab. Nun besteht aber ggf. der Wunsch, sich das Videobild auch live anzeigen zu lassen.
Diese Aufgabe erwies sich als problematisch, da das in OpenCV verwendete Bildformat (IplImage) ein proprietäres ist und nicht einfach in eine Bitmap o.ä. umgewandelt und exportiert werden kann.

Mit der Methode *DrawToHDC* gelang dann eine, etwas aufwendige aber sehr zufriedenstellende, Lösung dieser Problematik.

DrawToHDC erwartet als Parameter einen grafischen Zielgerätekontext und ein Rechteck mit den Koordinaten.
Einen solchen grafischen Kontext (auch Handle genannt) haben alle visuellen Komponenten eines Programms (Fenster, Panel, Buttons, ...).
Über die globale Variable *isImageRequested* wird nun geprüft, ob ein Kontext über die Funktion *requestVideoImage* (Abb. 28) angeben wurde und gegebenenfalls das Videobild in 320x240 Pixel Auflösung dorthin gezeichnet.

```
DLLEXPORT void requestVideoImage(HDC hdc)
    destinationHDC = hdc;
    isImageRequested = 1;
```

Abb. 28: Funktion *requestVideoImage* zur Übergabe eines grafischen Gerätekontext

Neben dieses Funktionen enthält die DLL darüber hinaus natürlich noch einige Zugriffsmethoden um die Anzahl der gefundenen Gesichter sowie um die absoluten und relativen Koordinaten des größten Gesichtes zu ermitteln, auf die hier aber nicht weiter eingegangen werden soll.

Mikrocontrollerprogramm

Herzstück der Mikrocontrollerkarte ist ein ATmega8 µC von Atmel.
Seine serielle Schnittstelle ist an die Parallax Servocontrollerkarte gekoppelt, so dass er die Kommunikation mithören kann.
Befehle an die Servos sind im ASCII-Format. Sie beginnen immer mit „!SC“ und enden mit einem Zeilenumbruchzeichen (0x0D in Hexadezimal).

Befehle, welche die Servocontrollerkarte nicht versteht, werden von ihr einfach ignoriert. Dies ermöglichte es über die gleiche serielle Verbindung auch dem µC Befehle zu schicken.
An den Controller angeschlossen sind zwei Relais und das Display zur Ausgabe von Texten (vgl. Abb.23).
Sein Aufgabenbereich war also überschaubar und weitgehend unkompliziert.

Programmiert wurde das Controllerprogramm mit der freien Entwicklungsumgebung WinAVR [13] in C++.

Main

Die Programmsteuerung wird allein durch eine Interruptroutine bestimmt.
In der main-Methode wird lediglich die Hardware initialisiert, bevor sie in eine leere Endlosschleife übergeht (Abb. 29).

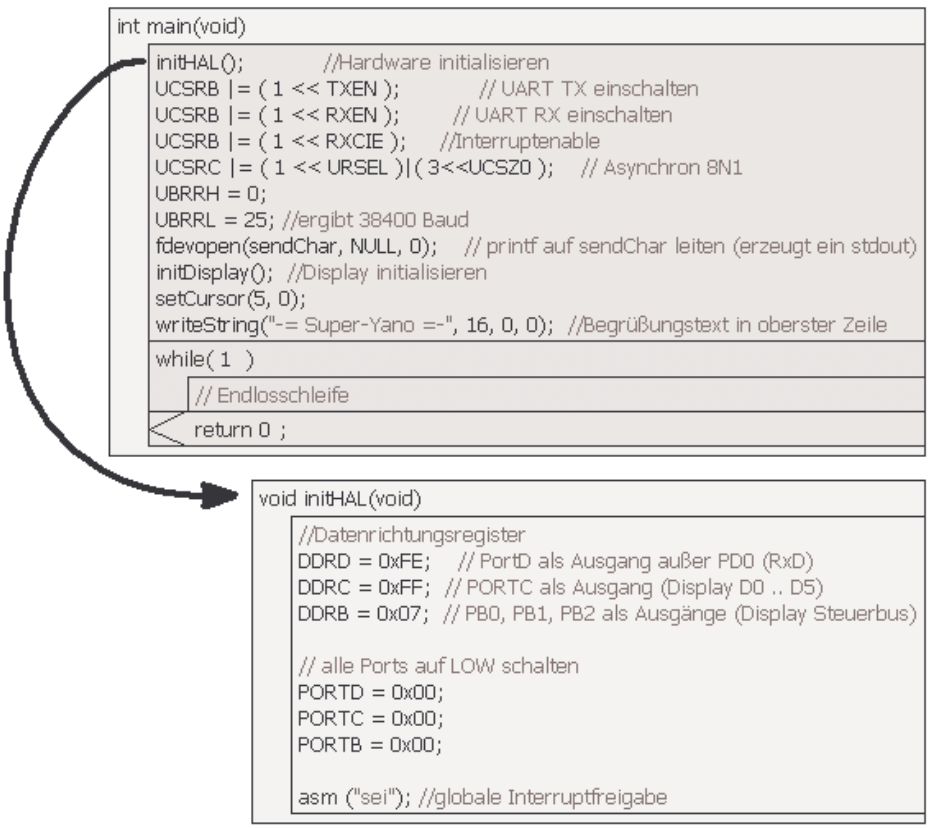


Abb. 29: main-Methode des µC-Programms und Hardwareinitialisierung

Interruptroutine

Interruptroutinen (kurz ISR) werden in WinAVR durch den Methodennamen SIGNAL gekennzeichnet.
Der Parameter gibt dabei an, welches Interruptereignis die Funktion auslöst - hier ist es der Empfang eines Zeichens über die serielle Schnittstelle. Abbildung 30 zeigt die Programmlogik dieser Methode:

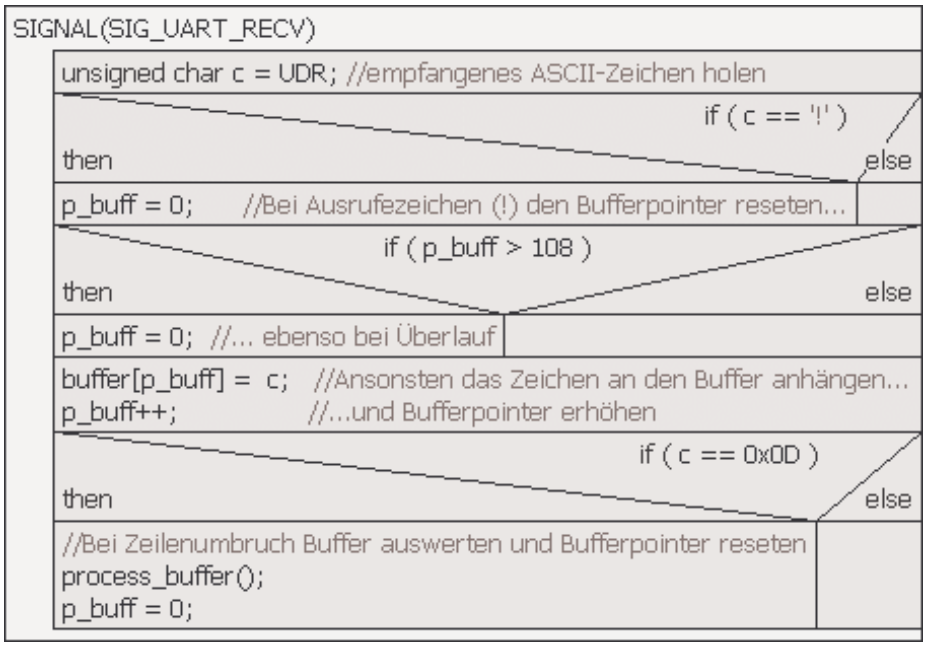


Abb. 30: Interruptroutinen werden bei WinAVR durch *SIGNAL* gekennzeichnet

Um die Funktion zu verstehen, muss man wissen, dass es ein globales Zeichenarray *buffer* gibt, in welchem die empfangenen Zeichen gesammelt werden.
Ferner gibt es eine globale Integervariable *p_buff*, welche als Bufferzeichen dient und die aktuelle Position in dem Zeichenarray angibt.

Bei Empfang eines Zeichens wird nun überprüft, ob es sich dabei um ein Ausrufezeichen handelt. Wenn ja, ist zu erwarten, dass nun ein Kommando folgt und der Bufferzeiger wird auf 0 gesetzt.
Andernfalls wird der Bufferzeiger um eins erhöht und das empfangene Zeichen in das Zeichenarray kopiert.
Bei Empfang eines Zeilenumbruchzeichens (0x0D in Hexadezimal) wird die Funktion *process_buffer* aufgerufen, die den Inhalt des Zeichenarrays *buffer* interpretiert.

Stringauswertung mit *process_buffer*

Diese Funktion ist sehr verschachtelt und umfangreich (vgl. Quellcode).
Sie überprüft als erstes, ob der Buffer mit einem Ausrufezeichen beginnt.
Falls dem so ist, wird das zweite Zeichen betrachtet.
Bei einem „E“ wird davon ausgegangen, dass das „!ENABLE“-Kommando gesendet wurde und das Relais für die Servostromversorgung aktiviert werden soll.
Eine Überprüfung der weiteren Zeichen findet dann nicht statt, so dass - rein theoretisch - auch „!EIN_BEISPIEL“ im Buffer das gleiche Verhalten bewirken würde.

Dies gilt ebenso für den konträren Befehl „!DISABLE“, welcher die Servostromversorgung wieder deaktiviert indem Pin PD3 wieder auf logisch low geschaltet wird.

Ähnlich funktioniert das Umschalten des Kamerarelais. Zur Erinnerung: mit diesem Relais wird das Videosignal der linken, bzw. rechten Kamera auf den mittleren Cinch-Stecker geleitet.
Hier wird erst überprüft, ob das zweite Zeichen in *buffer* ein „C“ ist (für Camera) und dann ob im 5. Zeichen eine „1“ (als ASCII-Zeichen) steht. Eine „1“ bewirkt die Ausgabe des linken Kamerasignals, alles andere die des rechten.

Als vierte Möglichkeit für das zweite Zeichen in *buffer*, wird das „>“-Zeichen behandelt, welches für die Ausgabe eines Wortes auf dem Display vorgesehen ist.
In diesem Fall könnte der Zeichenbuffer also beispielsweise so aussehen:

0	1	2	3	4	5	6	7
!	>	H	a	l	l	o	CR

Nun muss zuerst die Länge des eigentlichen Wortes „Hallo“ ermittelt werden, was gerade der Position des Bufferzeiger p_buff minus drei („!“, „>“ und Zeilenumbruchzeichen CR) entspricht.

Zur Textausgabe werden nur die beiden Zeilen der unteren Displayhälfte verwendet.
In der ersten Zeile der oberen Displayhälfte steht der Titel „Super-Yano“.
Da das Display zwei Controller für jeweils die obere und untere Displayhälfte mit je zwei Zeilen a 27 Zeichen besitzt, wird nachfolgend immer von 1. und 2. Zeile gesprochen, wenn die Ausgabe in der 3. und 4. Zeile gemeint ist.

Das Programm muss also als nächstes herausfinden, ob der Cursor in der 1. oder 2. Zeile steht und ob das Wort noch in die aktuelle Zeile passt.
Ist der Cursor in der ersten Zeile und das Wort passt nicht mehr in diese Zeile, so wird einfach am Anfang der zweiten Zeile weitergeschrieben.
Ist der Cursor allerdings bereits in der zweiten Zeile und das Wort passt hier nicht mehr hinein, so muss zuerst die erste Zeile gelöscht werden und durch die derzeitige zweite Zeile ersetzt werden, bevor das Wort dann an den Anfang der, durch dieses Aufrücken somit frei gewordenen, zweiten Zeile geschrieben werden kann.
Wie man sieht, kann das einfache vertikale Scrollen eines Textes also durchaus eine komplizierte Angelegenheit sein.

Etwas einfacher gestaltet sich die Antwort auf die Frage, wie denn eigentlich Ausrufezeichen auf dem Display ausgegeben werden können.
Wie bereits in der Interruptroutine erwähnt, werden Ausrufezeichen als Startzeichen eines Befehls interpretiert und bewirken, dass der Bufferzeiger auf 0 gesetzt wird. Hintergrund ist, dass Befehle an die Parallax Controllerkarte immer mit „!SC“ beginnen.
Wenn man nun „Hallo!“ auf dem Display ausgeben wollte, ergäbe das ein Problem, da so alle Zeichen bis zum Ausrufezeichen auf diese Weise verloren gehen würden. Die Lösung besteht einfach darin, auf Senderseite (also vom PC-Programm aus) Ausrufezeichen durch 0xFF zu ersetzen (das entsprechende ASCII-Zeichen wird ohnehin nicht benötigt) und es auf Empfängerseite wieder vor der Ausgabe auf dem Display in ein Ausrufezeichen zu wandeln.

Der senderseitig übermittelte String sähe in diesem Beispiel also wie folgendend aus:

0	1	2	3	4	5	6	7	8
!	>	**H**	**a**	**l**	**l**	**o**	**0XFF**	CR

Die Rückumwandlung erfolgt in der Funktion *writeData*, welche eine von mehreren Displayfunktionen ist und nebenbei auch die Umlaute ä, ö und ü sowie das ß-Zeichen an den Displayzeichensatz anpasst.
Groß geschriebene Umlaute (Ä,Ö, Ü) kann das Display leider nicht darstellen. Dieser Mangel könnte sich aber durch benutzerdefinierte Zeichen kompensieren lassen – das Display verfügt über einen RAM-Speicher für 16 selbstdefinierbare Symbole.

RobotControl

Über die Klasse RobotControl findet die Kommunikation mit der Servocontrollerkarte und der Mikrocontrollerplatine statt, sie managt die Servoeinstellungen und steuert das Verhalten des Roboters.
Mit zahlreichen Funktionen und Threads ist sie somit einer der umfangreichsten und aufwendigsten Softwareteile dieses Projekts.
Um den Rahmen dieser Dokumentation nicht zu sprengen, kann hier aus diesem Grund nicht jede Funktion im Detail beschrieben werden.
Ich werde mich daher auf die wichtigsten Bereiche beschränken.

Servosteuerung

Für das Verständnis ist es jedoch unerlässlich zuvor einige grundlegende Sachverhalte zu erklären.
Der Roboterkopf verfügt über insgesamt elf Servos. Von diesen wird nachfolgend immer wieder die Rede sein, daher in Kurzform eine Übersicht über die Nomenklatur und Funktion:

#	Bezeichnung	Funktion
1	ear_l	Linkes Ohr auf/ab bewegen
2	ear_r	Rechtes Ohr auf/ab bewegen
3	lid_l	Linkes Augenlid öffnen/schließen
4	lid_r	Rechtes Augenlid öffnen/schließen
5	eye_l	Linkes Auge drehen
6	eye_r	Rechtes Auge drehen
7	brows	Augenbrauen auf/ab
8	mouth	Mundwinkel auf/ab
9	jaw	Kiefer öffnen/schließen
10	turn	Kopf drehen
11	pitch	Kopf nicken

Bedingt durch die mechanischen Einbauverhältnisse, sind die Servos nicht über ihren gesamten Drehbereich bewegbar, sondern unterliegen hierbei unterschiedlichen Einschränkungen, deren Überschreitung Schäden verursachen würde. Es existieren also Minima und Maxima, *minValue* und *maxValue*, für die Drehwinkel, die für jedes Servo unterschiedlich sein können.
Zwischen diesen beiden Werten gibt es ferner einen Mittelpunkt welcher nachfolgend *neutralValue* genannt wird.
Der *neutralValue* muss aber nicht unbedingt der Ruhezustand eines Servos bei Einschalten und Initialisieren des Roboters sein. So sind die Augenlider beispielsweise beim Start ganz geöffnet und nicht etwa halb offen. Daher gibt es zudem einen Wert *startValue*, der die Servo-Startposition angibt.
Und schließlich ist es im laufenden Betrieb oft hilfreich zu wissen, welche Position ein Servo aktuell eingenommen hat, was im Wert *currentValue* gespeichert wird.

Jedes Servo hat also einen Servonamen sowie fünf spezifische Werte. Um dies organisieren zu können wurde ein Struct *Servo* erstellt und von diesem ein Array angelegt. Dieses Array *servos* hat 16 Felder statt 11, da die Servocontrollerkarte bis zu 16 Servos ansteuern kann.

PSCI-Programm

Bevor ich darauf eingehe, wie RobotControl mit der Servocontrollerkarte kommuniziert, möchte ich an dieser Stelle kurz das PSCI-Programm (Parallax Servo Controller Interface) vorstellen (Abb. 31).

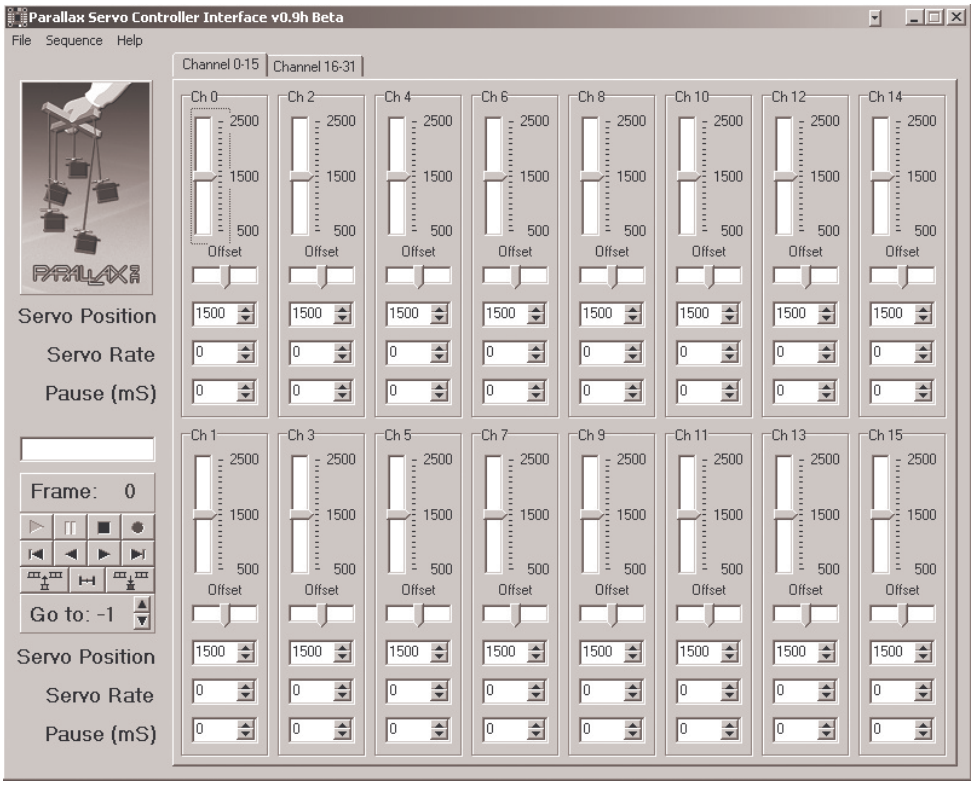


Abb. 31: Parallax Servo Controller Interface

Dieses Programm wird von Parallax mitgeliefert und dient dem Ansteuern der Servos über eine grafische Oberfläche. Wie auf der Abbildung ersichtlich, nimmt hier eine Servoposition Werte zwischen 500 und 2500 ein.
Diese Werte werden vom PSCI programmintern halbiert, bevor sie an die Servos gesendet werden – warum man das macht, erschließt sich mir nicht.
Vielleicht sieht 500-2500 ja benutzerfreundlicher aus als 250-1250...
Wir halten jedenfalls fest, die hier eingestellten Werte sind gerade doppelt so hoch, als die tatsächlich an die Servos gesendeten (vgl. auch Handbuch zur Controllerkarte).
Und da das PSCI zu Testzwecken verwendet wurde um die Minima, Maxima und Startwerte der Servos zu ermitteln, wurde bei dieser Technikerarbeit ebenso verfahren.

Serielle Verbindung

Wie bereits erwähnt, befindet sich auf der Servocontrollerkarte ein USB-Nach-Seriell-Wandler.
Die Karte erscheint also als gewöhnlicher COM-Port im Gerätemanager von Windows, über den sich die Servos mit ASCII-Befehlen ansteuern lassen.

Visual Studio verfügt über eine Reihe visueller und nicht-visueller Komponenten, die zu einem bestehenden Formular einfach per Drag&Drop hinzugefügt werden können. Im Objekteditor sind zur Entwurfszeit dann bereits Attribute und Reaktionen auf Ereignisse einstellbar.
Unter den nicht-visuellen Komponenten findet sich bei Visual Studio praktischerweise auch eine Comport-Komponente.
Es lag also nahe, der Klasse RobotControl solch eine Komponente zur Kommunikation über eine serielle Verbindung zu geben.
Da RobotControl kein visuelles Objekt darstellt, war dies jedoch nicht simpel per Drag&Drop machbar, sondern es bedurfte dazu eines ObjectContainers, welcher „von Hand“ in RobotControl implementiert wurde.

Das Erstellen eines Comport-Objekts erfolgt sodann im Konstruktor der RobotControl-Klasse, welcher nun näher beschrieben werden soll.

Konstruktor

Eine der ersten Aktionen des Konstruktors ist der Aufruf der Funktion *initiateServoSettings*, welche das erwähnte *servos*-Array anlegt und ihrerseits die Funktion *readServoSettings* aufruft.
readServoSettings lädt die Werte (*minValue*, *maxValue*, *neutralValue* und *startValue*) für jedes Servo und speichert diese Werte in den Feldern des Arrays.
Hierzu sollte man wissen, dass bei der von mir entwickelten Software die Servowerte und sonstigen Einstellungen nicht fest einprogrammiert, sondern in einer XML-Datei (super-yano.xml) hinterlegt sind.
Ursprünglich war das Speichern der Einstellungen in einer Ini-Datei vorgesehen. Diese, eigentlich unproblematische und bewährte, Methode scheint jedoch aus der Mode zu kommen und Visual Studio bietet hierzu auch keine vorgefertigten Komponenten an. Stattdessen wurde daher das XML-Format gewählt.
Was sich im Nachhinein als Fehlgriff erwies, da der damit verbundene Overhead in keinem Verhältnis zum Zweck (ein paar Einstellungen sichern) steht...

Nach dem Lesen der Servowerte aus der XML-Datei kehrt der Programmablauf dann wieder zurück zum Konstruktor.
Jetzt wird dem ObjectContainer eine SerialPort-Komponente hinzugefügt, deren Baudrate auf 2400 Baud gesetzt und das Ereignis für den Empfang eines Zeichens mit der Funktion *serialPort_DataReceived* verbunden.

Automatisches Finden der Servocontrollerkarte

Wie schon erwähnt, erscheint die Parallax-Karte im Gerätemanager als gewöhnlicher Comport. Nun hat Windows leider die unpraktische Eigenschaft, diesem Gerät jedes Mal einen anderen Comportnamen (COM1, COM2, ...) zuzuweisen, sobald das Gerät abgesteckt und an einen anderen USB-Port angeschlossen wird.
Um in einem solchen Fall das Rätselraten und Herumprobieren, welcher Port wohl richtig ist zu ersparen, wurde eine Funktion implementiert, die dies automatisch ermittelt.
Dem Konstruktor wurde beim Erstellen von RobotControl durch das Hauptprogramm (Form1.h) ein String-Parameter *port_name* übergeben.
Dieser gibt den Bezeichner des Comports an, der beim letzten Programmlauf verwendet wurde. Der Konstruktor ruft nun die Funktion *findComport* mit dem übergebenen *port_name* auf.
Es ist sicherlich logisch zuerst beim zuletzt verwendeten Comport nach dem Roboter zu suchen.
Nehmen wir beispielsweise an, zuletzt sei der Port COM3 verwendet worden und der Roboter ist nun aber an COM1 angeschlossen.
Der Konstruktor würde also findComport mit Parameter „COM3“ aufrufen.
Nehmen wir weiterhin an, es gäbe vier serielle Schnittstellen COM1 bis COM4.
Dann würde *findComport* ein Stringarray mit fünf Strings erstellen, bei welchem das erste Element der Bezeichner des zuletzt verwendeten Ports ist:

Index	0	1	2	3	4
Wert	COM3	COM1	COM2	COM3	COM4

Aber wie wird daraus nun der „richtige” Port ermittelt?
Die Lösung geht über eine simple Versionsabfrage: wenn an die Parallax-Karte der String „U!SCVER?“ gesendet wird, so antwortet sie mit „U!SCVER?\r1.4“ (wobei \r für das Zeilenumbruchzeichen steht).
findComport macht also nichts anderes, als über alle vorhandenen Comports eine Versionsabfrage an die Servocontrollerkarte zu senden. Antwortet sie, so ist der richtige Port gefunden und die Funktion endet mit der Rückgabe des Bezeichners. Wird die Karte nicht gefunden, so gibt sie einen leeren String zurück.
In diesem Beispiel würde die Funktion bei Index 1 abbrechen und „COM1“ zurückgeben.

Initialisierung

Die Parallax-Karte beherrscht die Baudraten 2400 und 38400 Baud.
Standardmäßig ist sie auf 2400 Baud eingestellt.
Dies erwies sich als Flaschenhals, wenn der Roboter Texte schnell vorliest.
Denn nicht nur die Servowerte der Mundbewegungen für jedes Phonem werden über die Leitung gesendet, sondern zudem jedes Wort, jeder vorgelesene Buchstabe zur Ausgabe auf dem Display.

Im Konstruktor wird daher die Servocontrollerkarte für eine höhere Baudrate eingestellt.
Hierzu gibt es das Kommando „U!SCSBR“. An diesen String wird der Hexwert 0x00 für 2400 Baud oder 0x01 für 38400 Baud angehängt.

Sobald die Karte konfiguriert ist, wird über den Aufruf der Funktion *initiateRobot*, welche ihrerseits über die Funktion *setServoValue* Startwerte an die Servos sendet, der Roboterkopf in seine Startstellung gefahren (vgl. Abb. 32).

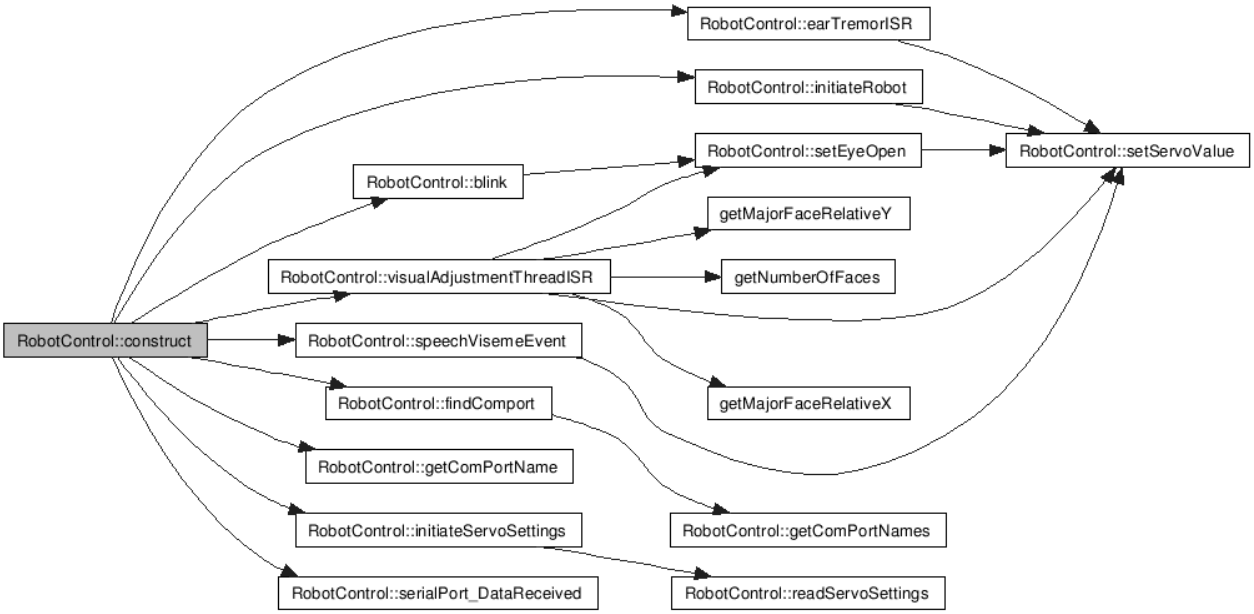


Abb. 32: Aufrufabhängigkeiten ausgehend vom Konstruktor der RobotControl-Klasse

Weiterhin wird im Konstruktor ein Eventhandler *spVisemeEventHandler* für das Sprechen von Phonemen erzeugt und drei Threads gestartet, welche das Verhalten des Roboters steuern und auf die nachfolgend eingegangen werden soll.

Sprachsynchrone Mundbewegung

Wie in dieser Dokumentation später noch beschrieben wird, existiert im Hauptprogramm ein Voice-Objekt, welches der Sprachausgabe dient.
Beim Sprechen erzeugt das Voice-Objekt verschiedene Ereignisse.
Diese „Events“ lassen sich mit Funktionen koppeln, um auf bestimmte Geschehnisse bezüglich der Sprachausgabe zu reagieren.
Ein für den Roboter relevantes Event ist die Aussprache eines Phonems.

Phoneme sind die kleinsten bedeutungsunterscheidenden, aber nicht bedeutungstragenden Einheiten einer Sprache [...].
Beispiele für deutsche Phoneme:
/p/, /t/, /k/ (stimmlose Plosive)
/m/, /n/, /ŋ/ (Nasale)
/a:/, /a/, /e:/, /ɛ/ (lange und kurze Vokale)
[...] *Phoneme tragen für sich genommen keine Bedeutung, ersetzt man jedoch in einem Wort ein Phonem durch ein anderes, ändert sich die Bedeutung: „Katze“ vs. „Tatze“, „Lamm“ vs. „lahm“, „Beet“ vs. „Bett“, „rasten“ (kurzes a) vs. „rasten“ (langes a). Dies ist mit „bedeutungsunterscheidend“ gemeint.*
(Quelle: wikipedia.org, 01.06.2007)

Um die Funktion *speechVisemeEvent*, welche die Mundbewegungen steuert, mit dem Ereignis der Aussprache eines Phonems des Voice-Objekts zu koppeln, sind drei Schritte notwendig.

Zuerst wird im Konstruktor von RobotControl ein Eventhandler-Objekt *spVisemeEventHandler* erzeugt und mit der Funktion *speechVisemeEvent* gekoppelt.
Dann muss der Zugriff auf dieses Objekt von außerhalb der Klasse gewährt werden, da das Voice-Objekt nicht zu RobotControl gehört - die Funktion g*etVisemeEventHandler* ermöglicht dies.
Und schließlich muss im Hauptprogramm das Eventhandler-Objekt aus RobotControl mit dem Phonemevent des Voice-Objekts verknüpft werden.

Nun wird bei jedem gesprochenen Phonem die Funktion *speechVisemeEvent* aufgerufen.
Die verwendete Speech API von Microsoft unterscheidet 22 verschiedene Phoneme:

0	silence
1	ae ax ah
2	aa
3	ao
4	ey eh uh
5	er
6	y iy ih ix
7	w uw
8	ow
9	aw
10	oy

11	ay
12	h
13	r
14	l
15	s z
15	sh ch jh zh
17	th dh
18	f v
19	d t n
20	k g ng
21	p b m

Jedes Phonem repräsentiert eine andere Mundform.
Leider ist der Roboter aufgrund seiner mechanischen Einschränkungen nicht in der Lage einen unterschiedlich breiten Mund zu ziehen, sondern kann nur mit seinem Kiefer und dem Heben und Senken der Mundwinkel die Phoneme versuchen zu simulieren.

Aufgrund dieser Einschränkung und weil sich viele Phoneme in ihrer Mundform sehr gleichen, wurden mehrere ähnliche Phoneme zusammengefasst, so dass nun 10 unterschiedliche Mundstellungen geformt werden.
Hierbei wurde nebenstehendes Bild von [14] als Vorlage verwendet und mit der PSCI-Software versucht jeweils eine ähnliche Mundform einzustellen um die notwendigen Servowerte zu ermitteln (s. Abb. 33).

Wenn nun die Funktion *speechVisemeEvent* aufgerufen wird, so gibt der Parameter *currentVisemeId* die Nummer des Phonem an (vgl. oben stehende Tabelle).
Und abhängig hiervon schickt *speechVisemeEvent* dann die mit PSCI ermittelten Servowerte an die Servos für Kiefer (jaw) und Mundwinkel (mouth).

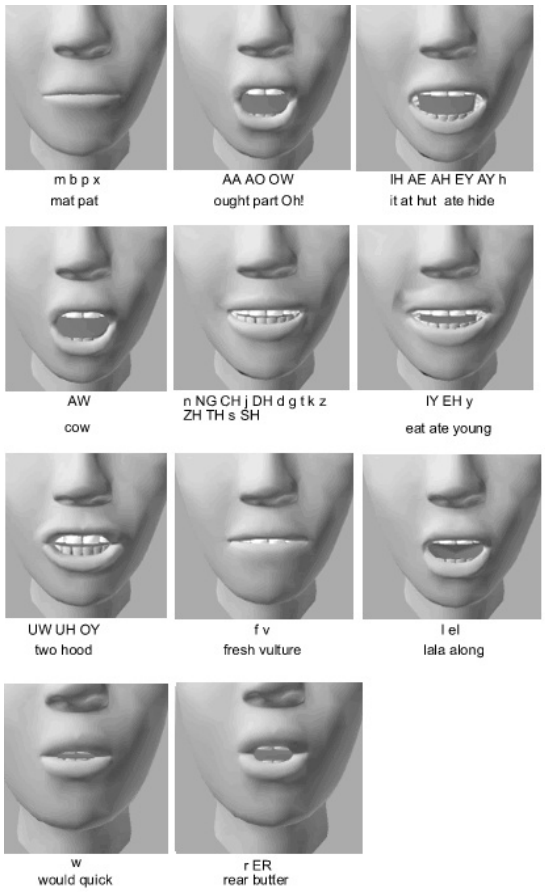


Abb. 33: Mundformen verschiedener Phoneme

Menschliches und tierisches Verhalten

Eine der wichtigsten Aufgaben dieses Roboters ist es, auf Personen lebhaft und interessant zu wirken um als Gesprächspartner akzeptiert zu werden.
Dieses Ziel würde er jedoch verfehlen, wenn er in den Zeiten, in denen er nicht spricht, statisch verharren würde.
Um ihn also auch in den passiven Zeiten lebendig wirken zu lassen wurde er programmiert mit den Augen zu blinzeln, mit den Ohren zu wackeln und sich auf Gesichter auszurichten.

Augenblinzeln

Das Blinzeln der Augen bewirkt nicht nur, dass der Roboter etwas tut, was die Aufmerksamkeit auf ihn lenken könnte, sondern ist darüber hinaus auch ein wichtiger Aspekt der nonverbalen Kommunikation.

[...] *so lässt sich feststellen, dass ein durchschnittlich frequenziertes Blinzeln Sympathie vermittelt, hingegen eine erhöhte Frequenz eine sehr höfliche oder ironische Einstellung des Gegenübers bedeutet.*
Ist der Wimpernschlag nicht vorhanden spricht man von Starren [...]*. Dieses Wort ist aus diesem Grunde negativ konnotiert, da eine anthropologische Antipathie zu erwarten ist.*
Eine These spricht davon, dass man Vertrauen durch Blinzeln verdeutlichen kann, hingegen das Starren notwendig ist, um in Kampfsituationen keine Schwachstelle zu bieten und schneller reagieren zu können.
Des Weiteren fällt [...] *auf, dass das Blinzeln nicht unerheblich für Sympathie sein kann. Ironie, das Gegenteil des eigentlich Gemeinten, kann durch das absente Augenblinzeln entscheiden sein.*
(Quelle: wikipedia.org, 01.06.2007)

```
void RobotControl::blink(void)
    long sleepTime;
    Random^ rnd = gcnew Random();
    while( blinkingState != STOPPED )
        while( blinkingState == RUNNING )
            sleepTime = rnd->Next(20000, 30000);
            for( int i=0; i< (sleepTime/100); i++ )
                if ( blinkingState == RUNNING )
                then                          else
                Thread::Sleep(100);           break;
            // kurzes Blinzeln
            setEyeOpen(LEFT, false);
            Threading::Thread::Sleep(50);
            setEyeOpen(RIGHT, false);

            Threading::Thread::Sleep(200);

            setEyeOpen(LEFT, true);
            Threading::Thread::Sleep(50);
            setEyeOpen(RIGHT, true);
        Thread::Sleep(100);
```

Abb. 34: Funktion *blink*

Um das Blinzeln beim Roboter zu simulieren existiert hierzu ein eigener Thread *blinkThread*. Dieser wird im Konstruktor von RobotControl erstellt und mit der Funktion *blink* verknüpft (s. Abb. 34).

Das menschliche Auge blinzelt ca. alle 5 bis 6 Sekunden.
Dies geschieht reflexhaft und wird vom Gehirn auch nicht bewusst wahrgenommen (andernfalls würden wir regelmäßig kurz schwarz sehen).
Versuche mit dem Roboter bei gleicher Frequenz zeigten jedoch überraschenderweise, dass es völlig übertrieben wirken würde.
Darum wurde das Intervall auf einen (bei jedem Blinzeln zufällig ermittelten) Bereich zwischen 20 und 30 Sekunden ausgedehnt.

Diese Wartezeit bis zum nächsten Blinzeln wird von der Funktion in „Häppchen“ zu je 100ms unterteilt.
Der Grund hierfür liegt darin, dass bei Programmende alle Threads beendet werden müssen, was bei diesem Thread durch das Setzen der Variable *blinkingState* auf *STOPPED* geschieht.

Wenn aber gerade erst geblinzelt wurde, würde das Programm noch bis zu 30 Sekunden weiterlaufen bis die Bedingung der äußeren Whileschleife überprüft werden würde.
Durch das Stückeln der Wartezeit aber, würde der Thread so im ungünstigen Fall 100ms warten, einmal kurz blinzeln, noch einmal 100ms warten und dann enden (s. Abb. 34).

Die letzten 100ms haben ihren Sinn darin, dass der Thread auch in einen *PAUSED*-Modus versetzt werden kann.
In diesem Fall dienen sie dazu den Thread kontinuierlich im Wartezustand zu halten (ein Thread ganz ohne Warteverzögerung würde die CPU völlig für sich in Beschlag nehmen und das System lahm legen).

Ohrenwackeln

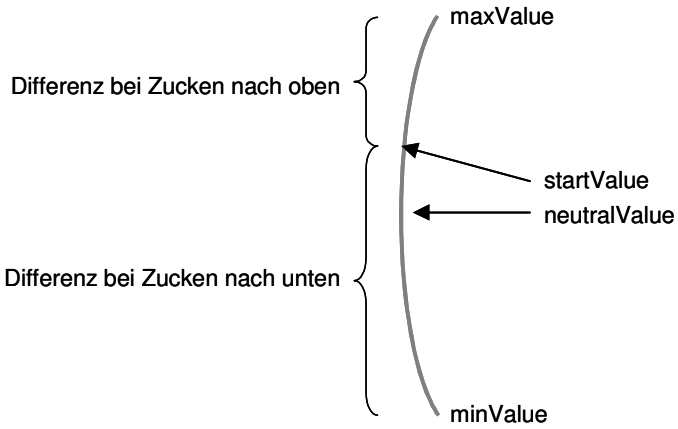
Von Tieren, genauer gesagt von Kühen, wurde ein weiteres Verhalten nachempfunden: das Zucken mit den Ohren, welches Rinder tun um lästige Fliegen zu verscheuchen.
Es klingt zunächst albern dies auf einen Roboter zu übertragen, hat aber einen durchaus beeindruckenden Effekt auf Versuchspersonen.

Auch dieses Verhalten wird wieder durch einen Thread *earTremorThread* (tremor = engl. Zittern, zucken) gesteuert, welcher im Konstruktor mit der Funktion *earTremorISR* verknüpft wird.
Da diese Funktion sehr diffizil gestaltet und bis ins kleinste Detail dem natürlichen Vorbild nachempfunden ist, soll sie hier etwas ausführlicher beschrieben werden.

Zunächst wird wie beim Blinzeln eine Wartezeit in 100ms Schritten abgewartet, die hier, bei jedem Durchlauf zufällig bestimmte Werte, zwischen 30 und 90 Sekunden betragen kann.
Ebenfalls zufällig bestimmt wird, welches der beiden Ohren zucken soll und wie oft direkt hintereinander (1 bis 3 mal).
Der Ausschlag des Ohres ist dabei umgekehrt proportional zur Anzahl der Wiederholungen.

Um noch mehr Varianz zu erhalten, wird darüber hinaus auch bestimmt, ob das Ohr nach unten oder nach oben zucken soll.

Um das Ohr also nun zucken zu lassen, wird, abhängig von der Richtung des Zuckens, die Differenz zwischen Startwert und Maxima, bzw. Minima, des Servos bestimmt.



Diese Differenz *diff* wird dann durch die Anzahl der Zuckbewegungen dividiert.
Ein einzelnes Zucken erscheint dadurch stärker, während ein dreifaches Zucken nur schwach ausgeprägt ist.

Also beim Zucken nach oben:

diff = (maxValue – startValue) / Anzahl der Zuckungen

und nach unten:

diff = (startValue - minValue) / Anzahl der Zuckungen

Bei einem Zucken nach oben, wird das Servo nun zuerst die Position *startValue+diff* anfahren, kurz warten und dann wieder die Startposition einnehmen, während es sich beim Zucken nach unten zwischen Startposition und *startValue-diff* bewegt.

Dieser Vorgang wird bis zu dreimal kurz hintereinander ausgeführt, bevor der Thread wieder 30 bis 90 Sekunden wartet.

Abbildung 35 zeigt den vollständigen Programmablauf der Funktion:

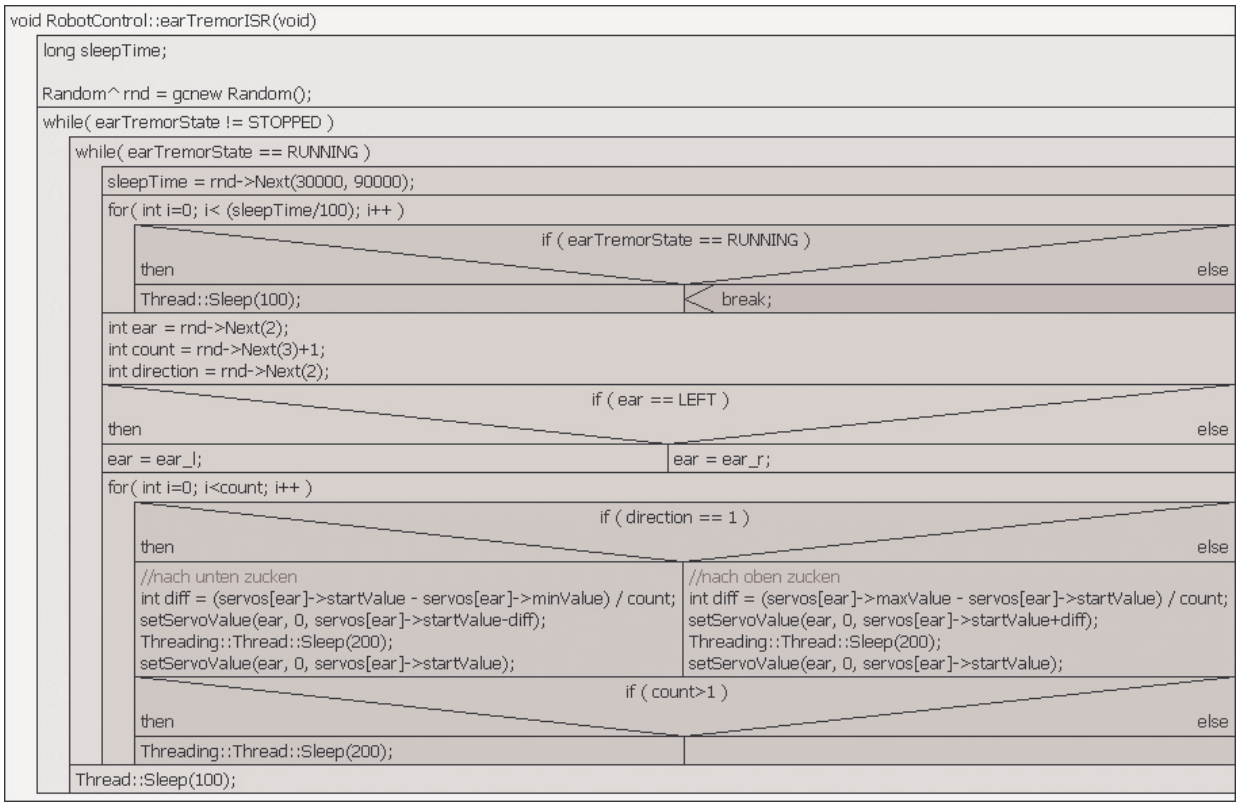


Abb. 35: Programmablauf der Threadfunktion *earTremorISR* zum Zucken der Roboterohren

Gesichtstracking

Eine der komplexesten Funktionen der gesamten Software ist die zum Gesichtstracking.
Die Funktion *visualAdjustmentThreadISR* wird im Konstruktor mit dem Thread *visualAdjustmentThread* verknüpft. Sie ist dafür zuständig, dass der Roboter seine Augen auf ein Gesicht fixiert und seinen Kopf und seine Augen dreht um das Gesicht im Zentrum des Blickfeldes zu halten.
Sie ist, soweit es die Bewegungsfreiheitsgrade des Roboters zulassen, am menschlichen Vorbild orientiert.

Um ihre Funktionsweise zu verstehen, muss man sich also zuerst Gedanken darüber machen, wie wir unseren Blick auf etwas richten.
Man stelle sich vor, ein interessantes Objekt befindet sich am Rand unseres Gesichtsfeldes.
Nun wird man meistens zuerst die Augen auf die entsprechende Stelle drehen.
Wenn das Objekt ganz am Rand des Gesichtfeldes ist und man die Stelle länger als nur flüchtig betrachtet, so wird man dann mit dem Kopf nachziehen, weil es als anstrengend empfunden wird, den Blick längere Zeit auf einen Punkt am Blickfeldrand zu fokussieren.
Wobei hierbei allerdings individuell unterschiedliches Verhalten möglich ist. So gibt es Personen, die den Kopf nur wenig drehen, während Brillenträger oft gleich den ganzen Kopf mitbewegen und dafür die Augenbewegung weniger nutzen.
Weiterhin gibt es, ebenfalls individuell unterschiedlich, einen Bereich im Zentrum des Blickfeldes, bei dem Objekte ausschließlich durch das Ausrichten der Augäpfel fokussiert werden. Hier ist es offenbar weniger anstrengend den Blick etwas abseits vom Zentrum zu halten, als den Kopf zu drehen.
Zusammengefasst ergibt sich also folgendes abstraktes Modell:

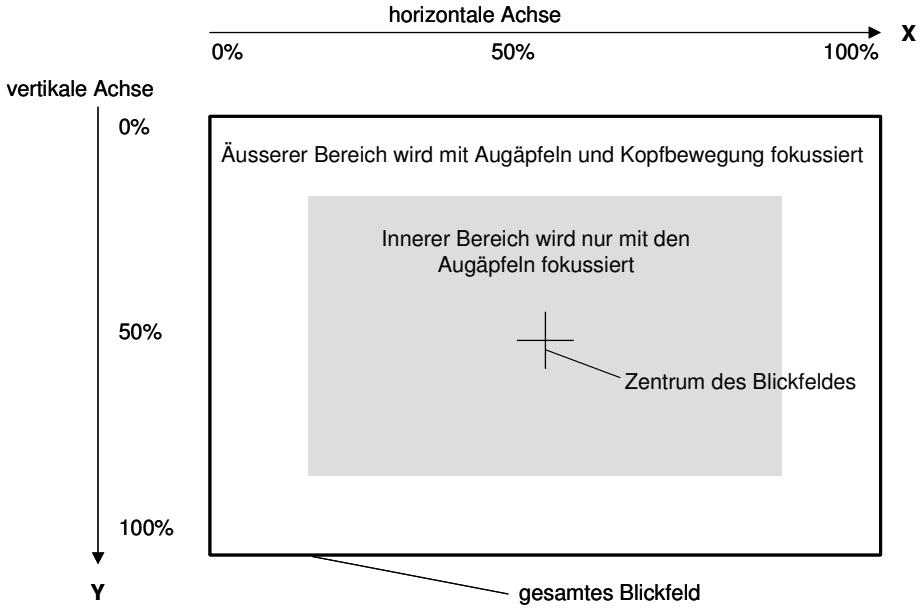


Abb. 36: Abstraktes Modell des menschlichen Blickfeldes

Man beachte, dass es keinen Bereich gibt, in dem die Augäpfel nicht bewegt werden. Da wir, im Gegensatz zu CCD-Kameras, nur im Zentrum scharf sehen können wird selbst bei minimalen Abweichungen das Auge bewegt. Es ist uns beispielsweise unmöglich ohne Bewegung des Augapfels ein Wort zu fokussieren und währenddessen ein Wort zwei Zeilen darunter zu erkennen.

Umsetzung der Gesichtsverfolgung
Um nun die Gesichtsverfolgung in einer Threadfunktion in RobotControl zu realisieren wurden die Funktionen aus der Gesichtserkennungs-DLL importiert. Benötigt werden die Anzahl der gefundenen Gesichter, sowie die relative XY-Position des größten Gesichtes im Videobild.

Bedingt durch die Reibung der Augenlider an der Gummihaut kann es vorkommen, dass ein Auge nach dem Blinzeln geschlossen bleibt.
Wenn dieses Auge das aktive ist, d.h. von diesem Auge wird das Kamerasignal ausgewertet, so kann die Gesichtserkennung natürlich kein Gesicht finden, selbst wenn eines da wäre.
Darum wird in der Threadfunktion *visualAdjustmentThreadISR* die Variable *numberOfFramesWithoutFaces*, für jedes Frame ohne gefundene Gesichter, um eins erhöht.
Erreicht die Variable einen Wert größer oder gleich dem Wert *maxNumberOfFramesWithoutFaces*, so werden beide Augenlider zwangsweise und unabhängig vom Blinzeln geöffnet.

Dieses Verhalten bewirkt, dass das ein eventuelles Hängenbleiben der Augenlider umgehend behoben wird.

Im normalen Betrieb stört es nicht, da *maxNumberOfFramesWithoutFaces* hoch genug gewählt ist, um beim Blinzeln nicht erreicht zu werden.

Und für den Fall, dass längere Zeit kein Gesicht gefunden wurde, weil keines da ist, bewirkt diese Vorgehensweise, dass der Roboter nicht mehr blinzelt. Was nicht nachteilig ist, da ihm somit auch kein Gesicht wegen kurzzeitigem Augenschließen entgehen kann.

Für den Fall aber, dass ein Gesicht gefunden wurde, muss nun untersucht werden, ob es sich im Zentrum des Blickfeldes befindet. Gegebenenfalls muss der Roboter dann seine Augen und seinen Kopf so drehen, dass es wieder fokussiert ist.

Um dies zu realisieren wurde das Modell aus Abbildung 36 an den Roboter angepasst und in horizontale und vertikale Ausrichtung unterteilt.

Horizontales Tracking

Abbildung 37 zeigt das Blickfeldmodell des Roboters für die horizontale Ausrichtung. Da Augen und Kopf drehbar sind, hat der Roboter einen sehr großen Sehbereich. Es kann jedoch immer nur ein Teil davon überblickt werden, was durch den grauen Bereich symbolisiert wird.
In dieser Abbildung hat das Servo für die Drehung des Auges 50% seines Servoweges eingenommen und der Roboter sieht geradeaus.
Innerhalb des gesehenen Bildes (grau) markiert die 50%-Stelle den Punkt im Bild, den der Roboter momentan fokussiert – also das Zentrum des Videobildes.

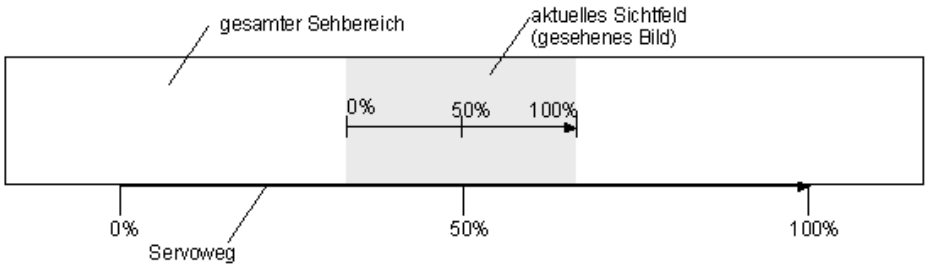


Abb. 37: Blickfeldmodell des Roboters.

Wichtigster Unterschied im Vergleich mit dem menschlichen Blickfeld ist dabei, dass der Roboter einen Bereich im Zentrum des Blickfeldes kennt, in dem er nicht auf eine Abweichung des Gesichtes reagiert (Abbildung 38).

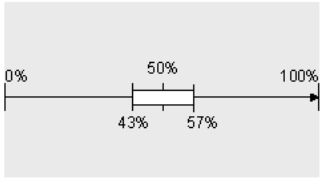


Abb. 38: Toleranzbereich

Der Grund für diesen Toleranzbereich ist, dass der Roboter die Kameras nicht pixelgenau positionieren kann.
Das heißt, ohne diesen Toleranzbereich hätte man einen endlos schwingenden Regelkreis und ein nervös mit den Augen zuckenden Roboter.

Was aber, wenn das Gesicht nun außerhalb des Toleranzbereiches liegt?
Abbildung 39 zeigt solch einen Fall.

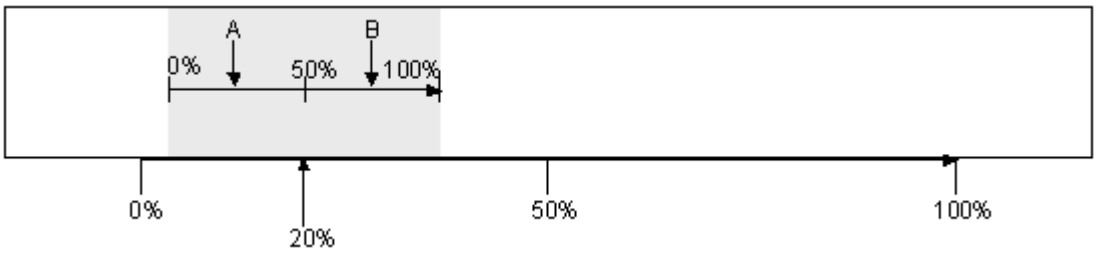


Abb. 39: Beispielszenario

In diesem Beispiel hat das Servo für die Augendrehung 20% des Servoweges eingenommen.
An Punkt A wird nun ein Gesicht im Display gefunden.
Die relative X-Position von A im Bild sei bei 25%.

In der Threadfunktion wird nun die Differenz zwischen Punkt A und dem Mittelpunkt des Bildes (50%) betrachtet.
In diesem Fall ist sie bei 25% und somit höher als die Toleranzschwelle von ±7% (vgl. Abb. 38).
Ebenso sähe das Ergebnis aus, für den Fall, dass das Gesicht in B bei 75% gefunden worden wäre.

Abhängig davon, ob sich der Punkt links oder rechts vom Bildmittelpunkt befindet wird das Auge nun ein kleines Stück nach links bzw. rechts gedreht.
Im Fall A also nach links, im Fall B nach rechts.
Dabei wird die Variable *numberOfFramesSinceLastEyeMove* auf 0 gesetzt.

Spätestens nach ein paar Schleifendurchläufen haben sich somit die Augen auf das Gesicht ausgerichtet und müssen sich nicht mehr bewegen, da das Gesicht nun innerhalb des Toleranzbereiches ist.
Nun wird bei jedem Frame die Variable *numberOfFramesSinceLastEyeMove* um eins erhöht.

Nach fünf Frames wird dann der Kopf nachbewegt.
Dazu wird ermittelt, ob die Augen nach links oder rechts bewegt wurden.
Das Servo für die Kopfdrehung wird dann schrittweise in die entsprechende Richtung bewegt.
Durch die Kopfdrehung werden natürlich aber auch wieder die Augen mitbewegt, so dass das Gesicht eventuell wieder den Toleranzbereich verlässt und die Augäpfel wieder zuerst ausgerichtet werden.

Abbildung 40 zeigt die Schritte des horizontalen Gesichtstracking an einem Beispiel:

Abb. 40: Phasen der Gesichtsverfolgung.

Von links nach rechts:

1. Das Gesicht ist fokussiert und innerhalb des Toleranzbereiches.
2. Das Gesicht ist weiter nach rechts gegangen, die Augen werden nach rechts gedreht, bis das Gesicht wieder im Zentrum ist.
3. Der Kopf wird nachgedreht. Wenn das Gesicht dabei den Toleranzbereich verlässt, werden die Augen neu justiert.
4. Das Gesicht ist wieder an seine erste Position gegangen. Die Augen werden wieder nach links gedreht bis das Gesicht wieder im Blickzentrum ist.
5. Der Kopf folgt der Augenbewegung nach links. Wenn das Gesicht dabei den Toleranzbereich verlässt, werden die Augen neu justiert.

Was man bezüglich des horizontalen Trackings noch wissen sollte ist, dass immer nur ein Kamerabild ausgewertet wird und die Ausrichtung somit auch nur für das eine entsprechende Auge korrekt ist.
Das andere Auge macht einfach die Bewegungen mit.
Schielen wird der Roboter daher nie – was aber bei normalem Abstand zwischen Roboter und Benutzer auch nicht auffällt.

Vertikales Tracking
Zusätzlich zum horizontalen Tracking ist natürlich auch das vertikale Tracking nötig, welches die Ausrichtung entlang der Y-Achse steuert.

Aufgrund der begrenzten Platzverhältnisse im Roboterkopf konnte keine Vorrichtung konstruiert werden, die es erlaubt die Augen nach oben oder unten zu drehen.
Die vertikale Blicksteuerung erfolgt daher allein durch Nickbewegungen des gesamten Kopfes.

Dieser Umstand vereinfacht allerdings auch die softwaretechnische Umsetzung.

So wird in *visualAdjustmentThreadISR* nach dem gleichen Prinzip wie beim horizontalen Tracking auch die vertikale Fokussierung gesteuert.
Allerdings mit dem Unterschied, dass nicht zuerst die Fokussierung der Augen abgewartet wird.

Zudem ist der Toleranzbereich mit 30% etwa doppelt so groß dimensioniert.
Würde dieser Wert verkleinert, würde der Regelkreis beginnen zu schwingen und der Roboter würde endlos mit dem Kopf nicken.

Hauptprogramm und grafische Benutzeroberfläche

Das Hauptprogramm steuert über Spracheingabe und Sprachausgabe die verbale Kommunikation mit dem Benutzer.
Diesem steht eine grafischen Oberfläche mit vier Tabs zur Verfügung.

Die Benutzeroberfläche

Hier kann er eine Geschichte öffnen und vorlesen lassen, Einstellungen zur Sprachausgabe und zur Robotersteuerung vornehmen und sich die Ergebnisse der Gesichtserkennung anzeigen lassen.

Tab *Geschichte*

Im ersten Tab „Geschichte“ (Abbildung 41) befindet sich eine Textbox, welche die geladene Geschichte anzeigt und zum Mitlesen das gerade gesprochene Wort hervorhebt. Ein Pegelmeter unten rechts zeigt die Lautstärke der Stimme an.
Über intuitiv bedienbare Buttons, wie man sie von Audiogeräten kennt, lässt sich die Geschichte abspielen, unterbrechen oder beenden.

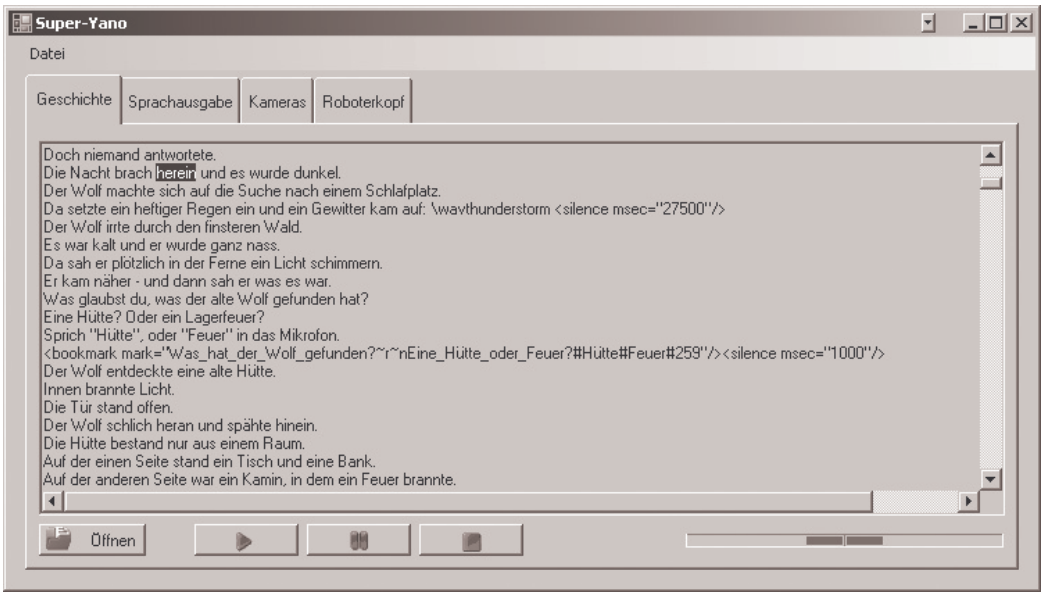


Abb. 41: Tab „Geschichte“ zum Vorlesen lassen eines Textes.

Tab *Sprachausgabe*

Im Tab „Sprachausgabe“ kann die zu verwendende Stimme ausgewählt und konfiguriert werden (Abb. 42).
Vorlesegeschwindigkeit und Lautstärke sind variierbar.
Im rechten Bereich werden Informationen, wie Name, Geschlecht oder Alter der ausgewählten Stimme angezeigt.
Mit dem Button Test, kann eine Vorschau der Stimme angehört werden.

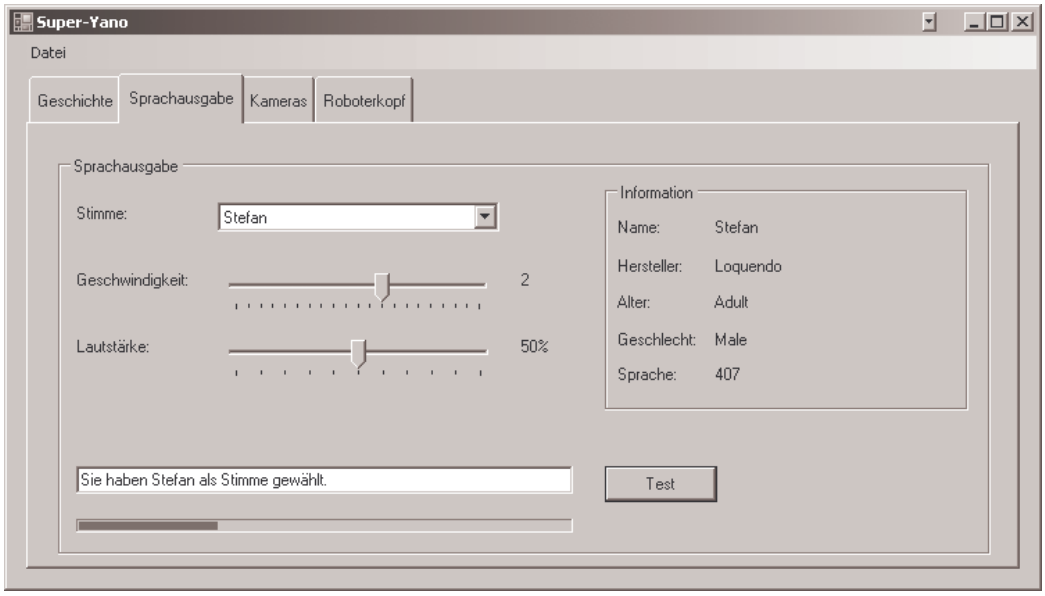


Abb. 42: Tab „Sprachausgabe“ zum Einstellen der Stimmoptionen.

Tab *Kameras*

Im Tab „Kameras“ (Abb. 43) kann festgelegt werden, ob die Gesichtsuche aktiviert sein soll und welches Auge dazu verwendet werden soll.
Im linken Bereich wird dann das ausgewertete Videobild abgebildet, während rechts Informationen über die Anzahl der gefundenen Gesichter, der Videoauflösung und der Berechnungszeit angezeigt werden.

Abb. 43: Tab „Kameras“ zur Auswahl des aktiven Auges.
Hier betrachtet der Roboter gerade sein Spiegelbild.

Tab *Roboterkopf*

Im Tab „Roboterkopf“ (Abb. 44) kann festgelegt werden, welche Verhaltensweisen der Roboter simulieren soll.
So lassen sich Mundbewegungen beim Sprechen, das Blinzeln mit den Augenlidern, das Zucken der Ohren und die Ausrichtung nach Gesichtern aktivieren, bzw. deaktivieren.
Mit der Checkbox „Servos aktiviert“ kann die Stromversorgung der Servos ein- und ausgeschaltet werden.
Der Button „Servos initialisieren“ dient dazu alle Servos in ihre Ruhestellung zu fahren.
Im unteren Bereich dieses Tabs wird der verwendete Comportbezeichner angegeben, sowie die Verbindungseinstellungen.

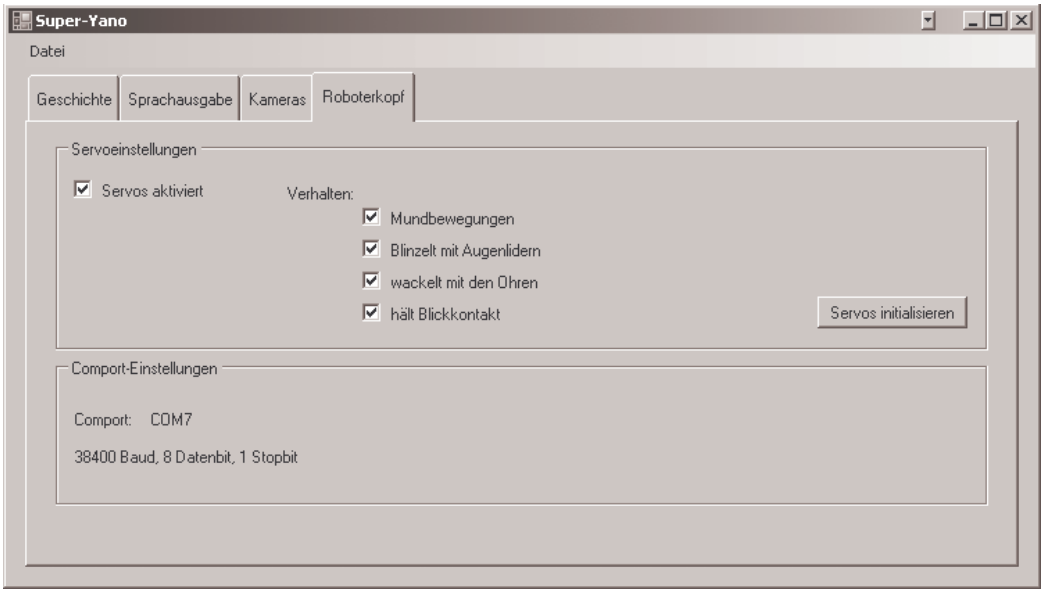


Abb. 44: Tab „Roboterkopf“ mit Optionen zum Roboterverhalten und zur Anzeige der Comport-Einstellungen.

Visuelle Wahrnehmung

Das Hauptprogramm *Form1* besitzt eine Thread *facedetectThread* und einen Timer *visualTimer*.
Mit diesen nimmt es Einfluss auf die visuelle Wahrnehmung des Roboters.
Um die Abhängigkeiten zwischen dem Hauptprogramm, der Gesichtserkennung und RobotControl zu verdeutlichen, sei dies exemplarisch an einem Modell erklärt:

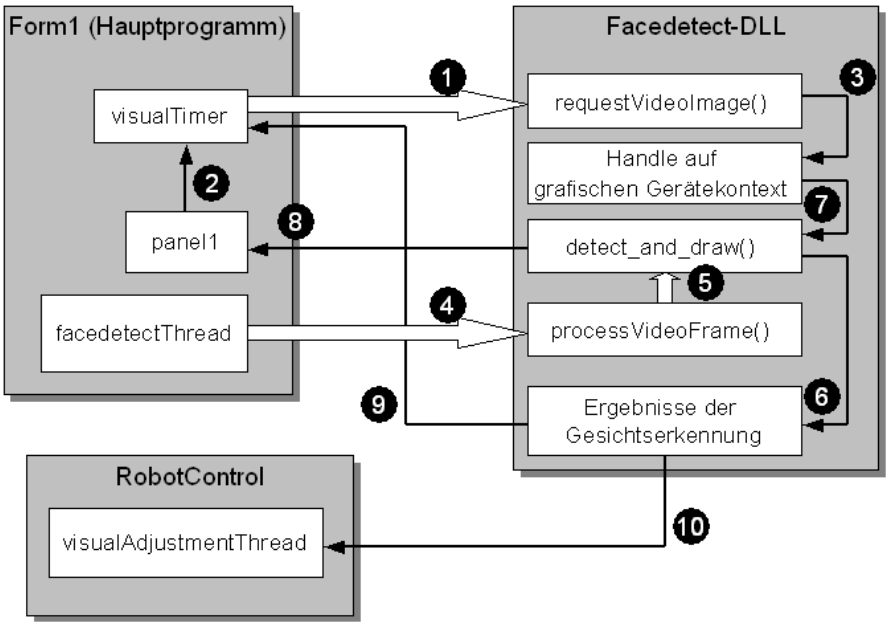


Abb. 45: Modell der visuellen Wahrnehmung

1. Die Timerfunktion des *visualTimer* aus *Form1* ruft alle 100ms die aus der DLL importierte Funktion *requestVideoImage* auf
2. und übergibt ihr dabei ein Handle auf *panel1*, ein Panel im Tab „Kameras“.
3. *requestVideoImage* speichert das Handle in einer globalen Variable der Facedetect-DLL.
4. Der Thread *facedetectThread* aus *Form1* ruft alle 100ms die Funktion *processVideoFrame* aus der DLL auf,
5. die ihrerseits die Gesichtserkennung in *detect_and_draw* aufruft.
6. Die Ergebnisse der Gesichtserkennung (Anzahl gefundener Gesichter und Position des größten Gesichts im Videobild) werden in globalen Variablen in der Facedetect-DLL abgelegt.
7. Wenn ein Bild des Videostreams angefordert wurde, also wenn ein Handle hinterlegt wurde,
8. wird *detect_and_draw* um gefundene Gesichter einen Rahmen zeichnen und das Bild auf die Zeichenfläche des Handle zu kopieren.
9. *visualTimer* ruft die Ergebnisse ab und gibt sie im Tab „Kameras“ neben *panel1* an.
10. Auch der Thread *visualAdjustmentThread* aus RobotControl ruft diese Ergebnisse ab, um den Roboter auf ein gefundenes Gesicht auszurichten.

Wenn im Hauptprogramm im Tab „Kameras“ die Checkbox „Gesichtsuche aktiviert“ deselektiert wird, so wird zugleich im Tab „Roboterkopf“ die Checkbox „hält Blickkontakt“ deselektiert und der *visualTimer* angehalten.
Denn wenn *processVideoFrame* nicht mehr aufgerufen wird, also die Gesichtserkennung deaktiv ist, braucht RobotControl auch nicht mehr den Kopf auf ein Gesicht auszurichten und der Timer keine aktuellen Videobilder anfordern.

Umgekehrt, wird beim Auswählen der Checkbox „hält Blickkontakt“ auch die Option „Gesichtsuche aktiviert“ um das kontinuierliche Aufrufen von *processVideoFrame* durch den Thread *facedetectThread* zu erzwingen.

Sprachausgabe

Das Speech Application Programming Interface von Microsoft - kurz SAPI genannt - ist eine Standardschnittstelle für Sprachausgabe, die in immer mehr Applikationen angewendet wird. Unter [15] findet man die SAPI-Homepage.

Ihr größter Vorteil im Vergleich zu anderen Sprachausgabeprodukten ist die große Verbreitung und die Erweiterbarkeit mit, meist kostenlosen, Paketen von Drittanbietern zur Installation weiterer Stimmen und Sprachen.
So wird in diesem Projekt ein Sprachpaket zur Sprachausgabe in Deutsch von Loquendo [16] verwendet.

Um nach der Installation von SAPI die Sprachausgabe in eigene Programme einzubinden gibt es mehrere Möglichkeiten.
Auf meiner Website [17] sind zwei Varianten für Delphi beschrieben.

Um die SAPI in diesem Projekt mit Visual Studio zu nutzen wurde im Hauptprogramm *Form1* der Namespace *SpeechLib* eingebunden und eine Variable *voice* vom Typ *SpVoice* aus diesem Namensraum angelegt.
In der Funktion *Form1_Load* wird *voice* dann bei Programmstart initialisiert und ist bereit für die Sprachausgabe.

Zuvor soll jedoch die zuletzt verwendete Stimme eingestellt werden.
Jede installierte Stimme hat einen Namen – z.B. „Stefan“.
Nun wird von der SpeechLib eine Liste aller verfügbaren Stimmen abgefragt und diese Liste nach dem Namen der zuletzt verwendeten Stimme durchsucht, weil der Index der Stimme benötigt wird.
Wird die Stimme nicht gefunden (es könnte ja sein, dass sie seit dem letzten Programmstart deinstalliert wurde) wird der Index auf 0 gesetzt.

Nachdem der Index der Stimme so ermittelt und das voice-Objekt entsprechend konfiguriert wurde, werden im Tab „Sprachausgabe“ Informationen zu dieser Stimme (Name, Alter, Geschlecht) angezeigt.

Features

Zum Sprechen eines Textes wird dem voice-Objekt der vorzulesende Text als Parameter der Methode *Speak* übergeben.
Es wird den Text dann vorlesen und zugleich auf Schlüsselwörter untersuchen. Solche Schlüsselwörter, folgend Tags genannt, sind im XML-Stil gehalten.
Sie wirken sich in Echtzeit auf das vorlesende Voice-Objekt aus und können somit genutzt werden um direkt aus dem Text Einfluss auf die Stimme zu nehmen.

Folgende Features werden in SAPI 5.1 und von der hier verwendeten Stimme unterstützt:

- Tonhöhe:

  ```
  <pitch absmiddle="10">
        Lässt den diesen Text mit hoher Stimmer vorlesen.
  </pitch>

  <pitch absmiddle="-10">
        Lässt diesen Text mit tiefer Stimmer vorlesen.
  </pitch>
  ```

  Es sind alle diskreten Werte zwischen –10 und +10 möglich.

- Lautstärke:

  ```
  <volume level="-10">
        Lässt diesen Text ganz leise vorlesen.
  </volume>

  <volume level="10">
        Lässt diesen Text ganz laut vorlesen.
  </volume>
  ```

  Auch hier sind alle Werte von –10 bis +10 möglich.

- Geschwindigkeit:

  ```
  <rate absspeed="-10">
        Lässt diesen Text ganz langsam vorlesen.
  </rate>

  <rate absspeed="10">
        Lässt diesen Text ganz schnell vorlesen.
  </rate>
  ```

  Hier sind ebenfalls Werte von –10 bis +10 möglich.

- Kombinationen:

```
<rate absspeed="-10">
     <pitch absmiddle="10">
          Auch Kombinationen aus den Tags
     </pitch>
</rate>

<rate absspeed="10">
     <pitch absmiddle="-10">
          sind möglich.
     </pitch>
</rate>
```

- Mit dem spell-Tag können Wörter buchstabiert werden:

```
<spell>
     Dieser Text wird buchstabiert.
</spell>
```

- Es existiert ein bookmark-Attribut, dass zum Stellen von Fragen an den Benutzer zweckentfremdet wurde (mehr dazu später):

```
<bookmark mark="bookmark_one"/>
```

- Das Vorlesen kann durch Pausen unterbrochen werden:

```
<silence msec="1500"/>
```
In diesem Beispiel für 1500ms.

- Und die Sprache bzw. Stimme kann durch Tags geändert werden:

```
<voice required="Language=809">
     Hello and welcome! This is an english speaking voice!
</voice>

<voice required="Language=c0c">
     Bon jour! Cela est une voix française !
</voice>

<voice required="Language=40A">
     ¡Hola! ¡Esto es una voz española!
</voice>
```

Um sich diese Features anzuhören, kann die Datei features.xml mit Super-Yano geladen und abgespielt werden.

Modell der Sprachausgabe

Für dieses Projekt war es aber erforderlich nicht nur einfach Texte vorlesen zu lassen, sondern dabei währenddessen auch auf Ereignisse bezüglich der Sprachausgabe zu reagieren.
Ein spVoice-Objekt kennt dazu eine Reihe von Events, die beim Sprechen auftreten können.
In *Form1_Load* werden einige dieser Ereignisse von *voice* mit Funktionen aus *Form1* verknüpft.

Bevor diese Funktionen und ihre auslösenden Events näher beschrieben werden, soll Abbildung 46 das Modell der Sprachausgabe dieses Projektes illustrieren:

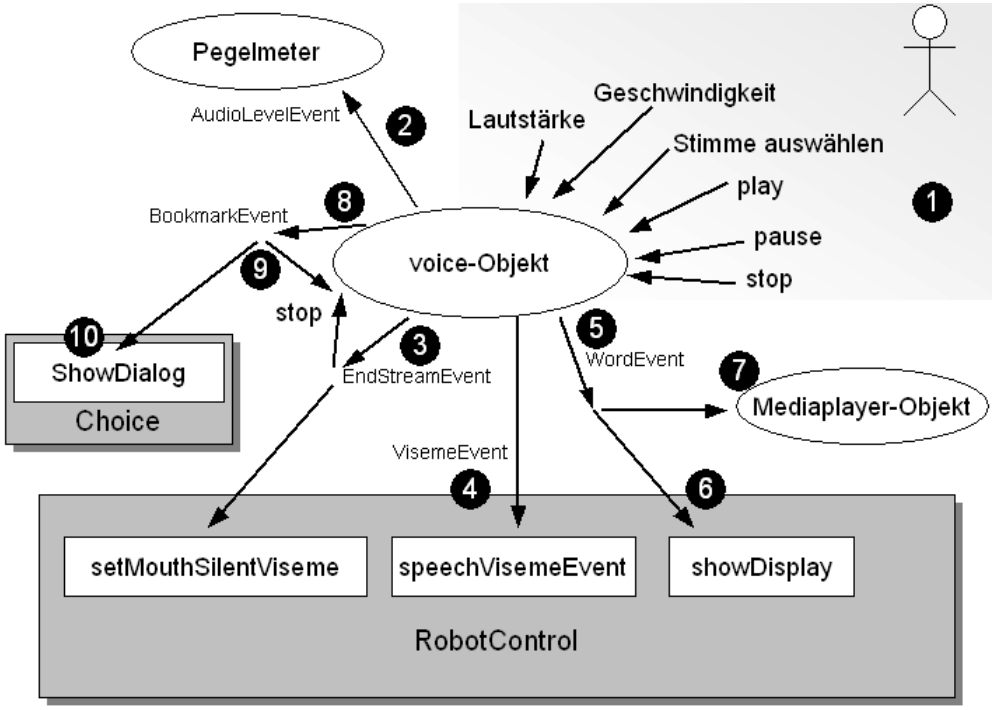


Abb. 46: Modell der Sprachausgabe in Super-Yano

Im Mittelpunkt steht dabei natürlich das voice-Objekt.
Auf dieses wirken eine Reihe benutzergesteuerter Aktionen (1).
So sind über die grafische Oberfläche die Lautstärke und Geschwindigkeit einstellbar, die Stimme ist variierbar und das Sprechen kann gestoppt, pausiert oder gestartet werden.

Sprachereignisse

Dem gegenüber stehen mehrere Ereignisse, die beim Sprechen auftreten können und sich auf den Programmablauf und das Verhalten des Roboters signifikant auswirken.

AudioLevelEvent (2)

Dieses Ereignis tritt während des Sprechens bei Lautstärkeschwankungen der Audioausgabe auf, was man leicht für einen Audio-Pegelmeter nutzen kann.
Dazu wurde dieses Event mit der Funktion *speechAudioEvent* verknüpft.
Diese Funktion macht nun nichts anderes als den Wert der aktuellen Lautstärke auf Balkengraphen in den Tabs „Sprachausgabe“ und „Geschichte“ anzuzeigen (Abb. 47).

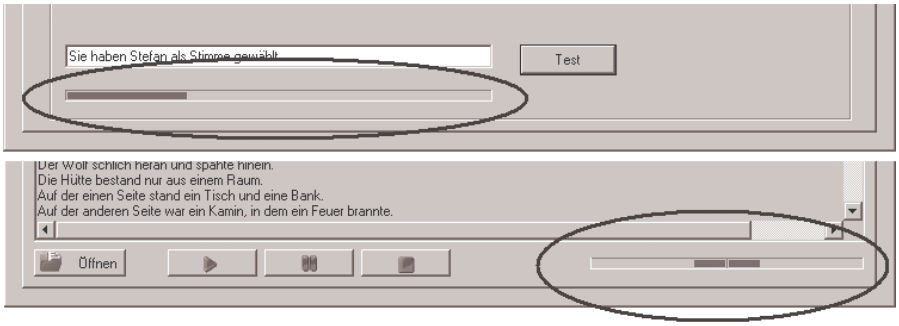


Abb. 47: Audiopegelmeter in den Tabs „Geschichte“ und „Sprachausgabe“

EndStreamEvent (3)

Dieses Ereignis ist mit der Funktion *speechEndStreamEvent* verknüpft.
Die Funktion ruft die OnClick-Ereignisfunktion des Stopbuttons auf.
Die Wirkung ist also identisch mit einem Klick auf Stop.
In diesem Fall wird der Rest des zu lesenden Textes übersprungen und der Roboter mit der Funktion *setMouthSilentViseme* aus RobotControl angewiesen die Servos für Mundwinkel und Kiefer wieder in Ruhestellung zu fahren.
Abbildung 48 zeigt den vollständigen Aufrufverlauf für *speechEndStreamEvent*.

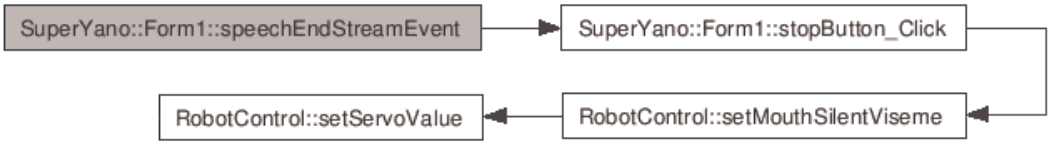


Abb. 48: Callgraph für *speechEndStreamEvent*

VisemeEvent (4)

Viseme (Phoneme) sind kleinste Bausteine der Sprache, die angeben wie der Mund für bestimmte Töne geformt wird.
Dieses, beim Sprechen ausgelöste, Event ist mit der bereits beschriebenen Funktion *speechVisemeEvent* aus RobotControl verknüpft und veranlasst den Roboter synchrone Mundbewegungen zur Sprachausgabe auszuführen.
Eine Reaktion auf dieses Ereignis innerhalb von Form1 findet nicht statt.

***WordEvent* (5)**
Dieses Event wird ausgelöst, sobald ein Wort gesprochen wird und ist mit der Funktion *speechWordEvent* verknüpft.
Als Übergabeparameter des Events erhält *speechWordEvent* nur die Position des Wortes im Text und die Länge des Wortes.
Es muss also zuerst anhand dieser Angaben ein Substring aus dem Text kopiert werden um das Wort als String zu erhalten.

Nun werden drei Fälle unterschieden:

Wenn das Wort mit einem „<“-Zeichen anfängt, so handelt es sich um ein XML-Tag, das nicht im Textfeld im Tab „Geschichte“ hervorgehoben werden soll.

Wenn das Wort mit „\wav“ beginnt, beispielsweise „\wavsound1“, so wird eine Sounddatei abgespielt. Dazu wird „\wav“ vom String abgeschnitten um den Wavedateinamen zu erhalten.
Nun wird ein Mediaplayerobjekt erstellt, diesem der Dateiname übergeben und der Sound abgespielt (7).
Alle Wavedateien müssen sich dazu im Unterverzeichnis „sounds“ befinden.

Wenn beide oben genannten Fälle nicht zutreffen, so handelt es sich um ein normales Wort.
In diesem Fall wird das Wort in der Textbox durch Markieren hervorgehoben, um das Mitlesen zu erleichtern.
Da sich der Benutzer aber in der Regel direkt vor dem Roboter befindet und Blickkontakt mit diesem hält, kann er nicht gleichzeitig in der grafischen Oberfläche den Text mit verfolgen.
Wenn der Benutzer nun aufgrund der qualitativen Unzulänglichkeiten der Sprachausgabe ein oder mehrere Wörter nicht versteht, soll er den Text auf dem Display direkt unter dem Roboter mitlesen können.
Hierzu wird das Wort an die Funktion *showDisplay* in RobotControl übergeben (6).
Allerdings muss es dazu bereits formatiert sein, d.h. ihm wird ein „!>“ vorangestellt und ein Leerzeichen angehängt. Ausrufezeichen werden durch 0xFF ersetzt.
Denn das Mikrocontrollerprogramm wertet solche als Beginn eines Befehls und löscht den Eingangsbuffer der seriellen Verbindung.

Aus dem Wort „Auto!“ wird somit:

!	>	A	u	t	o	0xFF	Leerzeichen

***BookmarkEvent* (8)**
Dieses Event wird ausgelöst, sobald die Sprachengine beim Sprechen auf ein Bokkmark-Tag stößt.
Solch ein Tag sieht beispielsweise so aus:

```
<bookmark mark="bmk1"/>
```

Die Funktion *speechBookmarkEvent* ist hiermit verknüpft.
Als Parameter erhält sie die den Text der Bookmark als String – bei obigen Beispiel also „bmk1“.

Super-Yano agiert als Geschichtenerzähler. Um nun Interaktivität herzustellen, war vorgesehen dem Benutzter Fragen zu stellen, deren Antwort den Verlauf der Geschichte beeinflussen.
Hierfür ließ sich das Bookmark-Feature für eigene Zwecke nutzen.

Benötigt werden ein Bookmark um das Vorlesen zu beenden und eines um Fragen zu stellen. Zusammen mit normalen Bookmarks gibt es so drei Fälle die beim Bookmark-Event eintreten können:

Wenn es sich um ein normales Bookmark wie nach obigem Beispiel handelt, wird *speechBookmarkEvent* es schlicht ignorieren.

Wenn das Bookmark den Text „END“ enthält, also die Form `<bookmark mark="END"/>` hat, so wird die OnClick-Ereignismethode des Stopbuttons *stopButton_Click* aufgerufen (9), welche die Sprachausgabe anhält und den Roboter veranlasst den Mund in Ruhestellung zu bringen.

Für den dritten Fall, das Stellen einer Frage an den Benutzer, muss das Bookmark aber vier Informationen kapseln:

- die Frage
- mögliche Antwort A
- mögliche Antwort B
- Anzahl der Sätze, die bei B übersprungen werden

Da ein Bookmark nur einen String als Parameter haben kann, wurde eine Lösung entwickelt, bei der diese Informationen mit #-Zeichen getrennt werden. Im vorzulesenden Text hat ein so formatiertes Bookmark somit beispielsweise die Form
`<bookmark mark="Was_hat_der_Wolf_gefunden?~r~nEine_Hütte_oder_Feuer?#Hütte#Feuer#259"/>`

Ein Bookmarkstring darf keine Leerzeichen enthalten, da dies die Sprachengine nicht parsen kann. Darum werden sie durch Unterstriche ersetzt.
Ebenso werden Zeilenumbrüche mit „~r~n“ substituiert.

Über den Parameter *bookmark* erhält die Funktion *speechBookmarkEvent* nun den String `="Was_hat_der_Wolf_gefunden?~r~nEine_Hütte_oder_Feuer?#Hütte#Feuer#259"`.
Durch das Splitten des Strings mit dem Trennzeichen # erhält man dann alle vier Informationen.

Jetzt kann eine Frage an den Benutzer gestellt werden. Diese soll er über Spracheingabe oder durch Anklicken eines Buttons in einem Dialogfenster (10) beantworten.

Spracheingabe

Als zusätzliches Feature wurde eine Spracheingabemöglichkeit implementiert. Hierzu werden ActiveX-Komponenten der Spracherkennungssoftware *Naturally Speaking* [18] von *DragonSoft* verwendet.
Nach der Installation von Naturally Speaking können diese, wie in Abbildung 49 gezeigt, in Visual Studio importiert und dann einfach zu bestehenden Formularen per Drag&Drop hinzugefügt werden.

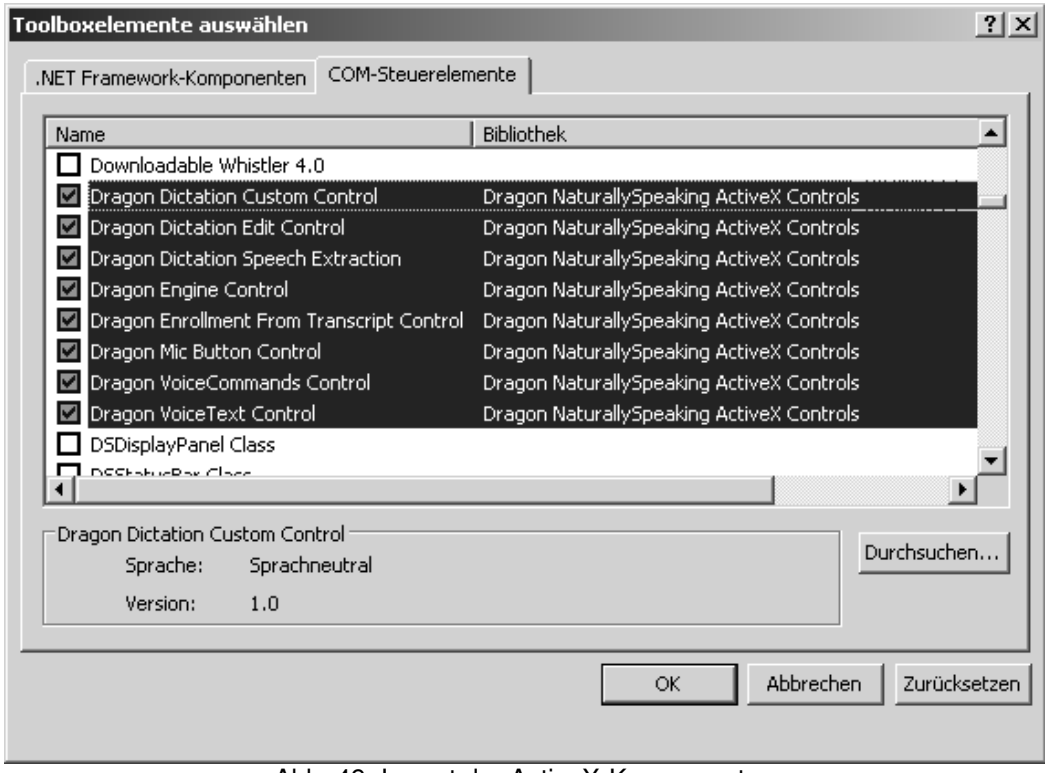


Abb. 49: Import der ActiveX-Komponenten

Um den Benutzer nun eine Frage zu stellen und die Antwort per Mikrofon zu erfassen, wird nicht viel benötigt.

Es bedarf eines Dialogformulars, zwei Buttons für Antwort A und B, sowie eine *AxDgnMicBtn*-Komponente aus den Naturally Speaking ActiveX-Komponenten.

Dialogformular *Choice*

Es wurde also ein Dialogformular Choice (engl. Auswahlmöglichkeit) erstellt, welches neben diesen Komponenten auch ein Textfeld zur textuellen Ausgabe der Frage aufweist (Abbildung 50).

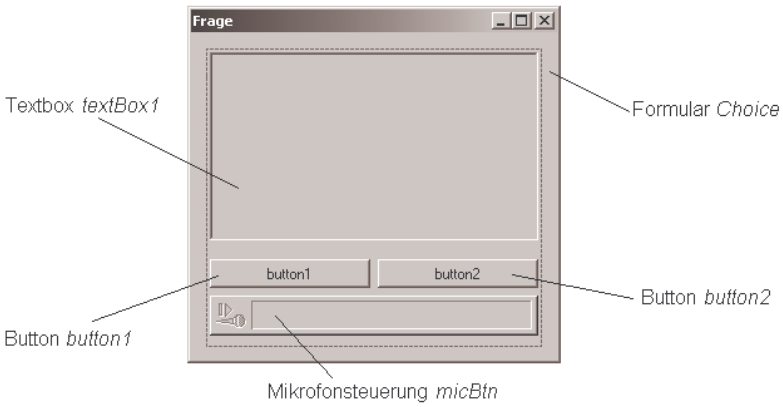


Abb. 50: Dialogformular Choice zur Entwurfszeit

Der Konstruktor dieser Formularklasse erwartet dabei drei Parameter:

```
Choice(String^ text, String^ btn1Text, String^ btn2Text)
{
	InitializeComponent();
	this->textBox1->Text = text;
	this->button1->Text = btn1Text;
	this->button2->Text = btn2Text;
}
```

Der Parameter *text* dient der schriftlichen Wiederholung der gesprochenen Frage und wird dem Textfeld *textBox1* zugewiesen.
Die Parameter *btn1Text* und *btn2Text* sind die möglichen Antworten auf die Frage und werden als Beschriftung der Buttons benutzt.

Um nun die Spracherkennung zu aktivieren bedarf es einzig nur zwei Zeilen Code in der Show-Methode des Formulars.
Da keine Dokumentation vorlag, kostete es allerdings trotzdem viel Zeit, Nerven und zahlreiche Versuche die Spracherkennungsengine zu initialisieren.
Erst mit Informationen aus [19] und [20] gelang die Realisierung, die hier nicht vorenthalten werden soll:

```
private: System::Void Choice_Shown(System::Object^  sender, System::EventArgs^  e)
{
 try
 {
  micbtn->Register(DNSTools::DgnRegisterConstants::dgnregGlobalCM);
  micbtn->MicState = DNSTools::DgnMicStateConstants::dgnmicOn;
 }
 catch(Exception^)
 {
 }
}
```

Ablauf

Was geschieht also?
Zuerst wird die Registrierungsfunktion von *micbtn* aufgerufen, damit die Spracherkennungsengine den Mikrofonbutton als Quelle annimmt und Events an ihn versendet.
Und als zweites wird das Mikrofon eingeschaltet.

Mit diesen beiden Schritten ist alles Erforderliche getan, um per Sprachbefehl auf eine Frage antworten zu können – weiterer Code wird nicht benötig.

Das Rätsel um diesen Minimalismus beruht auf einem simplen Trick:
Dialogformulare, wie das Choiceformulare, haben ein Attribut *DialogResult*, welches quasi als Rückgabewert betrachtet werden kann.

Nun wurde zur Entwurfszeit das Attribut *DialogResult* von *button1* auf YES gesetzt, das von *button2* auf CANCEL.
Das bedeutet, dass das Dialogfenster bei Klick auf einen der Buttons geschlossen wird und das Fenster dann einen unterschiedlichen Rückgabewert (YES oder CANCEL) liefert.

Somit kann der Benutzer durch Klicken auf einen der Buttons unterschiedliche Antworten wählen.

Weiterhin befindet sich der Mikrofonbutton standardmäßig in einem Befehlsmodus. Das heißt, dass er Wörter, welche die Spracherkennung aufnimmt, versucht Steuerelementen zuzuordnen.
Da die Buttons *button1* und *button2* die Aufschrift der möglichen Antworten tragen, reicht es, eine dieser Antworten in das Mikrofon zu sprechen und *micbtn* wird einen Klick auf den entsprechenden Button simulieren (allerdings beschränkt sich der Mikrofonbutton leider nicht auf die Anwendung, in der er sich befindet, sondern kann z.B. auch mit „Start“, „Beenden“, „Ausschalten“ genutzt werden um den Rechner herunterzufahren!).

Geschichten mit mehreren Ausgängen

Wie wirken nun das Bookmark-Tag und das Choice-Dialogformular mit der Spracherkennung zusammen? Und wie erreicht man eine Verzweigung der vorgelesenen Geschichte, so dass mehrere mögliche Ausgänge entstehen?

Die Beantwortung dieser Fragen lässt sich am besten anhand eines Beispiels erklären. Dazu dient folgende kurze Beispielgeschichte:

1	Wohin geht der Wolf?
2	`<bookmark mark="Wohin_geht_der_Wolf?~r~nIn_den_Wald_oder_in_die_Stadt?#Wald#Stadt#11"/>`
3	`<silence msec="1000"/>`
4	Der Wolf geht in den Wald!
5	Dort findet er einen hohlen Baumstamm.
6	Er greift hinein - und was glaubst du findet er?
7	Pilze oder Waldbeeren?
8	`<bookmark mark="Was_findet_der_Wolf?#Pilze#Waldbeeren#5"/>`
9	`<silence msec="1000"/>`
10	Der Wolf findet Pilze in dem hohlen Baumstamm.
11	Daraus macht er eine leckere Pilzsuppe.
12	`<bookmark mark="END"/>`
13	`<silence msec="500"/>`
14	ENDE!
15	Der Wolf findet ein paar Waldbeeren in dem hohlen Baumstamm.
16	Daraus macht er einen Früchtequark.
17	`<bookmark mark="END"/>`
18	`<silence msec="500"/>`
19	ENDE!
20	Er geht in die Stadt!
21	Dort geht er in ein Gasthaus und bestellt sich eine Portion Pommes.

Der Übersichtlichkeit halber ist die Geschichte als Tabelle mit Zeilennummerierung dargestellt.

Wenn der Benutzter die Geschichte geladen hat und auf den Playbutton klickt, so wird der gesamte Text an die *Speak*-Funktion des Voice-Objekts übergeben.

Die erste Zeile ist noch denkbar einfach zu verstehen: *voice* wird den Satz wortweise vorlesen.
Bereits in der zweiten Zeile stößt es jedoch auf eine Bookmark:

2	`<bookmark mark="Wohin_geht_der_Wolf?~r~nIn_den_Wald_oder_in_die_Stadt?#Wald#Stadt#11"/>`

Das Voice-Objekt wirft das BookmarkEvent, ohne den Text der Bookmark zu lesen und würde nun weiter im Text lesen.
Damit die mit dem BookmarkEvent gekoppelte Funktion Zeit hat, die Bookmark auszuwerten, folgt daher nun ein *silence*-Tag, dass die Sprachausgabe für 1000ms anhält.
Die Funktion *speechBookmarkEvent* wird nun also duch das Event aufgerufen und bekommt als Parameter den String
„Wohin_geht_der_Wolf?~r~nIn_den_Wald_oder_in_die_Stadt?#Wald#Stadt#11“ übergeben.
Sie wird erkennen, dass dies eine Frage ist und daraus den Text der Frage, die Antworttexte und die Anzahl zu überspringender Sätze ermitteln.

Zudem wird sie die *Pause*-Methode des Voice-Objekts aufrufen um zu verhindern, dass dieses nach Ablauf der 1000ms weiterliest, während noch die Frage gestellt wird.

Als nächstes wird *speechBookmarkEvent* ein Choice-Formular erstellen und dabei den Text der Frage und der möglichen Antworten übergeben.
Dann wird das Formular angezeigt (siehe Abbildung 51).

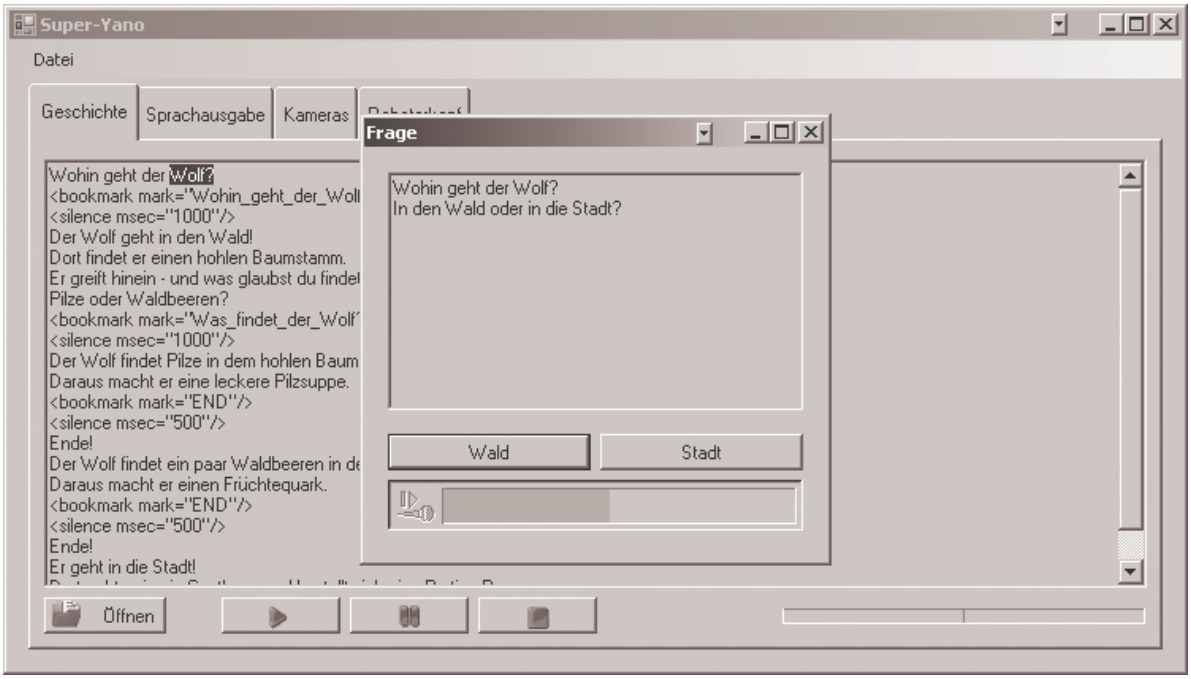


Abb. 51: Dem Benutzer wird eine Frage gestellt.

Der Benutzer muss nun eine der Antworten in das Mikrofon sprechen oder einen der beiden Buttons anklicken.
Daraufhin wird das Fragedialogfenster geschlossen und *speechBookmarkEvent* wird den DialogResult-Rückgabewert auswerten.
Wenn er YES ist, so wurde die erste Antwort ausgewählt („Wald“). Andernfalls wurde die zweite Antwort gewählt („Stadt“).

Nehmen wir in diesem Beispiel an, es sei die erste Antwort „Wald“ gewählt worden – DialogResult hätte also den Wert YES.
In diesem Fall geht die Geschichte in Zeile 4 weiter. Es ist also nichts weiter zu tun, als die *Resume*-Methode von *voice* aufzurufen, um mit dem Vorlesen in Zeile

4	Der Wolf geht in den Wald!

fortzufahren.

In Zeile 8 kommt tritt erneut ein Bookmark-Ereignis auf, das wie zuvor behandelt wird. Nehmen wir einmal an, es wird hier die zweite Antwort gewählt – DialogResult ist also ungleich YES.
Nun kann nicht einfach mit der Geschichte fortgesetzt werden, da ein Teil des Textes übersprungen werden muss.
Hierzu kennt die SAPI die *Skip*-Methode, der die Anzahl zu überspringender Elemente übergeben werden muss.
Als überspringbare Elemente werden derzeit nur Sätze unterstützt – Tags wie *bookmark* oder *silence* werden nicht mitgezählt.

Besonderheit beim Überspringen von Sätzen
Nun tritt eine Besonderheit beim Überspringen von Sätzen auf: es werden nur `n-1` Sätze übersprungen wenn *Skip* mit n aufgerufen wird und der erste übersprungene Satz ist der Satz, vor dem Fragen-Bookmark – im diesem Beispiel die Frage „Pilze oder Waldbeeren?“

speechBookmarkEvent ruft also mit dem letzten Wert aus dem Bookmarktext („5“) die *Skip*-Methode auf und es werden die Sätze

#	Zeile	Satz
1.	**7**	Pilze oder Waldbeeren?
2.	**10**	Der Wolf findet Pilze in dem hohlen Baumstamm.
3.	**11**	Daraus macht er eine leckere Pilzsuppe.
4.	**14**	ENDE!

übersprungen. Zur Erinnerung: Tags werden von *Skip* ohnehin ignoriert.

Die Geschichte geht somit in Zeile

15	Der Wolf findet ein paar Waldbeeren in dem hohlen Baumstamm.

weiter.

In Zeile 17 trifft *voice* erneut auf ein Bookmark-Tag:

17	`<bookmark mark="END"/>`

Hier handelt es sich dieses Mal nicht um eine Frage, sondern um eine Markierung, die ein Ende der Geschichte markiert.
In *speechBookmarkEvent* wird anhand des Bookmarktextes „END“ die Endmarkierung als solche erkannt und die *Stop*-Methode des Voice-Objekts aufgerufen, um zu verhindern, dass weiter im Text vorgelesen wird, nachdem ein Ende der Geschichte erreicht wurde.

Eine weitere Besonderheit
Wer die Eigenheiten beim Überspringen von Sätzen aufmerksam verfolgt hat, wird bemerkt haben, dass bei der zweiten Antwort („Stadt“) auf die erste Frage doch eigentlich 11 Sätze übersprungen werden müssten um zu

20	Er geht in die Stadt!

zu gelangen.

Es sind jedoch 10, da hier eine weitere Besonderheit auftritt:
Wenn im zu überspringenden Text Fragen-Bookmarks auftauchen, so betrachtet *voice* den vorangehenden und nachfolgenden Satz als ein zusammenhängendes Element.
Aus Sicht des Voice-Objektes würden die Sätze

#	Zeile	Satz
1.	**1**	Wohin geht der Wolf?
2.	**4**	Der Wolf geht in den Wald!
3.	**5**	Dort findet er einen hohlen Baumstamm.
4.	**6**	Er greift hinein - und was glaubst du findet er?
5.	**7 & 10**	Pilze oder Waldbeeren? Der Wolf findet Pilze in dem hohlen Baumstamm.
6.	**11**	Daraus macht er eine leckere Pilzsuppe.
7.	**14**	ENDE!
8.	**15**	Der Wolf findet ein paar Waldbeeren in dem hohlen Baumstamm.
9.	**16**	Daraus macht er einen Früchtequark.
10.	**19**	ENDE!

übersprungen werden, da Zeile 7 und 10 als ein Satz interpretiert werden.
Und somit es nur 10 Elemente sind.

Dieses merkwürdige Verhalten beruht sicherlich auf dem „Zweckentfremden“ des Bookmarktags bzw. einem Problem beim Skippen von Textteilen, die so formatierte Bookmarks enthalten – was genau die Ursache ist, konnte jedoch nicht zweifelsfrei ermittelt werden.

Vielzahl möglicher Geschichts-Enden
Mit jeder Frage teilt sich eine Geschichte in zwei unterschiedliche Handlungsstränge auf. Jeder Handlungsstrang kann wiederum eine weitere Frage aufweisen, welche die Geschichte noch weiter variiert.
Somit kann eine Geschichte mit n Fragen zwischen n+1 und 2^n mögliche Ausgänge finden.

Bei dem oben aufgeführten Beispiel sind es drei Enden:

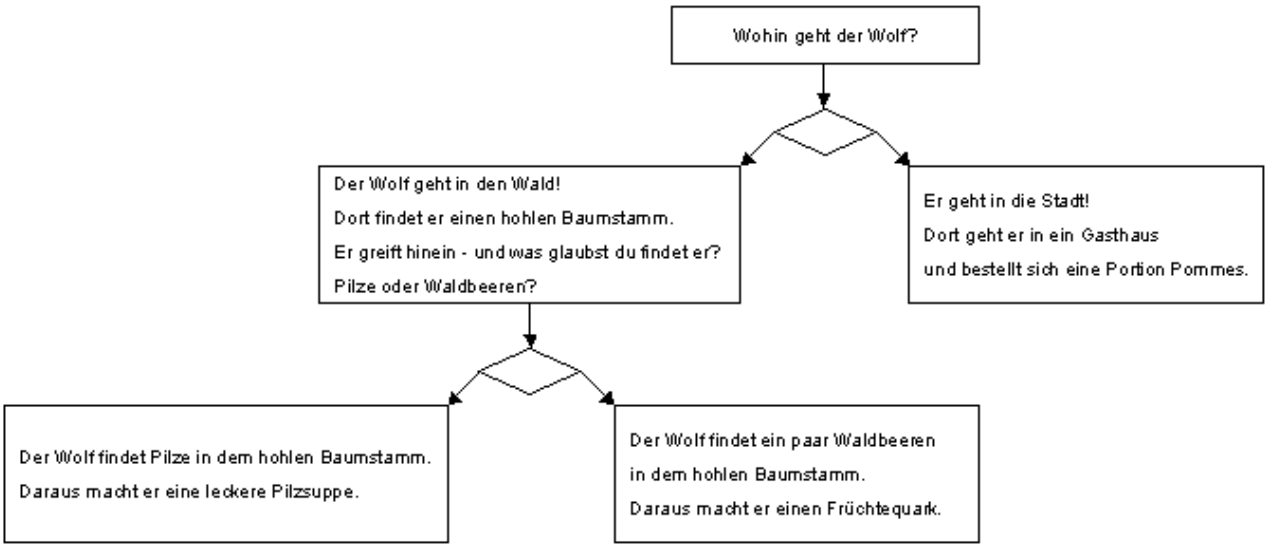


Abb. 52: Verzweigungen der Test-Geschichte

Diese Geschichte diente jedoch nur zur Erklärung der Abläufe - zu Vorführzwecken wurde noch eine längere Geschichte mit ca. 500 Zeilen und 7 möglichen Ausgängen verfasst.
Das Vorlesen eines Geschichtsstranges der ausführlichen Geschichte vom Anfang bis zum Ende dauert – abhängig von der Vorlesegeschwindigkeit - bis zu 10 Minuten.
Die Geschichte befindet sich mit auf der CD.

Manche finden Robotik etwas kindisch.
Nun, Kinder sind die geborenen Robotiker.
Sie hauchen Plüschtieren, Puppen und Plastikindianern Leben ein, und sie funktionieren von jeher vorhandenes Gerät um, etwa indem sie aus einem Kochlöffel und einem Holzklotz einen Bagger bauen.
Irgend etwas, heiße es Erziehung oder Kultur, zwingt die meisten von ihnen später, etwas anderes zu werden als Roboterforscher.

Gero von Randow

Epilog

Eine Technikerarbeit in 90 Tagen.
So könnte man diese Abschlussarbeit untertiteln.

Oft traten in dieser Zeit unvorgesehene Probleme auf.
Oft drohte dabei, dass die Zeitplanung in Verzug gerät oder die ehrgeizigen Ziele nicht erreichbar sein würden.
Oft wurde dann bis tief in die Nacht und den frühen Morgen getüftelt, gewerkelt und programmiert, um doch noch zu einer Umsetzung zu kommen.
Manchmal fanden sich unerwartet einfache Lösungen für schwierige Aufgaben.
Und manchmal mussten aufwendige Umwege gemacht werden.

Herauskam letztlich das umfangreiche Projekt eines komplexen Robotiksystems.
Und ein Roboter der immer wieder eine nachhaltige Wirkung bei Betrachtern findet.

Die Entwicklung von Super-Yano ist mit dieser Technikerarbeit aber nicht beendet.
Weitere Projekte und Experimente mit dem Roboter werden folgen.
Beispielsweise soll er Personen wiedererkennen können um sie namentlich anreden zu können. Auch wäre die Erweiterung zu einem Gesprächspartner, mit dem man sich „richtig“ unterhalten kann, eine reizvolle Aufgabe.

Einstweilen bin ich jedoch rückblickend mit den bisher erreichten Funktionen sehr zufrieden. Es war immer ein herausforderndes Projekt mit bewusst ambitioniert gewählten Zielen.
Dies ermöglichte es mir jedoch auch eine Fülle neuer Erfahrungen aus vielen anspruchsvollen Themenbereichen und Technologien zu sammeln, die ich in weitere Arbeiten einfließen lassen werde.

Abschließend möchte ich mich bei Herrn Schnaiter von der Gewerbeschule Offenburg für die Betreuung dieser Arbeit bedanken und vor allem meiner Familie, die während dieses Projekts viel auf mich verzichten musste und ohne deren Unterstützung Super-Yano nicht den geschaffenen Umfang hätte.

Linkverzeichnis

#	URL	Datum des letzten Aufrufes
1	http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/	23.04.2007
2	http://www.spiegel.de/panorama/0,1518,443593,00.html	24.06.2007
3	http://web.media.mit.edu/~stefanm/yano/	06.05.2007
4	http://www.cypax.net/snippets/	03.06.2007
5	http://www.intel.com/technology/computing/opencv/index.htm	03.06.2007
6	http://research.microsoft.com/srg/sapi.aspx	31.05.2007
7	http://www.nuance.de/naturallyspeaking/	20.04.2007
8	http://www.epinions.com/kifm-Toys-All-The_Original_San_Francisco_Toy_Makers_Yano_Interactive_Storyteller	03.06.2007
9	http://www.youtube.com/watch?v=aY2-XaVG8ng&eurl=http%3A%2F%2Fwww%2Ecypax%2Enet%2F	03.06.2007
10	http://www.parallax.com/detail.asp?product_id=28823	29.06.2007
11	http://www.pollin.de/shop/shop.php?cf=detail.php&pg=OA==&a=NzY3OTc4OTk=&w=OTg4OTk4&ts=40	03.06.2007
12	http://fowie.com/WordPress/wp-content/uploads/2006/01/OpenCV_visualstudio.pdf	27.04.2007
13	http://winavr.sourceforge.net/	03.06.2007
14	http://www.annosoft.com/phoneset.htm	30.05.2007
15	http://research.microsoft.com/srg/sapi.aspx	31.05.2007
16	http://actor.loquendo.com/actordemo/default.asp?page=id&voice=Stefan	03.06.2007
17	http://www.cypax.net/snippets	03.06.2007
18	http://www.nuance.de/naturallyspeaking/	20.04.2007
19	http://j-integra.intrinsyc.com/support/com/doc/other_examples/Dragon_NaturallySpeaking.htm	01.06.2007
20	http://support.lhsl.com/databases/dragon/webdisc.nsf	01.06.2007

Abbildungsverzeichnis

(Bilder ohne Quellenangabe wurden vom Autor selbst erstellt)

Quellenverzeichnis

#	Autor, Beschreibung und Quelle	Datum des letzten Aufrufes
	Bildverarbeitung	
1	Yahoo OpenCV-Forum http://tech.groups.yahoo.com/group/OpenCV/	25.05.2007
2	Alexander Kubias: OpenCV - Open Source Computer Vision Library http://www.uni-koblenz.de/~kubias/FolienOpenCV.pdf	23.05.2007
3	Gady Agam: Introduction to programming with OpenCV http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/index.html	03.06.2007
	Sprachausgabe	
1	Brian Long: Speech Synthesis & Speech Recognition http://www.blong.com/Conferences/DCon2002/Speech/Speech.htm	03.06.2007
2	Felix Burkhardt: Deutsche Sprachsynthese http://ttssamples.syntheticspeech.de/deutsch/index.html	03.06.2007
3	Mark Zartler: Information on converting phonemes to visemes http://www.annosoft.com/phoneset.htm	03.06.2007
4	Microsoft: Speech Application Programming Interface (SAPI) Software Development Kit (SDK) http://research.microsoft.com/srg/sapi.aspx	31.05.2007
5	Davide Bonardo, Paolo Baggia: SSML 1.0: an XML-based language to improve TTS rendering http://www.loquendo.com/en/technology/SSML.htm	03.06.2007
	Spracherkennung	
1	Rod Owen LLC: Dragon NaturallySpeaking SDK Client Edition Developer's Guide http://rodowen.com/images/dscsdk.zip	01.06.2007
2	Speech Recognition Technology Discussion List http://www.edc.org/spk2wrt/hypermail/	23.04.2007
3	Sven Wiener, Ahmad Hashemi-Sakhtsari, Oliver Carr: Accessing Dragon NaturallySpeaking (DNS) from Java http://j-integra.intrinsyc.com/support/com/doc/other_examples/Dragon_NaturallySpeaking.htm	03.06.2007
4	ScanSoft Forum http://support.lhsl.com/databases/dragon/webdisc.nsf	01.06.2007
	Anderes	
1	Mark Newman & Stefan Marti: F.A.C.E. Project http://web.media.mit.edu/~stefanm/yano/	03.06.2007
2	Manfred Claßen: Einführung in Visual C++ http://www.rz.rwth-aachen.de/mata/downloads/visual/skript.pdf	05.06.2007
3	Microsoft: Einlesen von Daten aus XML-Dateien mit Visual Studio http://support.microsoft.com/kb/815658/de	03.06.2007

Weitere themenverwandte interessante Artikel und Links

Cynthia Breazeal & Brian Scassellati: Robots that imitiate humans
http://web.media.mit.edu/~cynthiab/Papers/Breazeal-TICS02.pdf

Cynthia Breazeal u.a.: Humanoid Robots as Cooperative Partners for People
http://web.media.mit.edu/~cynthiab/Papers/Breazeal-etal-ijhr04.pdf

Cynthia Breazeal & Rodney Brooks: Robot Emotion: A Functional Perspective
http://web.media.mit.edu/~cynthiab/Papers/Breazeal-Brooks-03.pdf

Chris Willis: Android World
http://www.androidworld.com/index.htm

Ronny Kober: Verfahren zur Gesichtserkennung
http://www.ronnykober.de/download/frei/doku_opencv.pdf

Technische Universität München: Diplomarbeiten zur Bildverarbeitung
http://vision.cs.tum.edu/teaching/da_announced.php

Florian Adolf: OpenCV's Rapid Object Detection
http://robotik.inflomatik.info/other/opencv/OpenCV_ObjectDetection_HowTo.pdf

Roboter sind der volle Ausdruck dessen, wer ich bin und was ich bin.
Der Bau von Robotern fordert meine ganze Kraft heraus.
Die Niederlagen sind grässlich, die Triumphe total.

Red Whittaker