Ausarbeitung

Roboter-Kommunikation mit externen Systemen

im Rahmen des Seminars

Unterstützung von Landminendetektion durch Bildauswertungsverfahren und Robotereinsatz

vorgelegt von: Patrick Arendt (284338)
Bent Jakuboski (284467)
Andreas Wulf (284429)
Abgabetermin: 2004-01-05

Inhaltsverzeichnis

1 Einführung

Die automatisierte Detektion von Landminen erfordert Roboter, die autonom und zielgerichtet in ungewissen Umgebungen operieren. Zur Erfüllung dieser Aufgabe benötigt ein Roboter Zugriff auf unterschiedliche externe Systeme, sowohl in Form von direkt zum Agenten gehörenden Komponenten, als auch in Form von entfernten, unabhängigen Systemen.

Ziel dieser Arbeit ist die Entwicklung eines Roboters auf Basis des LEGO® Robotics Invention Systems 2.0[1], der mit einem Laptop ausgestattet über eine Kamera Objekte auffindet und Informationen mit einem entfernten Host-Rechner austauscht. Im folgenden werden sowohl die theoretischen Grundlagen als auch Möglichkeiten zur praktischen Anwendung der dabei betroffenen Kommunikationsebenen und -schnittstellen vorgestellt. Konkret wird im zweiten Kapitel die Kommunikation zwischen RCX[2] und Laptop mit Hilfe des LEGO Infrarot-Towers beschrieben. Das darauf folgende Kapitel widmet sich verschiedenen Möglichkeiten der Ansteuerung der Kamera zur Bildauswertung. Im vierten Kapitel werden Verfahren zum drahtlosen Austausch von Informationen zwischen Roboter und entfernten Systemen erläutert und bewertet. Auf Basis der in den vorangegangen Kapiteln gewonnenen Erkenntnisse wird im fünften Kapitel ein Roboter konstruiert, der autonom nach bestimmten Objekten sucht. Abschließend fasst das sechste Kapitel die dargestellten Inhalte bewertend zusammen und gibt einen Ausblick auf Ergänzungsmöglichkeiten.

---

1 Vgl. http://mindstorms.lego.com/eng/products/ris/index.asp.

2 Im Robotics Invention System enthaltener programmierbarer Mikroprozessor. Vgl. http://robofesta.open.ac.uk/techspec.html.

2 Kommunikation zwischen RCX und Laptop

Herzstück des LEGO-Roboters ist der so genannte RCX, ein auf dem Renesas[3] H8-Mikrocontroller basierender, programmierbarer LEGO-Stein, der sowohl bis zu drei Motoren ansteuern als auch auf die Daten von bis zu drei externen Sensoren zugreifen kann. Um mit anderen Einheiten kommunizieren zu können, besitzt der RCX weiterhin eine Infrarot-Schnittstelle, die aus einem Infrarot-Sender und Empfänger besteht.

2.1 LEGO Infrarot-Tower

Als Gegenstück zur Infrarot-Schnittstelle des RCX liefert LEGO im Robotics Invention System 2.0 einen Infrarot-Tower mit, der über den USB-Port[4] an einen PC angeschlossen wird. Über diesen Tower können sowohl Befehle an den RCX gesendet als auch Rückmeldungen vom RCX empfangen werden. Als Träger für den seriellen Strom der zu übertragenden Daten wird Infrarotlicht mit einer Frequenz von standardmäßig 38 kHz verwendet. Die Übertragungsrate beträgt bei Verwendung der empfohlenen Standardwerte 2400 Baud, wobei eine maximale Rate von 4800 Baud durch Verdoppelung der Übertragungsfrequenz auf 76 kHz erreicht werden kann.

Bei dem LEGO Infrarot-Tower handelt es sich um ein so genanntes Consumer Infrared-Gerät (CIR Device). CIR-Geräte zeichnen sich dadurch aus, dass sie Daten mit geringer Geschwindigkeit über eine relativ lange Distanz übertragen, und werden in der Regel zur Fernbedienung von elektronischen Endgeräten, wie beispielsweise Fernseher, Videorekorder oder HiFi-Anlagen, eingesetzt. Da Consumer IR bisher nicht standardisiert ist, sind die von den einzelnen Herstellern implementierten Protokolle typischerweise nicht kompatibel zueinander und unterscheiden sich beispielsweise bezüglich der Modulationsfrequenzen, Puls- und Pausenzeiten, Bitmustern oder Fehlerüberprüfungsverfahren. Da es sich auch bei dem von LEGO eingesetzten Übertragungsprotokoll um eine Eigenentwicklung handelt, ist es nicht ohne Weiteres möglich, die üblicherweise in PCs, Laptops oder Handhelds verwendete IrDA-Schnittstelle zur Infrarot-Kommunikation mit dem RCX zu verwenden.

Anders als CIR ist IrDA ein international anerkannter Industriestandard mit einer spezifizierten Protokollhierarchie. IrDA wurde zur bidirektionalen Datenübertragung über kurze Distanzen entwickelt und sieht im IrDA 1.1-Standard eine maximale Datenrate von 4 Mbit/s vor, dem 1748fachen der im RCX-Protokoll festgelegten Standardübertragungsgeschwindigkeit. Durch die hohen Anforderungen an Geschwindigkeit und Vielseitigkeit ist

---

[3] Ehemals Hitachi.

[4] Bis Version 1.5 des Robotics Invention Systems wurde der Tower über die serielle Schnittstelle angeschlossen.

das IrDA-Protokoll weitaus komplexer als das von LEGO verwendete proprietäre CIR-Protokoll und damit grundsätzlich nicht kompatibel. Somit lässt sich der RCX nicht ohne größere Anpassungen an Treiber und Software mit einem IrDA-Controller ansteuern. Daher wird hier auf die Verwendung eines Handhelds, der direkt über die integrierte IrDA-Schnittstelle mit dem RCX kommuniziert, verzichtet, und stattdessen ein Laptop samt LEGO Infrarot-Tower eingesetzt.

2.1.1 Programmierung des Towers

Über den symbolischen Gerätenamen "LEGOTOWER1" bietet der Windows-Treiber des USB-Towers Zugriff auf die Funktionalität des Geräts[5]. Der Gerätename wird der API-Funktion "CreateFile" übergeben, welche die Kommunikation zwischen PC und Tower initiiert und ein Handle zurückliefert, das für den weiteren Zugriff benötigt wird. Das Senden von Bytes über den Infrarot-Tower erfolgt über "WriteFile". Dieser Funktion werden das zuvor erhaltene Handle des Towers, ein Puffer mit den zu sendenden Daten, die Anzahl der zu verschickenden Bytes und ein Zeiger auf eine Variable, in welche die Anzahl der tatsächlich gesendeten Bytes geschrieben wird, übergeben. Analog dazu erfolgt das Lesen von empfangenen Daten mit Hilfe von "ReadFile". Abschließend sollte das Objekt-Handle mit "CloseHandle" wieder freigegeben werden. Der Quellcode in Anhang A demonstriert die Verwendung dieser Funktionen in Form eines simplen Beispiels.

2.2 Kommunikationsprotokoll

Das Protokoll zur Kommunikation zwischen Infrarot-Tower und RCX lässt sich in vier Schichten einteilen, deren Abstraktionsniveau im folgenden von Schicht zu Schicht, ausgehend von der Beschreibung von Befehlen durch Bytecode bis hin zur physischen Bitkodierung mittels Infrarotstrahlung, abnimmt.

2.2.1 Bytecode-Schicht

Die Steuerung bzw. Programmierung des RCX erfolgt bei Verwendung der Standard-Firmware über LASM[6] Bytecode-Befehle. Jede Bytecode-Instruktion besteht dabei aus einem ein Byte langen Opcode und eventuell notwendigen Operanden. Auflistungen der verfügbaren Befehle finden sich in [Le00] und [Pr99]. Als Beispiel soll hier der Ping-Befehl mit dem Opcode `0x10` dienen, der lediglich überprüft, ob der RCX eingeschaltet und bereit ist. Zur Bestätigung des Empfangs erwidert der RCX Anfragen stets mit dem Komplement des gesendeten Opcodes, in diesem Fall `0xEF`. Empfängt der RCX zweimal

---

5 Weitere eventuell angeschlossene Tower heißen entsprechend LEGOTOWER2, LEGOTOWER3, ...

6 LEGO Assembler.

nacheinander den gleichen Opcode, wird die zweite Nachricht ignoriert, da der RCX hierbei von einem Fehler bzw. Missverständnis bei der Datenübertragung ausgeht. Aus diesem Grund existiert zu nahezu jedem Opcode ein alternativer Code, der sich durch Addition von `0x08` berechnen lässt. Somit ergibt sich `0x18` als alternativer Opcode zum Ping-Befehl `0x10`.

2.2.2 Paketschicht

Auf der Paketschicht werden die Bytecode-Instruktionen in Pakete verpackt. Wie in Abbildung 1 dargestellt, beginnt jedes Paket mit einem festen, drei Byte langen Kopf. Anschließend folgen die Opcodes und Operanden der Bytecode-Schicht, wobei jedem Byte sein Komplement angehängt wird. Das Ende eines Pakets bildet eine Prüfsumme, die sich als Summe über alle Datenbytes ohne Berücksichtigung von Überträgen errechnet und der ebenfalls ihr Komplement folgt.

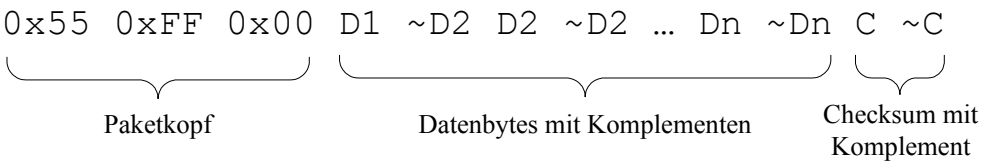

Abb. 1: Paketstruktur

Der vorgestellte Ping-Befehl `0x10` wird auf dieser Ebene folglich zu `0x55 0xFF 0x00 0x10 0xEF 0x10 0xEF` kodiert. Wie das Beispiel in Anhang A zeigt, reichen die bis hierhin vorgestellten Informationen über das Kommunikationsprotokoll bereits aus, um den RCX ansprechen zu können, da Treiber und Hardware des Towers die Aufgaben der beiden folgenden Schichten übernehmen.

2.2.3 Byte-Kodierungsschicht

Auf der Byte-Kodierungsschicht kodiert der Treiber des Infrarot-Towers die einzelnen Bytes der Pakete für den Versand. Jedem Byte wird das Startbit 0 vorangesetzt und sowohl ein so genanntes Odd-Parity-Bit als auch das Stopbit 1 angehängt. Somit wird, entsprechend Abbildung 2, jedes Datenbyte auf insgesamt elf Bits ausgedehnt.

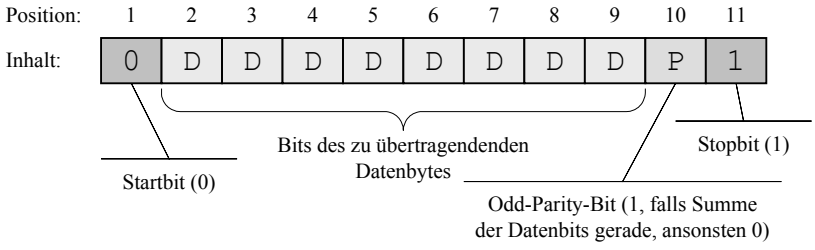

Abb. 2: Kodierungsschema

2.2.4 Infrarot-Übertragungsschicht

Die physikalische Übertragung der Daten erfolgt, wie in Abschnitt 2.1 erwähnt, standardmäßig mit einer Übertragungsrate von 2400 Baud. Dadurch ergibt sich eine Übertragungszeit von etwa 417 µs pro Bit (1/2400 bps), wobei jede 0 als 417 µs gepulste Infrarotstrahlung und jede 1 als 417 µs lange Pause kodiert wird.

3 Kommunikation zwischen Webcam und Laptop

Eine Webcam – als visueller Sinn – ist die mächtigste Schnittstelle des Roboters mit der Außenwelt. Umso bedachter sollte die Auswahl der Mittel erfolgen, mit denen auf das „Roboterauge“ zugegriffen wird, denn dieses liefert die Grundlage für alle anschließenden Algorithmen und Verfahren zur Bilderkennung und Musterauswertung.

Ein erster, simpler Ansatz, um an die Daten einer Webcam zu gelangen, ist das direkte Ansprechen des Gerätetreibers. Problematisch hierbei ist die Tatsache, dass Treiber keine einheitliche Schnittstelle haben, so dass eine Anwendung im Extremfall mit nur einer spezifischen Hardware und einer einzigen Treiberversion funktionsfähig sein kann, oder für alle verfügbaren Gerätemodelle angepasste Zugriffsmethoden implementiert werden müssen.

Multimedia-Frameworks lösen dieses Problem, indem sie dem Anwender Hardwareunabhängigkeit[7] anbieten. Dieser muss sich keine Gedanken mehr um Gerätemodelle und -versionen machen, sondern kann sich auf die Funktionalität seiner Anwendung konzentrieren.

Im Folgenden werden drei verschiedene Multimedia-Frameworks vorgestellt und auf ihre Tauglichkeit als Mittel zur Beherrschung der Kommunikation mit einer Roboter-Webcam hin untersucht: Aus der Windows-Welt stammen *Video for Windows* und *DirectX*, während das *Java-Media-Framework* einen plattformübergreifenden Open-source-Ansatz repräsentiert.

3.1 Video for Windows

3.1.1 Entstehung

Anfang der neunziger Jahre zeichnete sich eine wachsende Bedeutung von Multimedia in der Computerwelt ab. Steigende Prozessorleistung und höhere Speicherkapazität ermöglichten neuartige multimediale Anwendungen, für die die bisherige Konzeption der aktuellen Betriebssysteme nicht ausgelegt war. Das Unternehmen Microsoft® gelang zu der Überzeugung, dass das Windows-eigene *GDI*[8] für die neuen Anforderungen zu leistungs-

---

[7] Vgl. Kapitel 3.2.2.1, in dem die besonders eindrucksvolle und effiziente Lösung der DirectX-Technologie beschrieben wird.

[8] Das Graphics Device Interface, kurz GDI, ist als Bestandteil des Windows-Betriebssystems für die Bereitstellung von Grafikprimitiven zuständig und dient als Schnittstelle zwischen Anwendung und Gerätetreiber. Vgl. auch die offizielle Dokumentation unter http://msdn.microsoft.com/library/en-us/winprog/winprog/graphics_device_interface.asp.

schwach war, und entwickelte eine Sammlung von Schnittstellen zur Aufnahme und Wiedergabe von Audio- und Videoclips – *Video for Windows* (im Folgenden VFW genannt).

Im Jahr 1992 wurde erstmals das VFW-SDK[9] für die Entwicklung von Anwendungen veröffentlicht. Später wurde VFW integraler Bestandteil der Betriebssysteme Windows 95 und Windows NT 3.51[10]. Nach der Veröffentlichung von Windows 98 stellte Microsoft die Entwicklung ein, gab Entwicklern jedoch die Zusage, dass VFW in allen zukünftigen Windows-Versionen enthalten sein werde.

3.1.2 Die Programmierschnittstelle „AVICap“

Zugriffe auf Videogeräte erfolgen bei VFW grundsätzlich über das API[11] *AVICap*. Dieses bietet eine einheitliche Schnittstelle zwischen Anwendung und Hardware, indem es die Details der Kommunikation mit den Gerätetreibern verbirgt, und garantiert damit Hardwareunabhängigkeit. *AVICap* ermöglicht die Bearbeitung und Speicherung von sowohl Streaming-Videodaten als auch Einzelbildern. Bei der Programmierung mit *AVICap* sind vier grundlegende Schritte zu beachten:

(1) Erstellen eines Aufnahmefensters

Die Funktion *capCreateCaptureWindow()* erzeugt ein Capture-Fenster[12], dessen Eigenschaften durch die übergebenen Parameter bestimmt werden. Diese legen den Fenstertitel fest, der in der Fensterleiste angezeigt wird, den Fensterstil, die Anfangposition, die Auflösung des Kamerabildes, das Handle des Elternfensters und einen eindeutigen Identifier.

Falls erfolgreich, liefert die Funktion das Handle des erstellten Capture-Fensters zurück.

```
CapHWnd = capCreateCaptureWindow(
     "Capture-Fenster",          // Fenstername
     WS_CHILD | WS_VISIBLE,      // Fensterstil
     0, 0,                       // Fensterposition
     320, 240,                   // Fensterdimensionen
     ParentHWnd,                 // Handle des Elternfensters
     0);                         // Child ID
```

(2) Verbinden mit einem Videotreiber

---

9 Software Development Kit.

10 Zum Verdruss vieler Entwickler sind die VFW-Ausgaben für Windows 95/98 auf der einen Seite und Windows NT auf der anderen nicht miteinander kompatibel, da erstere auf 16-Bit- und letztere auf 32-Bit-Technologie basieren.

11 Application Programmers Interface.

12 Mit dem Begriff „Capture-Fenster“ ist in diesem Fall ein Anwendungsfenster gemeint, das die von einer Webcam eingelesenen Bilder anzeigt.

Anschließend muss das erstellte Capture-Fenster noch mit dem Videotreiber in Verbindung gebracht werden, der als Datenquelle fungiert. Dieses kann auf zwei Weisen geschehen: Der direkte Weg besteht darin, eine Nachricht (*Message)* zu senden, die das Fenster und die Videoquelle miteinander verknüpft. Als Alternative dazu kann auch auf vorgefertigte Makros zurückgegriffen werden, die die Handhabung vereinfachen und den Quelltext übersichtlicher halten.

Verbinden eines Capture-Fensters mit einer Videoquelle:

... mittels SendMessage:

```
Success = SendMessage(CapHWnd, WM_CAP_DRIVER_CONNECT, 0, 0L);
```

... und mittels eines Makros:

```
Success = capDriverConnect(CapHWnd, 0);
```

(3) Festlegen der Bildwiederholrate

Mittels des Makros *capPreviewRate* kann die Bildwiederholrate festgelegt werden. Als Parameter wird keine Hertz-Kennzahl (Frames pro Sekunde) übergeben, sondern die Länge in Millisekunden, die ein einzelner Frame angezeigt wird (Millisekunden pro Frame). Einer Bildwiederholrate von 24 Hertz entspricht folglich einer Anzeigedauer von ca. 42 ms pro Frame.

Festlegung der Bildwiederholrate auf ca. 24 Hertz:

```
capPreviewRate(CapHWnd, 42);  // Rate in ms
```

(4) Starten und Beenden der Videodarstellung

Mit Hilfe des Makros *capPreview* lässt sich die Anzeige des Videosignals über den übergebenen Parameter starten und beenden.

Starten der Videodarstellung:

```
capPreview(CapHWnd, TRUE);  // Anzeige starten
```

Beenden der Videodarstellung:

```
capPreview(CapHWnd, FALSE); // Anzeige beenden
```

3.2 DirectX

3.2.1 Entstehung

Konzeptionelle Schwächen vorhandener Multimedia-Frameworks und stetig steigende Anforderungen an Multimedia-Leistungsfähigkeit veranlassten Microsoft dazu, eine Sammlung von APIs zusammenzustellen, die drei grundlegenden Zielen gerecht werden sollte: hohe Performance, hardwarenaher Zugriff und ressourcenschonendes Laufzeitverhalten.[13]
1995 wurde dieses auf dem *Component Object Model* (COM) basierende Framework unter dem Namen *DirectX* veröffentlicht. Die Version 1.0 erwies sich kurz nach Einführung als zu unausgereift und konnte sich nicht durchsetzen. Erst mit Veröffentlichung der Version 2.0 gelang *DirectX* der Durchbruch. Mittlerweile liegt die Version 9.0b vor.

3.2.2 Architektur

Basis von DirectX ist eine Drei-Schichten-Architektur. Schnittstelle zur Hardware ist die *DirectX-Foundation*-Schicht, die die Details der Gerätetreiber vor den höheren Schichten verbirgt. Sie umfasst Low-Level-APIs wie *DirectDraw* und *DirectSound*. Darauf aufsetzend beinhaltet die *DirectX-Media*-Schicht fünf High-Level-APIs, die auf Funktionen der Foundation-Schicht zugreifen. Die High-Level-API *DirectShow*, wird im nächsten Kapitel näher erläutert. Oberhalb der Media-Schicht befindet sich die *Components-Schicht* auf, welche eine Reihe von anwendungsspezifischen Modulen umschließt, darunter zum Beispiel das für Video-Konferenzen genutzte *Netmeeting*.

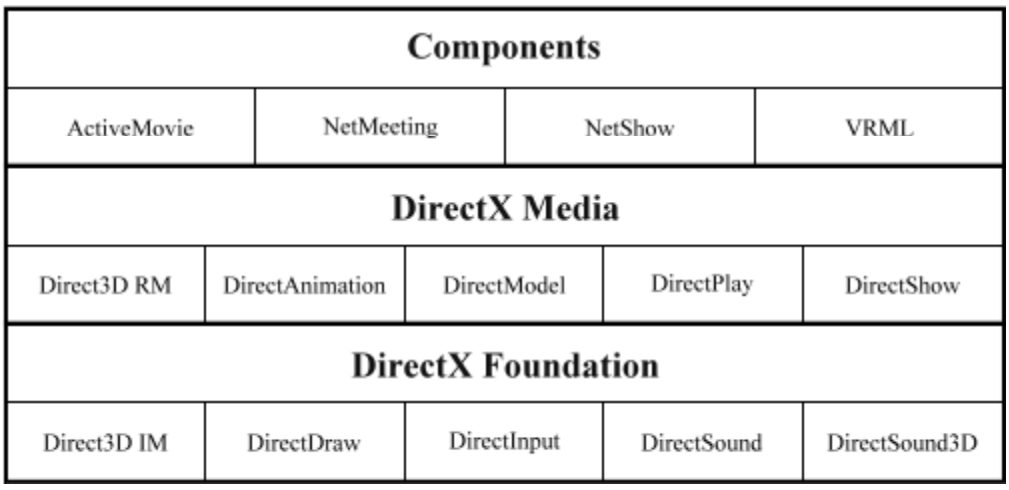

Abb. 3: DirectX-Architektur und Komponenten

[13] Vgl. [BD98].

3.2.2.1 Hardwareunabhängigkeit

Ein fundamentaler Gedanke von DirectX ist die Unabhängigkeit von verwendeten Multimedia-Geräten. Um diese zu erreichen, also dem Entwickler von Multimedia-Software die Auseinandersetzung mit diversen unterschiedlichen Hardwaretreibern zu ersparen, verwendet Microsoft ein Zwei-Schichten-Prinzip bestehend aus *Hardware Abstraction Layer* (HAL) und *Hardware Emulation Layer* (HEL).

Die HAL ist Bestandteil DirectX-kompatibler Hardware und stellt alle Features, über die die Hardware verfügt, direkt zur Verfügung. Die HEL ist Bestandteil des DirectX-Systems – also Software – und implementiert alle Features, die DirectX bietet. Dieses Konzept ermöglicht es, dass eine Anwendung mit Hardware funktioniert, die nicht alle benötigten Features zur Verfügung stellt.

Beispielsweise beinhaltet der Treiber einer DirectX-kompatiblen Grafikkarte eine *DirectDraw*-HAL und eine *Direct3D*-HAL mit jeweils allen DirectX-Features, die die Grafikkarte unterstützt. Nutzt eine Anwendung ein DirectDraw-Feature, so wird zunächst überprüft, ob dieses von der Grafikkarte unterstützt wird. Wenn dem so ist, wird es über die *DirectDraw*-HAL direkt angesprochen. Unterstützt die Karte das Feature nicht, so emuliert die *DirectDraw*-HEL dieses als Software-Implementierung.

3.2.3 DirectShow

DirectShow ist eine objektorientierte Schnittstelle zur Wiedergabe von Audio- und Videodaten. Innerhalb der DirectX-Architektur ist sie als High-Level-API auf der *Media-Schicht* angesiedelt, d.h. sie selbst greift intern auf Funktionalitäten von Low-Level-APIs der Foundation-Schicht zu, wie zum Beispiel *DirectSound*.

DirectShow bietet dem Entwickler ein enormes Potential; so erlaubt es den Zugriff auf eine Vielzahl an Geräten, wie TV-Karten und Webcams, und arbeitet mit allen gängigen Multimedia-Datenformaten, wie MP3, AVI, und DivX, zusammen. Die Funktionspalette reicht dabei vom einfachen Abspielen und Bearbeiten von Dateien bis zum Konvertieren von Formaten.[14]

[14] Vgl. [BN03, S. 56].

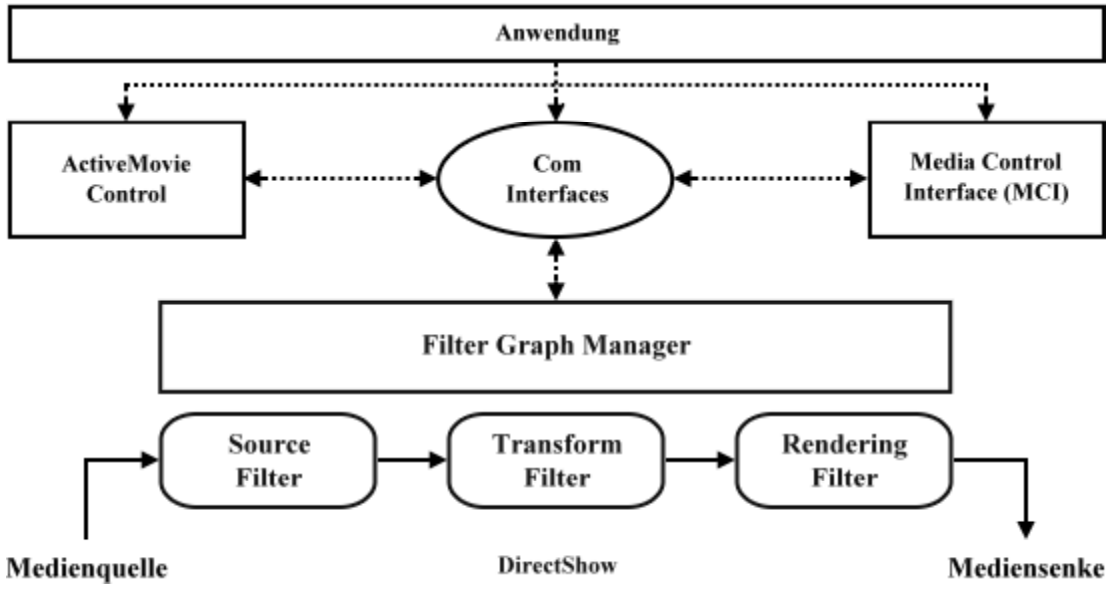

Abb. 4: DirectShow-Architektur

Eine bestimmte Aufgabe, wie zum Beispiel das Abspielen einer AVI-Datei, wird in DirectShow durch einen gerichteten Graphen (*FilterGraph*) repräsentiert. Dieser besteht aus modularen Komponenten (*Filtern*), die mittels Ein- und Ausgabeschnittstellen (*Pins*) miteinander verbunden werden können. Es werden drei Arten von Filtern unterschieden:

- *Source Filter* lesen Daten aus einer beliebigen Quelle, wie der Festplatte oder dem Internet und geben diese aus.
- *Transform Filter* nehmen Daten entgegen, bearbeiten sie und geben die veränderten Daten aus.
- *Rendering Filter* nehmen Daten entgegen und speichern sie an einem festgelegten Platz.

Eine übergeordnete Komponente, der *FilterGraphManager*, ist für das Erstellen und Konfigurieren des *FilterGraphen* zuständig. Außerdem koordiniert und kontrolliert sie den Datenfluss und dient als Schnittstelle für die zugreifende Applikation.

Beispiel: Abspielen einer AVI-Datei.

Nach dem Einlesen werden Audio- und Video-Stream voneinander getrennt. Der Video-Stream wird daraufhin dekomprimiert, gerendert und auf dem Bildschirm ausgegeben, während der Audio-Stream an eine DirectSound-Komponente übergeben wird, die ihn schließlich über die Soundkarte ausgibt. Für jeden Schritt ist ein entsprechender Filter zuständig (siehe Abbildung 5).

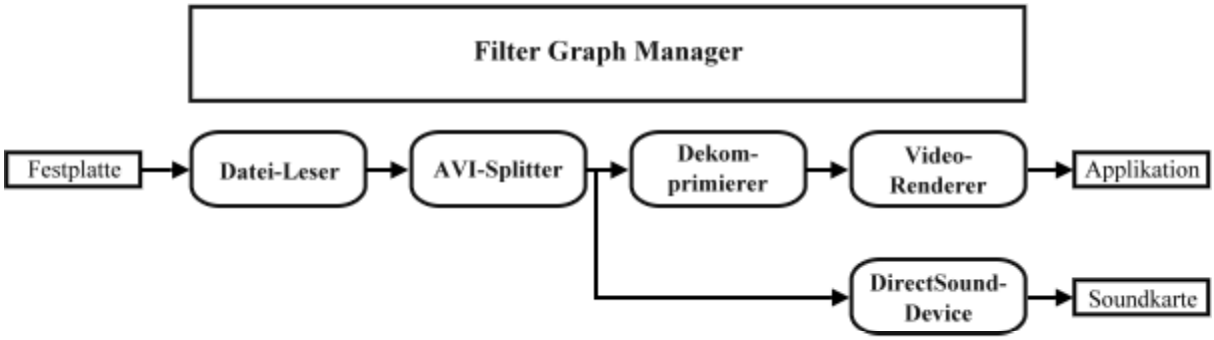

Abb. 5: FilterGraph zum Abspielen einer AVI-Datei

In einem DirectShow-Programm wird zunächst ein *FilterGraphManger-Objekt* erzeugt, mit dessen entsprechenden Methoden ein *FilterGraph* erstellt und konfiguriert wird, der den beschriebenen Ablauf widerspiegelt. Sobald dem *FilterGraphManager* ein eindeutiger Pfad zu einer AVI-Datei übergeben wird, initiiert und kontrolliert dieser den Media-Stream.

3.3 Java Media Framework

3.3.1 Entstehung

Im Jahr 1997 definierten Sun, Intel und Silicon Graphics in Kooperation die Version 1.0 des *Java Media Framework* (JMF). Zielsetzung dabei war die Schaffung einer API für die Integration von zeitabhängigen Medien in Java-Applets und -Applikationen, die das Auslesen, Verarbeiten und Präsentieren der Multimediadaten ermöglicht.[15] Von drei angebotenen Implementierungen konnte sich jedoch keine durchsetzen, was vor allem am stark eingeschränkten Funktionsumfang lag. Bereits 1998 zerbrach die Allianz als Intel seinen Rücktritt erklärte. Mit IBM gewann Sun anschließend einen neuen kompetenten Partner, so dass 1999 die Version 2.0 des JMF veröffentlicht werden konnte, die erstmals auch Videocapturing ermöglichte. Mittlerweile liegt das JMF in der Version 2.1.1 vor und befindet sich in fortlaufender Weiterentwicklung.

3.3.2 Architektur

Dem JMF liegt das gleiche Modell zu Grunde, das auch beim alltäglichen Gebrauch einer Multimedia-Anlage Anwendung findet: Mit einer Videokamera werden Bild- und Tondaten auf einer Videokassette gespeichert. Dem Videorekorder wird durch Einführen der Kassette ein Media-Stream zur Verfügung gestellt, den dieser ausliest und interpretiert.

---

[15] Für weitergehende Informationen vgl. http://java.sun.com/products/java-media/jmf/2.1.1/guide/JMFTBM.html.

Schließlich sendet der Rekorder entsprechende Signale an den Fernseher und die Lautsprecher.

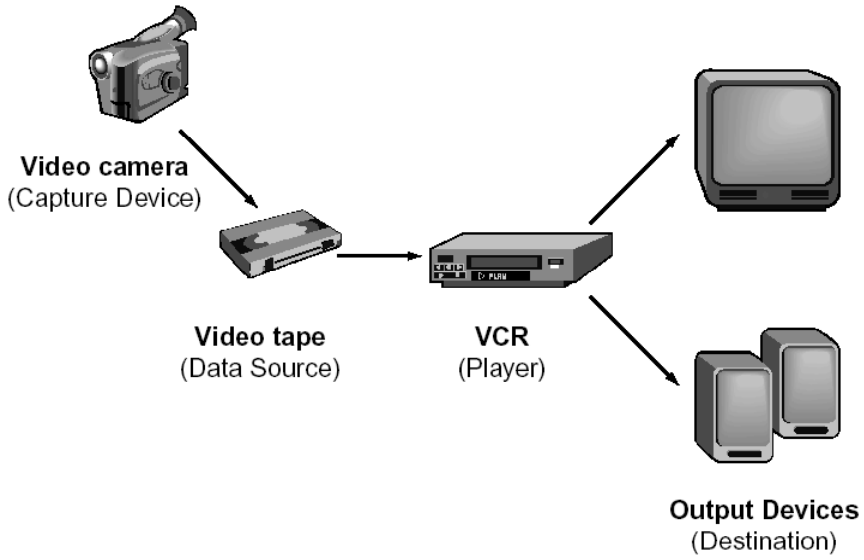

Abb. 6: Architektur des JMF

Im JMF liest ein *CaptureDevice* z.B. Daten einer Webcam ein und legt sie in einer *DataSource* – als Pendant zur Videokassette – ab. Ein *Player* greift nun, so wie ein Videorekorder, auf den Media-Stream der *DataSource* zu, interpretiert die Daten und sendet schließlich Signale an Ausgabegeräte, wie z.B. Grafik- und Soundkarte.

Die angesprochenen Elemente sind allesamt Bestandteile der High-Level-API des JMF. Daneben existiert noch eine Lower-Level-API, die es Entwicklern erlaubt, auf einzelne Komponenten des Streaming-Prozesses zuzugreifen, sie in eigene Programme einzubinden und zu erweitern.

3.3.3 Programmierung

Das JMF beinhaltet einige Komponenten mit fest definierten Aufgabenbereichen, deren Kenntnis für das nachfolgende Quellcode-Beispiel erforderlich ist.

3.3.3.1 Manager

Manager steuern und verwalten Objekte, die zum Capturing, Verarbeiten und Präsentieren von Media-Streams benutzt werden, und sind damit direkte Ansatzpunkte für den Programmierer. Es werden insgesamt vier verschiedene Manager unterschieden:

Manager:	ist für die Konstruktion von Player-, Processor-, DataSource- und DataSink-Objekten verantwortlich
PackageManager:	verwaltet Pakete mit JMF-Klassen.
CaptureDeviceManager:	verwaltet verfügbare Capture-Geräte, wie z.B. Webcams
PlugInManager:	verwaltet verfügbare JMF-Plugins, wie z.B. Codecs

Tab. 1: Unterschiedliche Manager-Typen im JMF

3.3.3.2 Controller

Controller-Objekte sind für die Präsentation der Multimediadaten verantwortlich und bieten entsprechende Kontrollmechanismen. Zu diesen zählt auch der bereits erwähnte *Player*, der für die Verarbeitung und das Rendering von Media-Streams verantwortlich ist. Eine Spezialform eines *Players* stellt der *Processor* dar, der zusätzlich die Nachbesserung des Streams durch Codecs und Effekt-Filter zulässt.

3.3.3.3 Programmierbeispiel: Video-Capturing[16]

Um im JMF Daten von einer Webcam einlesen, wiedergeben und gegebenenfalls weiterverarbeiten zu können, sind mindestens vier grundlegende Schritte notwendig:

(1) Zugriff auf die Webcam mittels des *CaptureDeviceManager*-Objektes und Instantiierung eines *CaptureDeviceInfo*-Objektes, das auf die Cam verweist. (Falls der Name der Webcam nicht bekannt ist, hilft ein Blick auf die Liste der verfügbaren Geräte, die die Methode *CaptureDeviceManager.getDeviceList* liefert)

```
CaptureDeviceInfo webcam =
  CaptureDeviceManager.getDevice("Webcam-Name");
```

(2) Um einen Media-Stream von der Webcam auslesen zu können, muss mit Hilfe des *CaptureDeviceInfo*-Objektes, „webcam“, ein *MediaLocator*-Objekt „mLocator“ erzeugt werden,

```
MediaLocator mLocator = webcam.getLocator();
```

(3) Mit dessen Hilfe wird ein *DataSource*-Objekt erzeugt und an einen Player übergeben.

```
Player camPlayer =
  Manager.getPlayer(mLocator.getDataSource());
```

(4) Durch Starten des *Players* beginnt der Capturing-Prozess.

```
camPlayer.start();
```

[16] Für weitere und detailliertere Beispiele vgl. http://java.sun.com/products/java-media/jmf/2.1.1/guide/

3.4 Vergleich der Alternativen

	Video for Windows	**DirectX**	**Java Media Framework**
Proprietät	closed-source	closed-source	open-source
Akzeptanz	+	++	-
Plattform	Windows	Windows	alle
Hardwareunabhängigkeit	+	++	+
Programmiersprachen	VB, VC++, C, Delphi, u.v.a.	VB, VC++, C, Delphi, u.v.a.	Java
Komplexität	+	-	o
Funktionalität	+	++	+
Performance	+	++	o

Tab. 2: Gegenüberstellung von VFW, DirectX und JMF

In der Frage, welches Multimedia-Framework bei der Integration einer Webcam in ein Robotersystem zur Programmierung gewählt werden sollte, lässt sich keine generelle Empfehlung geben. Viele externe Faktoren haben Einfluss auf diese Entscheidung. So sind nicht nur die Präferenzen und Fähigkeiten der Programmierer zu berücksichtigen, sondern auch die Anforderungen, die sich aus der Bildweiterverarbeitung ergeben.

Eine Übersicht einiger ausgewählter Charakteristika der drei vorgestellten Frameworks zeigt *Tabelle 2*.

Video for Windows wurde schon kurz nach Veröffentlichung für sein Treibermodell kritisiert, welches Inkonsistenzen aufweist. In der Praxis weist VFW vor allem Schwächen bei der Synchronisierung von Audio- und Videosignalen, sowie der Kontrolle von Qualitätsmerkmalen wie Helligkeit und Kontrast auf. Auch ist VFW angesichts der rasanten Entwicklung der Informationstechnologie anzumerken, dass es schon seit einigen Jahren nicht mehr weiterentwickelt wird – die Größe einer einzelnen Datei ist auf zwei GB begrenzt, ebenso ist der I/O-Durchsatz beschränkt. Für die Kommunikation mit einer Webcam fallen die genannten Schwächen jedoch kaum ins Gewicht. Inkonsistenzen im Treibermodell sind hierfür irrelevant, eine synchrone Wiedergabe von Video und Audio ist nicht notwendig und die Manipulation von Bildqualitätsmerkmalen lässt sich über Umwege erreichen. Unter dem Strich sprechen vor allem die Effizienz und Einfachheit der Programmierung für Video for Windows.

DirectX ist das mit Abstand kompletteste und effizienteste Multimedia-Framework. Neben einer wohldurchdachten und gut strukturierten Architektur sind vor allem die hohe Performance und führende Marktstellung die größten Pluspunkte. Diese Vorteile werden jedoch mit einem sehr hohen Grad an Komplexität erkauft. Ausgehend von den Anforderungen, die eine nachfolgende Bildaufbereitung an ein Framework stellt, ist die Überlegung anzustellen, ob der Mehrwert an Funktionalität und Performance von DirectX gegenüber VFW und dem JMF die höhere Komplexität rechtfertigt.

Das Java Media Framework reicht zwar nicht an die Leistungsfähigkeit von DirectX und die Effizienz von Video for Windows heran, stellt aber eine solide und qualitativ ansprechende Alternative dar. Die Plattformunabhängigkeit von Java eröffnet dem Programmierer zudem die Wahl eines anderen Betriebssystems als die der Microsoft-Familie.

4 Kommunikation zwischen Laptop und Host-Rechner

Mit Hilfe von Laptop und Webcam sind Roboter in der Lage, eigenständig Bilder ihrer Umgebung zu erstellen und zu analysieren. Bedingt durch die Komplexität der Landminendetektion ist jedoch die Kooperation mit einem Host-Rechner wünschenswert, sei es beispielsweise zum Abgleich mit einer umfangreichen Datenbank oder zur zentralen Koordination mit anderen Robotereinheiten.

4.1 Drahtlose Übertragungstechnologien

Auf Minenfeldern steht in der Regel keine Netzinfrastruktur zur Verfügung, so dass die Errichtung einer eigenen Infrastruktur unumgänglich ist. Da sich das Verlegen von Kabeln in diesem Umfeld als unmöglich erweist, wird auf drahtlose Netze zurückgegriffen, um einen Abgleich der gesammelten Daten und den Abruf neuer Aufgaben herzustellen. Im Folgenden werden die zur Zeit wichtigsten und verbreitetesten Techniken, Bluetooth und WLAN, vorgestellt und auf ihre Eignung für die vorgegebene Aufgabe hin untersucht.

4.1.1 Bluetooth

4.1.1.1 Definition und Entstehung

Im Jahr 1994 wurde vom norwegischen Unternehmen Ericsson eine Studie ins Leben gerufen, die die kabellose Verbindung von Computern, Peripherie- und anderen Endgeräten wie Mobiltelefonen zum Gegenstand hatte. Hardwareunabhängigkeit, eine preisgünstige Schnittstelle, geringer Energieverbrauch und auf Hardware-Seite integrierte Sicherheitsfunktionen galten hierfür als strenge Prämissen. Die Studie mündete 1998 in die offizielle Gründung der *Bluetooth-SIG* [17]. Im Juli 2000 wurde schließlich der Standard 1.0 offiziell vorgestellt. Im November 2003 verabschiedete die Bluetooth-SIG die Spezifikation 1.2, die mit einigen Neuerungen aufwarten konnte.[18]

Der Begriff *Bluetooth* bezieht sich auf den dänischen König Harald Blåtand, der im Jahre 983 n.Chr. die Königreiche Dänemark und Norwegen vereinte. Diese ungewöhnliche Namenswahl soll zum einen dem norwegischen Ursprung der Idee Anerkennung zollen, zum anderen soll sie das Hauptbestreben der Bluetooth-Technologie unterstreichen, die Kommunikationswege zwischen Geräten unterschiedlicher Hersteller zu „vereinen“.

---

17 Special Interest Group: IBM, Intel, Nokia und Toshiba.

18 Vgl. hierzu https://www.bluetooth.org/foundry/adopters/document/Bluetooth_Core_Specification_v1.2.

4.1.1.2 Architektur

Bis zu acht aktive Geräte bilden zusammen ein *Piconetz*, in dem ein Gerät die Rolle des Masters übernimmt und sowohl Verbindungsart und -geschwindigkeit bestimmt, als auch Steuersignale an die maximal sieben Slaves sendet. Alle Geräte in einem *Piconetz* folgen der gleichen Kanalsprung-Sequenz (dazu mehr im nächsten Abschnitt) und dem Zeittakt des Masters. Diese befinden sich in einem von mehreren möglichen Modi, welche Einfluss auf den Stromverbrauch und die Reaktionsgeschwindigkeit haben. Neben den aktiven Geräten kann ein Piconetz noch bis zu 255 Geräte im *parked*-Modus einschließen.

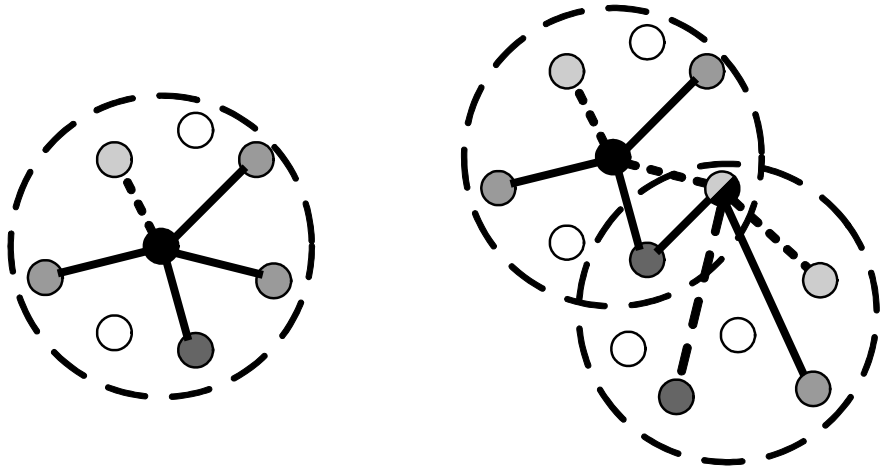

Abb. 7: Piconetz (links) und Scatternetz (rechts)

Piconetze, die sich überlappen, bilden zusammen ein *Scatternetz*. Ein Slave eines *Piconetzes*, der entweder Master oder Slave in einem anderen *Piconetz* ist, wird als Brückenknoten bezeichnet. *Pico*- und *Scatternetze* bilden sich dynamisch und autonom, so dass sich der Anwender keinerlei Gedanken um die Topologie machen muss.

4.1.1.3 Protokolle

In der Bluetooth-Technologie findet eine Vielzahl an Protokollen Anwendung, die anhand eines Protokollstapels in Schichten eingeteilt sind[19]. Die *Baseband*-Schicht stellt die physikalische Verbindung mittels Funk her und übernimmt Synchronisationsfunktionen. Das darauf aufsetzende *Link Manager Protocol (LMP)* ist für das Verbindungsmanagement und Sicherung zuständig. Das *Logical Link & Adaption Protocol (L2CAP)* verbirgt die Details der unteren beiden Schichten vor den aufsetzenden Protokollen. Das *Service Discovery Protocol (SDP)* liefert Informationen über Dienste und Verbindungen. Die weiteren Schichten sind Middleware-Protokolle: *RFComm* emuliert den seriellen PC-Port zur An-

[19] Das Schichtenmodell orientiert sich dabei jedoch nicht am ISO-OSI-Modell oder einem anderen bekannten Modell. Zur besseren Standardisierung arbeitet die IEEE an einer 802.15-Version des Protokoll-Stapels. Hierzu vgl. [Ta03, S.349f].

bindung älterer Geräte, *AT-Commands* und *TCS BIN* dienen zur Kontrolle von Telefonverbindungen.

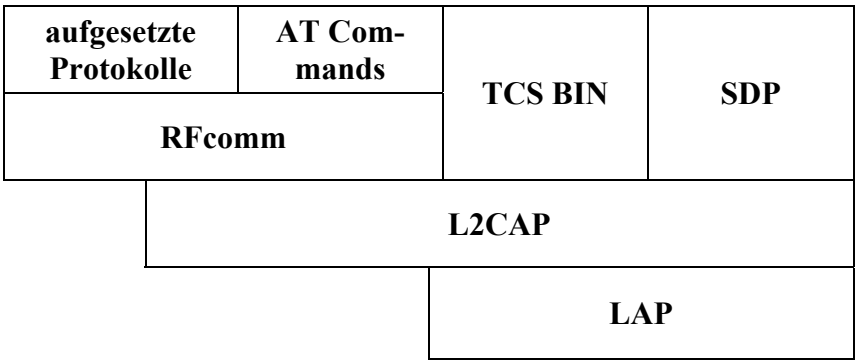

Abb. 8: Bluetooth-Protokollstapel

4.1.1.4 Bitübertragung & Medienzugriffssteuerung

Zur Funkübertragung wird das lizenzfreie ISM-Band[20], im Bereich von 2,402 - 2,48 GHz genutzt, so dass 79 Kanäle zu je 1 MHz zur Verfügung stehen[21]. Die Übertragung der Datenpakete erfolgt zeitschlitzgesteuert in Verbindung mit einem Frequenzsprungverfahren (FHSS[22]) mit bis zu 1600 Frequenzsprüngen pro Minute, welches zur Minimierung von Störungen im gleichen Frequenzband[23] dient.

Die Brutto-Übertragungsrate beträgt 1 Mbit/s. Die Netto-Übertragungsrate ist je nach Verbindungsprofil verschieden. Zur asynchronen, verbindungslosen und paketvermittelten Datenübertragung gibt es zwei Varianten: Die asymmetrische Variante bietet 723 kBit/s Download- und 57,6 kBit/s Upload-Rate; die symmetrische Variante stellt in beide Richtungen 432,6 kBit/s zur Verfügung.

Je nach Anwendungsgebiet können Bluetooth-Geräte in drei verschiedenen Sendeklassen mit unterschiedlichen Sendeleistungen betrieben werden, die maximal eine Reichweite von 100 Metern zulassen.

---

20 Industrial-, Scientific- and Medical-Band.

21 Eine Ausnahme hierbei bilden Frankreich, Spanien und Japan, in denen wegen gesetzlicher Bestimmungen nur 23 Kanäle genutzt werden.

22 Frequency Hopping Spread Spectrum, vgl Kapitel 4.1.2.1.

23 Das ISM-Band wird u.a. auch von Mikrowellengeräten und den WLAN-Standards IEEE 802.11b und IEEE 802.11g genutzt.

	Reichweite	Sendeleistung
Sendeklasse 1:	ca. 100 Meter	100mW
Sendeklasse 2:	ca. 10 Meter	2,5 mW
Sendeklasse 3:	ca. 0,1 - 10 Meter	1 mW

Tab. 3: Sendeklassen von Bluetooth-Geräten

4.1.1.5 Profile

Im Unterschied zu anderen Vernetzungstechnologien sieht Bluetooth für verschiedene Anwendungen einen anderen Protokollstapel vor, um deren jeweiligen Anforderungen gerecht werden zu können. Diese *Profile* sind im Bluetooth-Standard verbindlich definiert.

Neben den Basis-Profilen wie *GAP*[24], das beispielsweise Verfahren zur Authentifizierung und zum Verbindungsaufbau bereitstellt, gibt es spezielle Schnittstellen zu anderen Technolgien wie *PAN*[25], welches den Zugriff auf ein Ethernet erlaubt.

Nicht jedes Bluetooth-fähige Gerät unterstützt alle Profile, weshalb hierbei besonders auf Kompatibilität geachtet werden muss.

4.1.2 WLAN

Mit der Entstehung der Notebooks entwickelte sich in den neunziger Jahren das Interesse an drahtlosen Netzwerken. Realisiert wurden diese häufig über Funksender und -empfänger, welche leider nicht standardisiert und somit inkompatibel zueinander waren. Zur Vereinheitlichung verabschiedete das IEEE-Komitee 1997 den Standard 802.11 für Wireless LAN, der bis zum jetzigen Zeitpunkt sechs Varianten umfasst.

Zwei verschiedene Arten von Netzwerken für WLAN wurden realisiert, das Infrastruktur-Netzwerk und das Ad-hoc-Netzwerk.

---

[24] Generic Access Profile.
[25] Personal Area Networking Profile.

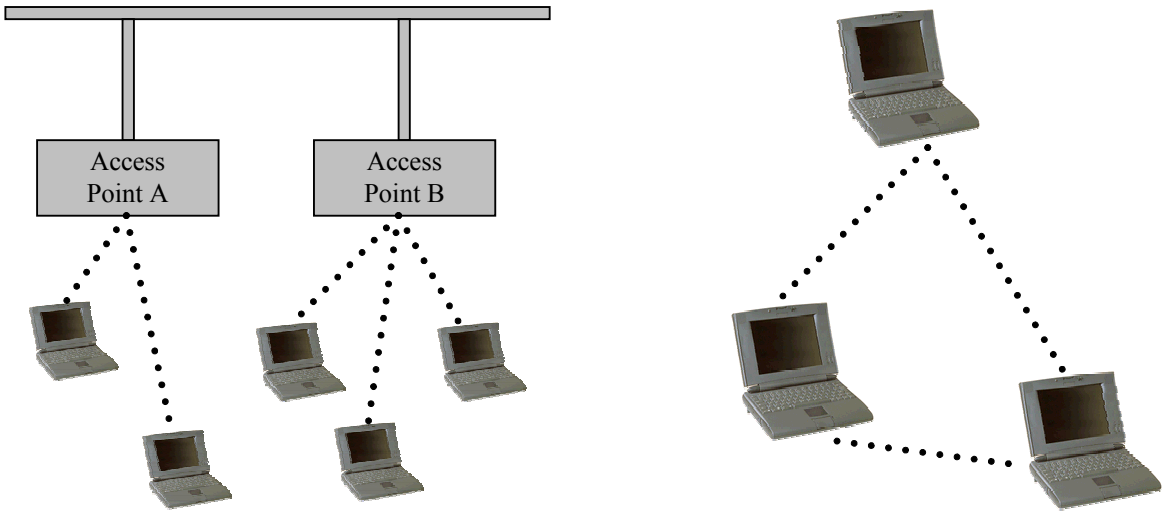

Abb. 9: Infrastruktur-Netzwerk (links) und Ad-hoc-Netzwerk (rechts)

Für unsere Aufgabe erweist sich das Ad-hoc-Netzwerk als vorteilhaft: es ist einfach, günstig und vor allem mobil, während ein Infrastruktur-Netzwerk nur für Gebäudevernetzung eingesetzt werden kann. Allerdings müssen beim Ad-hoc-Netzwerk Abstriche bei der Reichweite und der Bandbreite akzeptiert werden.

Die WLAN-Protokollarchitektur behandelt die Bitübertragungsschicht und die Sicherungsschicht des ISO-OSI-Modells[26], wobei die Sicherungsschicht laut 802.11 in MAC (Medium Access Control) und LLC (Logical Link Control) aufgeteilt wird.

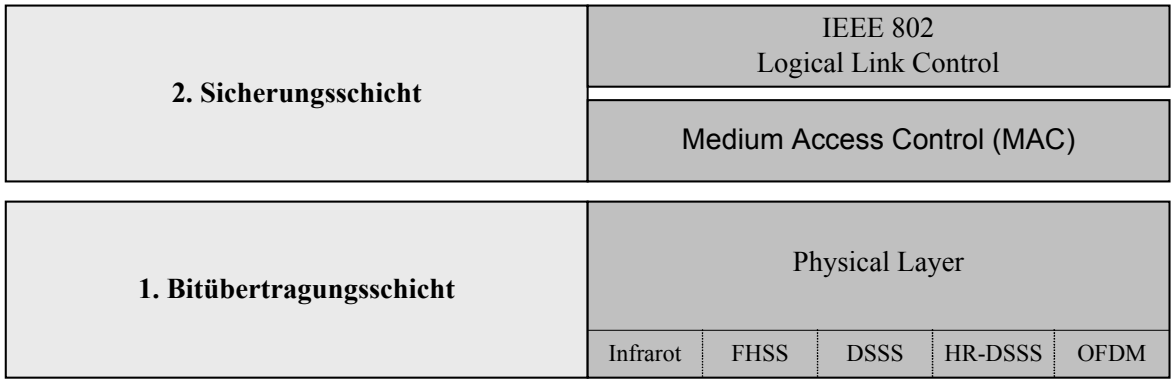

Abb. 10: Der Zusammenhang zwischen dem ISO-OSI-Modell und IEEE 802.11

4.1.2.1 Die Bitübertragungsschicht

In den sechs 802.11-Varianten werden fünf verschiedene Übertragungstechniken verwendet, die im Folgenden kurz vorgestellt werden[27].

---

[26] So wie bei allen anderen IEEE 802-Standards.

[27] Der interessierte Leser sei an dieser Stelle auf [Er02] verwiesen.

Die erste Möglichkeit basiert auf Infrarot. Hierbei können Datenübertragungsraten von 1 Mbit/s und 2 Mbit/s verwendet werden. Für Freiluftanwendungen ist diese Technik allerdings nicht von Interesse, da neben der niedrigen Geschwindigkeit die Signale durch Sonnenlicht beeinflusst werden.

Der zweite WLAN-Standard mit niedriger Geschwindigkeit verwendet FHSS (Frequency Hopping Spread Spectrum), welches auch bei Bluetooth eingesetzt wird. Dabei handelt es sich um ein einfaches und energiesparendes System im 2,4-GHz-Band. 79 Kanäle mit 1 MHz Breite werden bei diesem Verfahren gebraucht, um mit Hilfe einer pseudo-zufälligen Hopping-Sequenz die Anzahl der Kollisionen zu minimieren. Stationen, die dieses benutzen, müssen die gleichen Startparameter verwenden und die ganze Zeit über synchronisiert sein. Ein Vorteil von FHSS ist eine erhöhte Abhörsicherheit, da die Startparameter nach außen hin unbekannt sind und ständig wechseln.

Eine weitere Methode ist DSSS (Direct Sequence Spread Spectrum). Sie arbeitet mit einer Signalspreizung durch einen Pseudonoise-Code, dem so genannten Chipping. Durch diese Spreizung wird aus jedem Bit eine 11-stellige Chip-Sequenz erstellt, die selbst bei teilweiser Zerstörung das originäre Bit erkennen lässt. Nachteilig ist allerdings, wie bei den beiden zuvor behandelten Optionen, die geringe Bandbreite von 1 oder 2 Mbit/s.

Anders als bei den vorherigen Techniken lassen sich mit der Erweiterung HR-DSSS (High Rate DSSS) Geschwindigkeiten von bis zu 11 Mbit/s erzielen. Dieses Verfahren verwendet ebenfalls das lizenzfreie 2,4-GHz-ISM-Band und ist bei niedrigen Übertragungsraten kompatibel zu DSSS.

Die letzte verwendete Übertragungstechnik ist OFDM (Orthogonal Frequency Division Multiplexing). Diese Methode ist als einzige kompatibel zum HiperLAN/2-System, durch Nutzung des 5-GHz-ISM-Bands aber nicht kombinierbar mit anderen 802.11-Standards. Gearbeitet wird bei OFDM mit mehreren Frequenzen, auf denen gleichzeitig Übertragungen stattfinden. Der große Vorteil dieses Verfahrens ist die hohe Bandbreite von 6 - 54 Mbit/s.

4.1.2.2 Die Medium Access Control

Mit Hilfe von Kollisionserkennung, Kollisionsvermeidung[28] und Reservierung sichert das MAC-Teilschicht-Protokoll eine korrekte Übertragung von Datenrahmen. Benötigt wird die MAC, um die drei im folgenden vorgestellten Probleme zu lösen, welche in drahtlosen Netzen auftreten können.

---

[28] Zu diesem Zweck werden den Stationen die jeweiligen Kanäle zugewiesen und die Übertragungsreihenfolge festgelegt.

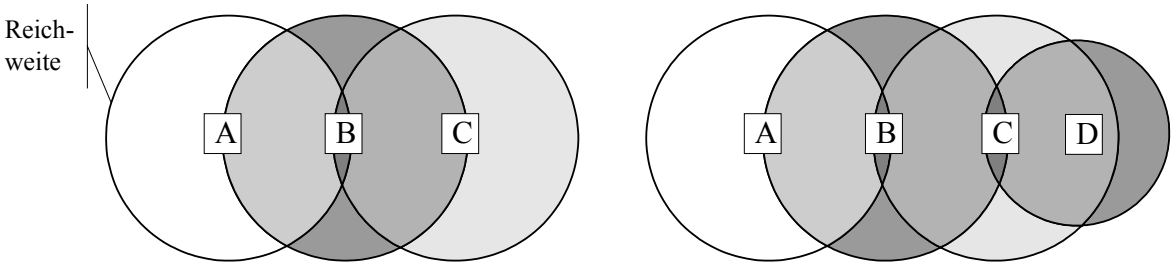

Abb. 11: Das Hidden Station Problem (links) und das Exposed Station Problem

Das erste Problem, das „Hidden Station Problem“, tritt auf, wenn Stationen zu weit voneinander entfernt sind. Sendet laut obigem Beispiel Station A an B, kann C dieses nicht hören, da A sich außerhalb der Reichweite von C befindet. C denkt nun, dass sie mit der Übertragung an B beginnen kann und verursacht somit Überlagerungen bei Station B, d.h. der Rahmen von A wird zerstört, da diese Station für C versteckt ist.

Ist eine Station einer anderen ausgeliefert, spricht man vom „Exposed Station Problem“. Hier sendet beispielsweise B einen Rahmen an Station A und C möchte an D senden. Das Problem besteht in diesem Fall darin, dass C mit der Übertragung wartet, weil ein besetztes Medium signalisiert wird, obwohl die Rahmen nur zwischen B und C kollidieren und zerstört würden. Somit geht für C Zeit verloren.

Die letzte Schwierigkeit stellt das „Near/Far Problem“ dar, welches bei stark unterschiedlichen Entfernungen zwischen den Stationen auftritt[29]. Da die Signalstärke quadratisch mit der Entfernung abnimmt, übertönt das Signal der näheren Station das der weiter entfernten, falls beide senden.

Behandelt werden diese Probleme mit CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Hierbei wird zuerst ein RTS-Rahmen (Request To Send) von dem Sender an den Empfänger übertragen, welcher bei Einverständnis mit einem CTS-Rahmen (Clear To Send) antwortet. Erhält der Sender die Antwort, beginnt er mit der Übertragung des Rahmens und startet einen ACK-Timer (Acknowledgement-Timer). Sobald der Empfänger den gesamten Datenrahmen erhalten hat, sendet er einen ACK-Rahmen zurück. Erhält der Sender diesen Rahmen vor Ablauf des Timers, dann war der Datenaustausch erfolgreich, ansonsten wird das Protokoll erneut ausgeführt. Damit andere Stationen diese Übertragung nicht stören, werden sie, falls sie ein RTS und ein CTS oder nur den CTS-Rahmen von anderen Stationen erhalten, in einen Ruhezustand (NAV, Network Allocation Vector) geschaltet, bis sie den ACK-Rahmen erhalten. Um zu verhindern, dass bei geringfügiger Zerstörung eines Rahmens immer wieder ein großer Rahmen übermittelt werden

[29] Das Near/Far Problem bleibt in der Literatur aufgrund geringerer Relevanz häufig unerwähnt, hier aber der Vollständigkeit halber kurz erläutert.

muss, wird häufig ein sogenannter Fragment-Burst eingesetzt. Hier werden die langen Rahmen in kürzere Segmente aufgeteilt, die nacheinander übertragen werden.

4.1.2.3 Die logische Verbindungssteuerung

Die LLC (Logical Link Control) setzt direkt auf der MAC-Teilschicht auf und befindet sich somit ebenfalls auf der OSI-Sicherungsschicht. Sie dient als Schnittstelle zur Vermittlungsschicht und verbirgt durch ein einheitliches Format alle Unterschiede zwischen den verschiedenen IEEE 802-Netzen (z.B. Ethernet, WLAN oder Bluetooth).

4.1.2.4 IEEE 802.11-Standards

Das Institute of Electrical and Electronics Engineers hat inzwischen sechs verschiedene 802.11-Varianten freigegeben. Dabei handelt es sich jeweils um drei mit hoher und drei mit niedriger Geschwindigkeit. Die drei Hochgeschwindigkeits-LANs sind auf dem Markt erhältlich und unterscheiden sich in nennenswerten Merkmalen.

1999 wurde die Version 802.11a vorgestellt. Sie arbeitet mit OFDM im 5 GHz-ISM-Band und stellt somit eine Bitrate von 54 Mbit/s zur Verfügung. Allerdings wird diese Frequenz auch von Radaranlagen benutzt, weshalb 802.11a nur für den Indoorgebrauch zugelassen ist.

Der zweite Standard, 802.11b, ist im gleichen Jahr veröffentlicht worden, verwendet aber eine andere Technik. 802.11b gebraucht HR-DSSS im lizenzfreien 2,4-GHz-ISM-Band und erreicht deshalb nur eine geringere Bandbreite. Da dieser Standard aber auch für den Outdoorgebrauch zugelassen ist, ist er am weitesten verbreitet.

Eine Kombination dieser beiden Varianten, 802.11g, wurde 2001 freigegeben. Hier wird die Technik von 802.11a (OFDM) benutzt, aber in dem gleichen Frequenzband mit 802.11b gearbeitet. Diese Variante ist somit abwärtskompatibel zu 802.11b und setzt sich dank der höheren Bandbreite zunehmend durch.

4.1.3 Bluetooth oder WLAN?

Nach der Vorstellung und Erläuterung der beiden am weitesten verbreiteten Technologien zur drahtlosen Übertragung von Daten, soll schließlich aufgezeigt werden, welche unserem Erkenntnisinteresse dient.

Die Bluetooth-Technologie kann mit überzeugenden Argumenten aufwarten. Neben geringen Hardwarekosten und niedrigem Stromverbrauch besticht die Technologie vor allem durch hohe Störungsresistenz, Sicherheit und Simplizität. Allerdings war es nie als Netzwerkersatz gedacht, sondern nur zur drahtlosen Anbindung von Peripherie über Kurzstre-

cken. Infolgedessen muss sich Bluetooth in den Punkten Bandbreite und Reichweite eindeutig anderen Funktechniken geschlagen geben. So ist für unsere Aufgabe, die Kommunikation zwischen Robotern und einem Host-Rechner, Bluetooth nicht zu gebrauchen, da unebenes Gelände die Reichweite sehr negativ beeinflusst. Ebenfalls wird die Übertragungsrate bei zunehmender Entfernung viel stärker gemindert als bei WLAN.

Schon früh wurden optimistische Prognosen über Bluetooth gestellt, die eine rasante Verbreitung vorhersagten. Bisher konnten diese nur in Fernost[30] erfüllt werden, aber gerade die breite Unterstützung, derer sich Bluetooth rühmen kann[31], scheint ein Indiz dafür zu sein, dass die Technologie sich in absehbarer Zeit weltweit durchsetzen wird.

Da WLAN für anspruchsvollere Aufgaben Vorteile betreffend der Reichweite, der Bandbreite und der Störanfälligkeit bietet, ist diese Technologie für unsere Zwecke nützlicher. Um die Kommunikation unter realistischen Verhältnissen zu garantieren, ist zur Zeit WLAN die beste Lösung. Ein kombinierter Einsatz beider Möglichkeiten ist nicht zu empfehlen, da beide das gleiche Frequenzband benutzen und somit vermehrte Störungen hervorrufen würden.

4.2 Softwareprotokoll

Für den Austausch von Informationen zwischen Roboter und Host-Rechner wird, aufbauend auf den vorgestellten drahtlosen Übertragungstechnologien, ein Software-Protokoll benötigt.

4.2.1 TCP/IP

Zur Übermittlung der Daten vom Sender zum Empfänger soll das TCP/IP-Protokoll eingesetzt werden. Entsprechend seinem Namen bilden das "Transmission Control Protocol" (TCP) und das "Internet Protocol" (IP) den Kern von TCP/IP. Hauptaufgaben des Internet Protocols, das sich im ISO/OSI-Modell auf der Vermittlungsschicht einordnen lässt, sind die Adressierung von Rechnern sowie das Fragmentieren bzw. Reassemblieren von Datenpaketen. Das Transmission Control Protocol, im ISO/OSI-Modell auf der Transportschicht liegend, stellt den sicheren und zuverlässigen Transport der Daten durch das Netzwerk sicher und bietet Fehlererkennung und -korrektur auf dem gesamten Übertragungsweg. Aus programmiertechnischer Sicht sind genaue Kenntnisse über die Einzelheiten von TCP/IP bzw. den unteren Schichten der Protokollhierarchie nicht von Bedeutung, da Betriebssysteme oder Programmiersprachen die Details der Netzwerk-Kommunikation in

---

30 Dortiger Verkaufsschlager ist eine „intelligente Toilette“, die via Bluetooth die Stuhlgang-Messwerte des Benutzers auf dessen PDA überträgt.

31 Die Bluetooth-SIG ist mit über 2500 Mitgliedern eine der größten Organisationen ihrer Art.

Form von Sockets kapseln, wodurch sich der Programmierer auf die Realisierung der Anwendung konzentrieren kann.

4.2.2 Robot Communication Markup Language

Zur Kodierung der auszutauschenden Nachrichten wird mit Hilfe von XML (Extensible Markup Language) eine als "Robot Communication Markup Language" bezeichnete Markup-Sprache definiert. Die bedeutendsten Vorteile von XML liegen in der universellen Lesbarkeit, die einen systemunabhängigen Datenaustausch ermöglicht, der leicht verständlichen Syntax und der guten Erweiterbarkeit. Der Aufbau der XML-Dokumente ist durch die in Anhang C aufgelistete Document Type Definition (DTD) festgelegt. Die DTD ermöglicht die syntaktische Validierung der übertragenden XML-Strukturen.

5 Konstruktion eines Suchroboters

Im folgenden Kapitel werden die bis hierher beschriebenen theoretischen Inhalte in die Praxis umgesetzt. Dazu wird zunächst aus den LEGO-Teilen zweier Robotics Invention Systems eine Plattform für den Transport von Laptop und Webcam konstruiert und anschließend eine Software zur Steuerung des Roboters entwickelt. Um an die Bilder der installierten Webcam zu gelangen, werden die in Abschnitt 3.1.2 beschriebenen VfW-Funktionen verwendet. Die Motoren des Roboters werden, entsprechend Kapitel 2, direkt vom Laptop aus über den fest installierten Infrarot-Tower gesteuert.

5.1 LEGO-Konstruktion

Um das, im Vergleich zur LEGO-Konstruktion, relativ hohe Gewicht des Laptops tragen zu können, werden für den Antrieb des Roboters insgesamt vier Motoren eingesetzt, die sich auf vier Ketten verteilen. Aufgrund der Beschränkung von maximal drei Motoren pro RCX macht dies die Verwendung eines zweiten RCX notwendig.

Abb. 12: Roboter samt Laptop, Webcam & IR-Tower

Da das Kommunikationsprotokoll des RCX nicht verbindungsorientiert ist, kann die Steuerung der beiden RCX-Einheiten über einen einzigen Infrarot-Tower erfolgen. Hierbei ergibt sich jedoch das Problem, dass sich die per Infrarot gesendeten Antworten der RCX überlagern, was die Auswertung der zurückgeschickten Nachrichten unmöglich macht. Somit lässt sich nicht überprüfen, ob die Nachrichten auch tatsächlich vom RCX empfangen wurden, und auch die Abfrage von Statusdaten, wie beispielsweise dem Batteriestand der Einheiten, ist nicht durchführbar. Für den hier vorgestellten Einsatz hat diese Einschränkung jedoch keine große Bedeutung.

5.2 BallFinder-Software

Aufgabe der als "BallFinder" bezeichneten Software ist das Auffinden eines roten Balls. Dazu wertet das Programm die mit Hilfe von Video for Windows aufgezeichneten Bilder der Webcam aus und sendet entsprechend der gewonnenen Informationen Befehle an die beiden RCX des Roboters. Konkret dreht die Software den Roboter schrittweise solange, bis ein roter Ball mittig im Bild auftaucht und lässt den Roboter dann, basierend auf den Daten der in Abschnitt 5.2.1.2 erläuterten Distanzmessung, geradeaus fahren. Die Programmierung von "BallFinder" erfolgt mit Borland Delphi[32].

5.2.1 Bildauswertung

Um zu erfahren, ob das von der Kamera gewonnene Bild einen roten Ball enthält, wird es der im folgenden Abschnitt erläuterten Objekterkennung unterzogen, die im Erfolgsfall die Pixel-Koordinaten und die Größe des Balls zurückliefert. Anschließend werden diese Daten mittels eines Algorithmus zur Distanzmessung in Maße der realen Welt übertragen.

5.2.1.1 Objekterkennung

Die Farben der einzelnen Pixel im Bild stehen in Form von RGB-Tripeln zur Verfügung, deren Elemente den Anteil der Farben Rot (R), Grün (G) und Blau (B) an dem jeweiligen Pixel repräsentieren. Ein Tripel besteht aus drei 8-Bit-Integer-Zahlen, mit einem jeweiligen Wertebereich von 0 bis 255.

Das RGB-Farbmodell entspricht der Funktionsweise von Farbmonitoren, die Farben durch additive Mischung von rotem, grünem und blauem Licht erzeugen. Dieses Modell eignet sich jedoch nur bedingt zur Farbklassifikation von Pixeln, da die Farbwerte sich teilweise sprunghaft ändern, obwohl die Farbe subjektiv die gleiche bleibt.

Aus diesem Grund werden die einzelnen Farbwerte vom RGB-Farbmodell in den HLS-Farbraum transformiert. Im HLS-Modell, auch bekannt unter der Abkürzung HSL, wird jede Farbe durch die Angabe des Farbtons (Hue), der Helligkeit (Luminance) und der Sättigung (Saturation) beschrieben. Dieses Modell entspricht in etwa der menschlichen Wahrnehmung von Farben und lässt sich beispielsweise durch die Vorgehensweise eines Malers darstellen, der zur Farbmischung zunächst ein reines Pigment (Farbton) verwendet, weiß hinzufügt (Sättigung) und schließlich schwarz ergänzt (Helligkeit). Im HLS-System wird der Farbton in der Regel in Grad als Wert zwischen 0 und 360 und sowohl Helligkeit als auch Sättigung in Prozent (0 bis 100) angegeben. Zur einfacheren Verarbeitung werden die

---

[32] Vgl. http://www.borland.de/delphi/index.html.

drei Werte hier jedoch in Zahlen zwischen 0 und 255 umgewandelt. Ein Ansatzpunkt zur Konvertierung von Farbwerten vom RGB-Farbmodell in den HLS-Farbraum findet sich in [FD82, S. 618 f.].

Anhand der HLS-Werte lassen sich die Pixel nach Farben klassifizieren. In zahlreichen Tests hat sich gezeigt, dass Rotwerte einem Farbton (Hue) zwischen 0 und 14 bzw. 241 und 255 entsprechen. Die Helligkeit sollte, abhängig vom jeweiligen Farbton, im Bereich 60 bis 210 liegen und die Sättigung einen Wert zwischen 80 und 255 annehmen, um subjektiv als rot empfunden zu werden.

Mit Hilfe dieser Werte werden in dem von der Kamera aufgenommenen Bild alle roten Pixel schwarz markiert und die restlichen Bereiche weiß eingefärbt. Um den Suchraum für die nachfolgenden Algorithmen zu verkleinern und so eine akzeptable Geschwindigkeit zu ermöglichen, werden anschließend zwei Optimierungsschritte durchgeführt. Zunächst erfolgt eine Skalierung des Bildes auf 40 Prozent der Originalgröße, was einer Reduzierung der Pixelzahl um 84 Prozent entspricht. Weiterhin werden Bereiche im Bild herausgesucht, in denen sich möglicherweise Kreise befinden. Diese Kandidatengenerierung erfolgt, indem mit Hilfe eines Füllalgorithmus zusammenhängende Flächen gesucht werden und anschließend überprüft wird, ob die horizontale Ausdehnung der jeweiligen Fläche in etwa der vertikalen Ausdehnung entspricht (eine Abweichung von 30 Prozent wird toleriert) und die Füllung der Fläche annäherungsweise mit der Fläche eines Kreises übereinstimmt. Im nächsten Schritt wird ein Filter zur Kantendetektion auf das Bild angewendet. Der entsprechende Algorithmus geht davon aus, dass neben einem Punkt im Bild weitere Punkte in einer bestimmten Anordnung vorhanden sein müssen, damit dieser Pixel zu einer Kante gehört. Um dies festzustellen, wird die Farbe des Punktes neu berechnet, indem die Farbwerte des Pixels und seiner direkten Nachbarn jeweils mit den entsprechenden Zellen einer 3x3-Matrix multipliziert und die Produkte aufaddiert werden.

Nach diesen Aufbereitungsschritten beginnt die eigentliche Kreiserkennung auf den Bereichen, die zuvor als Kandidaten für mögliche Kreise lokalisiert wurden. Die Erkennung erfolgt unter Verwendung der so genannten Hough-Transformation, einem ursprünglich im Jahr 1962 von P. V. C. Hough zur Geradendetektion entwickelten mathematischen Algorithmus.

Die hier verwendete Variante zur Kreiserkennung berechnet basierend auf der Kreisgleichung $(x - x_{M;})^2 + (y - y_M)^2 = r^2$ zu jedem Kantenpunkt im Bild die möglichen Mittelpunkte und trägt diese in den Akkumulator ein. Der Akkumulator, auch Hough-Raum genannt, ist in diesem Fall ein dreidimensionales Array aus Integer-Variablen, das zu jeden Punkt und Radius die Anzahl der für diese Position bestimmten Mittelpunkte aufnimmt. Da der Radius des zu suchenden Kreises im vorhinein nicht bekannt ist, wird der Algo-

rithmus auf alle Radien zwischen 2 und 50 Pixeln (im auf 128 mal 96 Pixel verkleinerten Bild) angewendet. Im Anschluss wird aus dem Akkumulator-Array der maximale Wert, d.h. die höchste Anzahl an Schnittpunkten, herausgesucht und daraus der Mittelpunkt und Radius des gefundenen Kreises abgelesen. Um Position und Radius auf das Originalbild beziehen zu können, werden die Daten abschließend noch mit 2,5 multipliziert.

5.2.1.2 Distanzmessung

Zur Berechnung der Distanz zwischen Kamera und Ball wird das in Abbildung 13 dargestellte approximative Modell verwendet.

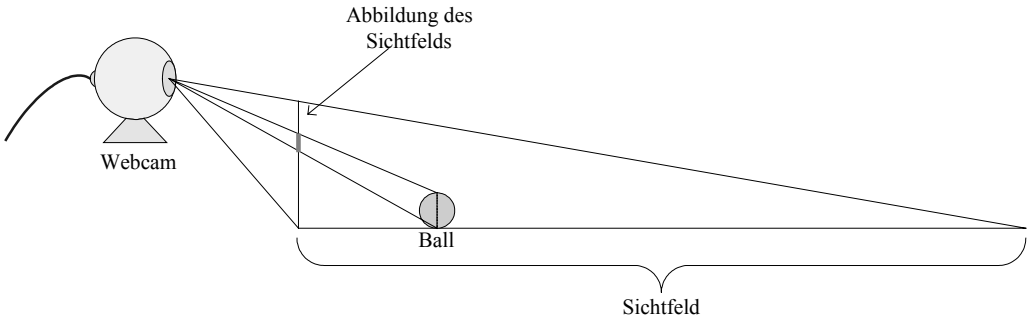

Abb. 13: Modell zur Distanzmessung

Die eingesetzte Webcam ist so auf dem Roboter montiert, dass sie von oben herab nach vorne schaut. Dadurch lässt sich über die vertikale Position im aufgenommenen Bild die Entfernung eines Gegenstandes ablesen bzw. berechnen. Dazu müssen vorher jedoch drei Werte gemessen werden. Dies ist zum einen die Höhe der Kamera (y_{Cam}), d.h. die Entfernung der Kameralinse vom Boden, dann die minimal im Bild sichtbare Entfernung von der Kamera (x_{Min}) und schließlich die an dieser Stelle maximal sichtbare Höhe (y_{Max}). Diese Kalibrierungsdaten sollten jeweils in Zentimeter angegeben werden.

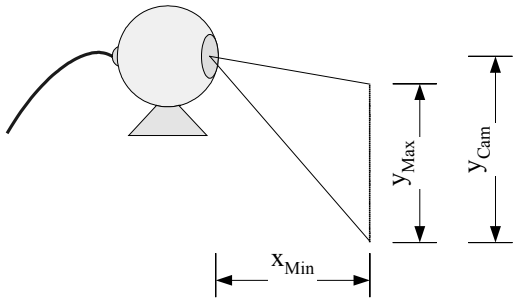

Abb. 14: Benötigte Maße

Die aus dem Bild der Kamera abgelesene vertikale Pixelposition des Gegenstands wird über die folgende Formel in Zentimeter umgerechnet, wobei der Wert 240 für die vertikale Auflösung des Bildes steht:

$$y_{Abb} = \frac{y_{Max}}{240} \cdot Pixelposition\ im\ Bild$$

Da davon ausgegangen wird, dass der Gegenstand direkt auf dem Boden liegt, lässt sich seine Position durch den Punkt $(x_{Ball},0)$ beschreiben. Auf der Linie von der Kameralinse aus bis hin zum Gegenstand ist, neben der Kameraposition $(0,y_{Cam})$ selbst, auch der soeben berechnete Punkt (x_{Min},y_{Abb}) bekannt. Somit lässt sich die Entfernung x_{Min} über die Zwei-Punkte-Form der Geradegleichung berechnen:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Durch Einsetzen ergibt sich:

$$0 = y_{Cam} + \frac{y_{Min} - y_{Cam}}{x_{Min}} x_{Ball}$$

$$\Leftrightarrow \quad x_{Ball} = \frac{x_{Min} \cdot y_{Cam}}{y_{Cam} - y_{Abb}}$$

Da das zugrundeliegende Modell auf approximativen Annahmen beruht und eine präzise Messung der benötigten Inputdaten in der Praxis nahezu unmöglich ist, kann die Rechnung nur eine ungefähre, für unsere Zwecke jedoch völlig ausreichende, Schätzung der Distanz liefern.

6 Zusammenfassung/Fazit/Ausblick

In den vorangegangenen Kapiteln wurden die bei einem Roboter zur Objekterkennung betroffenen Kommunikationsschnittstellen beschrieben, ohne dabei den Praxisbezug aus den Augen zu verlieren. Zu jeder Schnittstelle wurden Lösungsmöglichkeiten aufgezeigt und diese hinsichtlich ihrer Eignung untersucht. Es hat sich gezeigt, dass zum Bau eines Roboters Kenntnisse in vielen Bereichen von Nöten sind und die Wahl des zu verwendenden Protokolls eine äußerst schwierige Entscheidung ist.

Beim Bau eines Roboters auf Grundlage des LEGO Robotics Invention Systems 2.0 ergaben sich insbesondere Probleme durch das Gewicht des Laptops, das lediglich durch den Einsatz von zwei RCX-Einheiten getragen werden konnte. Für zukünftige Projekte wäre daher die Verwendung eines leichten Handhelds überlegenswert, der unter Umgehung des LEGO Towers direkt über die integrierte Infrarot-Schnittstelle mit dem RCX kommuniziert. Die Lösung des Problems der Inkompatibilität zwischen dem von LEGO verwendeten Infrarot-Protokoll und dem IrDA-Standard würde sicherlich genügend Stoff für eine eigene Ausarbeitung liefern. Weiterhin wurde bei der im fünften Kapitel beschriebenen Roboter-Konstruktion die Koordination mit einem Host-Rechner oder mit anderen Robotern außer Acht gelassen. Aufbauend auf dem in Kapitel 4 vorgestellten Kommunikationsprotokoll ließe sich hier mit der Entwicklung eines zur Landminendetektion konzipierten Multi-Agenten-System (MAS) anknüpfen.

Abschließend soll nicht unerwähnt bleiben, dass der in Kapitel 5 konstruierte Roboter selbstverständlich für den praktischen Einsatz zur Suche von Landminen absolut ungeeignet ist[33], sondern lediglich der Demonstration der notwendigen Grundlagen dienen soll.

[33] Die Wahrscheinlichkeit der Existenz von gut sichtbaren roten Landminen in der Größe eines Tischtennisballs geht in der Praxis wohl gegen Null.

A rcxtower.cpp

```
// ***********************************************************
// Kleines Beispielprogramm, das über einen LEGO USB-Tower den
// Ping-Befehl an einen RCX sendet und die empfangenen Bytes
// in der Konsole anzeigt. Der Code ist so einfach wie möglich
// gehalten und beinhaltet keinerlei Maßnahmen zur Fehler-
// behandlung oder -korrektur.
//
// Copyright (c) 2003 by Andreas Wulf
// ***********************************************************

// rcxtower.cpp

#include <windows.h>
#include <stdio.h>

void main(void)
{
  HANDLE TowerHandle;

  /* Kommunikation mit Tower 1 (lesender und schreibender
     Zugriff) herstellen */
  TowerHandle = CreateFile("\\\\.\\LEGOTOWER1",
    GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, 0);

  /* Array mit den zu sendenden Bytes (Ping-Befehl), das später
     auch für den Empfang verwendet wird */
  unsigned char Buf[255] =
    {0x55, 0xff, 0x00, 0x10, 0xef, 0x10, 0xef};

  /* Anzahl der tatsächlich geschriebenen bzw. gelesenen
     Datenbytes */
  unsigned long ByteCount;

  /* Sende 7 Bytes aus dem Puffer an den RCX */
  WriteFile(TowerHandle, Buf, 7, &ByteCount, 0);

  /* Empfange Antwort mit bis zu 255 Zeichen */
  ReadFile(TowerHandle, Buf, 255, &ByteCount, 0);

  /* Kommunikation mit Tower beenden bzw. Handle freigeben */
  CloseHandle(TowerHandle);

  /* formatierte Ausgabe der empfangenen Daten; im Erfolgsfall
     sollte "Return: 0x55 0xff 0x00 0xef 0x10 0xef 0x10"
     ausgegeben werden */
  unsigned long I;
  printf("Return: ");
  for (I = 0; I < ByteCount; I++) {
    printf("0x%02x ", Buf[I] & 0xff);
  }
}
```

B AVICap-Makros

`capDriverGetCaps:`	Abfrage der Fähigkeiten des Aufnahmetreibers (z.B. Overlay-Fähigkeit)
`capGetStatus:`	Ermittlung des Status des Aufnahmefensters (z.B. Dimensionen)
`capGetVideoFormatSize,` `capGetVideoFormat,` `capSetVideoFormat:`	Makros zur Ermittlung und Einstellung des Videoformats (z.B. Farbtiefe)
`capDlgVideoSource,` `capDlgVideoFormat,` `capDlgVideoDisplay:`	Anzeige von Dialogfenstern zur Konfiguration der Videoaufzeichnung
`capSetCallbackOnError,` `capSetCallbackOnStatus,` `capSetCallbackOnVideoStream,` `capSetCallbackOnFrame:`	Makros zur Registrierung von Callback-Funktionen, die bei bestimmten Ereignissen aufgerufen werden
`capGrabFrame:`	Abfrage und Darstellung eines Einzelbildes Bei Aufruf werden die Bilddaten an die mit capSetCallbackOnFrame registrierte Callback-Funktion geschickt
`capDriverDisconnect:`	Trennen der Verbindung zwischen Aufnahmefenster und Videotreiber

C rcml.dtd

```
<!ELEMENT rcml (header, (cmd|reply))>

<!ELEMENT header (sender, receiver, replyto?, uuid)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT receiver (#PCDATA)>
<!ELEMENT replyto (#PCDATA)>
<!ELEMENT uuid (#PCDATA)>

<!ELEMENT cmd (runmotor|stopmotor|setmotordirection|setmotorpower|
getsensordata|getbatterylevel|getwebcamimage|sendbytecode|ping|sta
rtprogram|transferprogram|powerdown)>
<!ELEMENT runmotor EMPTY>
  <!ATTLIST runmotor motor (Motor_A|Motor_B|Motor_C) #REQUIRED
duration CDATA #REQUIRED>
<!ELEMENT stopmotor EMPTY>
  <!ATTLIST stopmotor motor (Motor_A|Motor_B|Motor_C) #REQUIRED>
<!ELEMENT setmotordirection EMPTY>
  <!ATTLIST setmotordirection motor (Motor_A|Motor_B|Motor_C)
#REQUIRED direction (Forwards|Backwards|Toggle) #REQUIRED>
<!ELEMENT setmotorpower EMPTY>
  <!ATTLIST setmotorpower motor (Motor_A|Motor_B|Motor_C)
#REQUIRED power (0|1|2|3|4|5|6|7) #REQUIRED>
<!ELEMENT getsensordata EMPTY>
  <!ATTLIST getsensordata sensor (Sensor_0|Sensor_1|Sensor_2)
#REQUIRED>
<!ELEMENT getbatterylevel EMPTY>
  <!ATTLIST getbatterylevel device (RCX|Laptop) #REQUIRED>
<!ELEMENT getwebcamimage EMPTY>
  <!ATTLIST getwebcamimage mode (Raw|Processed) #REQUIRED>
<!ELEMENT sendbytecode (#PCDATA)>
<!ELEMENT ping EMPTY>
<!ELEMENT startprogram EMPTY>
  <!ATTLIST startprogram program (0|1|2|3|4) #REQUIRED>
<!ELEMENT transferprogram (#PCDATA)>
  <!ATTLIST transferprogram program (0|1|2|3|4) #REQUIRED>
<!ELEMENT powerdown EMPTY>

<!ELEMENT reply (error|sensordata|bytecodereply|batterylevel|
pingreply|webcamimage)>
<!ELEMENT error (#PCDATA)>
<!ELEMENT sensordata (#PCDATA)>
  <!ATTLIST sensordata sensor (Sensor_0|Sensor_1|Sensor_2)
#REQUIRED>
<!ELEMENT bytecodereply (#PCDATA)>
<!ELEMENT batterylevel (#PCDATA)>
  <!ATTLIST batterylevel device (RCX|Laptop) #REQUIRED>
<!ELEMENT pingreply EMPTY>
<!ELEMENT webcamimage (imgdata, objdata?)>
<!ELEMENT imgdata (#PCDATA)>
  <!ATTLIST imgdata format (BMP|JPG) #REQUIRED>
<!ELEMENT objdata (distance, center, radius)>
<!ELEMENT distance (#PCDATA)>
<!ELEMENT center (#PCDATA)>
<!ELEMENT radius (#PCDATA)>
```

Literaturverzeichnis

[BD98] Inside DirectX: Bradley Bargen, Peter Donnelly, Microsoft Press Deutschland, 1998

[BN03] Kai Bruns, Benjamin Neidhold: Audio-, Video- und Grafikprogrammierung, Hanser Fachbuchverlag, 2003

[Er02] Mustafa Ergen: IEEE 802.11 Tutorial, 2002, http://www.eecs.berkeley.edu/~ergen/docs/ieee.pdf, Abrufdatum: 2003-11-07

[Fa03] Andrea Fasano: Using sockets, 2003, http://www.bridgespublishing.com/articles/issues/0004/Using_sockets.htm, Abrufdatum: 2003-12-11

[FD82] James D. Foley, Andries van Dam: Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1982

[HS01] Paul J.M. Havinga, Gerard J.M. Smit: Energy-efficient wireless networking for multimedia applications, 2001, http://wwwhome.cs.utwente.nl/~havinga/papers/eewn.pdf, Abrufdatum: 2003-09-18

[Ir03] Infrared Data Association, IrDA Standards, 2003, http://www.irda.org/standards/standards.asp, Abrufdatum: 2003-10-29

[Le00] LEGO Company: RCX 2.0 Firmware Command Overview, In: LEGO MINDSTORMS SDK 2.0, 2000, http://mindstorms.lego.com/sdk2/default.asp, Abrufdatum 2003-10-15

[Mi03] Microsoft: Video for Windows, In: MSDN Library, 2003, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_video_for_windows.asp, Abrufdatum: 2003-12-12

[NMG01] Edgar Nett, Michael Mock, Martin Gergeleit: Das drahtlose Ethernet, Addison-Wesley, 2001

[Pr99] Kekoa Proudfoot: RCX Opcode Reference, 1999, http://graphics.stanford.edu/~kekoa/rcx/opcodes.html, Abrufdatum: 2003-12-08

[Ta03] Andrew S. Tanenbaum: Computernetzwerke, Pearson Studium, 2003

[Wi03] Wikipedia: Internet-Protokoll-Familie, 2003, http://de.wikipedia.org/wiki/TCP/IP-Referenzmodell, Abrufdatum: 2003-12-01

[ZT98] Arkady Zaslavsky, Zahir Tari: Mobile Computing: Overview and Current Status, 1998, http://goanna.cs.rmit.edu.au/~zahirt/Papers/acj.pdf, Abrufdatum: 2003-09-18