# PROJEKTPRAKTIKUM ROVIO

eingereichter
Projektpraktikumsbericht
von

Dipl. Ing (FH) Jochen Guck

geb. am 05.05.1985
wohnhaft in:
Carl-Orff-Weg 16
82110 Germering
Tel.: 0172 6568486

Lehrstuhl für
STEUERUNGS- UND REGELUNGSTECHNIK
Technische Universität München
Univ.-Prof. Dr.-Ing../Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

| | |
|---|---|
| Beginn: | 19.04.2010 |
| Abgabe: | 02.07.2010 |

TECHNISCHE UNIVERSITÄT MÜNCHEN
**LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK**
ORDINARIUS: UNIV.-PROF. DR.-ING./UNIV. TOKIO MARTIN BUSS
EXTRAORDINARIA: UNIV.-PROF. DR.-ING. SANDRA HIRCHE

3.4.2010

PRACTICAL COURSE
for
Jochen Guck, Mat.-Nr. 03614445

**Development of software and hardware interfaces to a "Rovio" mobile robot**

Problem description:

The commercially available omnidirectional robot "Rovio" (http://www.meetrovio.com) used for remote household surveillance, offers a mobile platform equipped with a video camera and a microphone. This robot integrates in an existing WLAN network and communicates over standard TCP/IP ports with a remote software suite or a user-GUI in a web-browser. Such human control options are sufficient to purposefully navigate Rovio in remote environments; for research in autonomous robotics we would like to acquire a lower-level access to the robot's internal functions and possibly extend the robot's hardware with customized sensors.

Tasks:

In this practical, the students will work with an omnidirectional "Rovio" robot. Initially, the students will investigate in the communication protocol used to control the robot and aim to develop their own control software for remote control. We would like to use the robot - or parts of the robot - for our own experiments about autonomous navigation. In a second stage, students shall modify the existing hardware design of Rovio, such that the on-board camera is turned into a 360deg omnidirectional vision system (instead of a forward facing camera). In a third stage students shall explore if they can add further electronic devices (such as sensors or ideally a microcontroller) to Rovio existing hardware, which allows flexible extensions of Rovio as test-systems for robotic experiments.

- setup of Rovio, integration in WLAN, exploration of hardware and software
- analysis of TCP/IP-based communication protocol Rovio — host PC
- software development of simple interface to control Rovio's basic features
- hardware/mechanics modification of existing Rovio robot
- extension of existing electronic/hardware with customized sensors

This practical requires some experience with TCP/IP / internet programming in C and Java and some hands-on experience in mechanics/electronics. The web portal http://www.robocommunity.com is a great source for detailed insider information about Rovio.

Supervisor: Jörg Conradt

(J. Conradt)
Univ.-Professor

# Inhaltsverzeichnis

# 1 Einleitung

Im Projektpraktikum „Telepräsenz und Telerobotik“ soll ein Interface entwickelt werden welches den Rovio-Roboter der Firma WowWee kontrollieren kann. Das Interface soll in Java implementiert werden. Auf diese Weise sollen einige Grundfunktionalitäten bereitgestellt werden. Die beiden bedeutendsten sind die Bewegungsfunktionen und das Auswerten der Webcam. Außerdem ist es wichtig auch ein Auslesen des Roboterstatus zu ermöglichen. Wichtige Statusinformationen sind zum Beispiel der Akkustatus, sowie die relative Position zur Basisstation des Roboters. Des Weiteren soll die Möglichkeit von Hardwareerweiterungen erörtert werden. Außerdem muss die voraussichtlich benötigte Bandbreite abgeschätzt und die Bewegungsgenauigkeit vermessen werden.

## 1.1 Bestehende Technik des Rovio

Der Rovio Roboter ist ein Produkt der Firma WowWee. Er ist konzipiert, um sich über das Internet Zuhause umsehen und über Mikrophon und Lautsprecher mit Personen verständigen zu können. Zu diesem Zweck wurde eine Vielzahl an funktionaler Hardware verbaut. Neben der schon erwähnten Cmos Webcam, dem Lautsprecher und dem Mikrophon, wurde ein TrueTrack Navigationssystem eingebaut. Dieses System ermöglicht es dem Rovio seine Position im Raum zu bestimmen. Zur Positionsbestimmung wird, von der Basisstation aus, ein Positionierungssignal an die Decke projiziert. Dieses wird von einem im Rovio integrierten Empfänger ausgewertet und in Raumkoordinaten umgewandelt. Des Weiteren existiert ein Infrarot-Sensor an der Front des Roboters. Dieser Sensor kann zur Erkennung von Hindernissen verwendet werden. Zur Fortbewegung wurde ein Omnidirektionales Fahrwerk verbaut, welches alle Arten von Bewegungen, also translatorische und rotatorische auf der Fortbewegungsebene ermöglicht. Der Rovio wurde außerdem mit einem über W-Lan zu erreichenden Webserver bestückt. Dieser

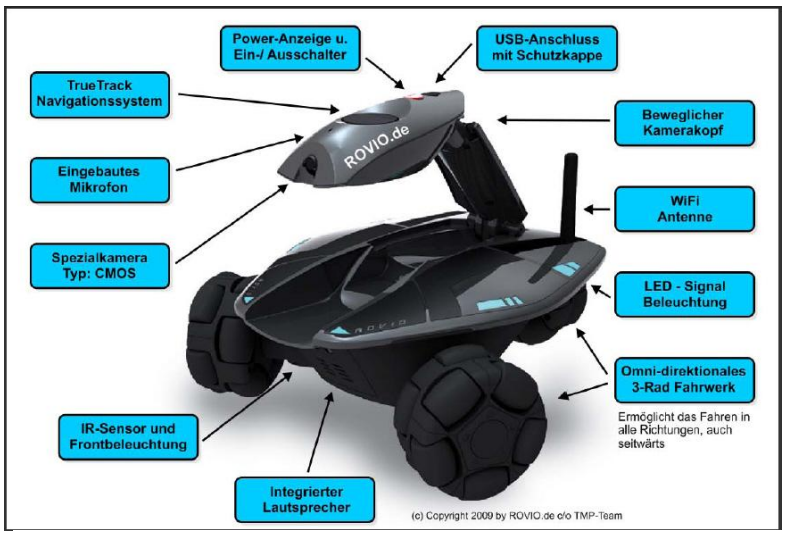


Abbildung 1: Rovio Schaubild[1]

kann entweder über ein ad-hoc oder über ein schon bestehendes W-Lan mit einem Computer verbunden werden. In diesem Netzwerk stellt der Webserver eine Weboberfläche zur Verfügung, über die alle Funktionalitäten des Roboters genutzt werden können. Diese Oberfläche ist über die IP-Adresse mit einem Browser erreichbar.

## 1.2 Lösungsansatz

In der von WowWee ausgegebenen API Spezifikation[2] ist der Kommunikationsablauf mit dem Roboter und der Befehlsumfang festgehalten. Es ist aus der Dokumentation zu entnehmen, dass Kommunikation über CGI-Skripte abläuft. Für die Entwicklung des Javainterfaces erleichtert diese Tatsache die Entwicklungsarbeit, da schon im Java-Framework Möglichkeiten geschaffen wurden, um ein CGI-Skript komfortabel aufzurufen. Zur Erstellung des Interfaces ist es einzig und allein nötig die einzelnen CGI-Aufrufe sinnvoll in die Methoden einer Java-Klasse einzubetten.

# 2 Das Javainterface

In diesem Kapitel soll die Interfacestruktur und der Softwareaufbau des Javainterface beschrieben werden.

## 2.1 Interfacestruktur

Um ein übersichtliches Interface zu erstellen, wird bewusst gegen eine Javakonvention verstoßen. Um eine Gruppierung der einzelnen Interface Funktionen zu erreichen werden Unterklassen implementiert, welche Methoden einer Funktionsgruppen enthalten. Auf diese Unterklasse wird über die Hauptklasse direkt, also nicht über einen Methodenaufruf, zugegriffen. So entsteht eine Struktur die der Java System Klasse ähnlich ist.

Java System Klasse:

```
System.out.println("Hallo");
```

Favorisierte Rovio Architektur :

```
rovio1.video.getImage();
rovio1.move.forward(int speed);
rovio1.move.backward(int speed);
```

Formal korrekte Rovio Architektur:

```
rovio1.video().getImage();
rovio1.move().forward(int speed);
rovio1.move().backward(int speed);
```

Abbildung 2: Interface Strukturbeispiel

In Abbildung 2 ist die Struktur der Systemklasse, der des Roviointerface gegenübergestellt. Es wird deutlich, dass eine Möglichkeit geschaffen wurde, elegant die Funktionen des Interfaces zu Gruppieren, um auf diese Weise die Übersichtlichkeit zu erhöhen. Es wäre auch möglich eine formal korrekte Struktur zu erzeugen. Dies würde jedoch bedeuten, alle Unterklassen (z.B. video, move) nur über einen Methodenaufruf erreichen zu können. Die Übersichtlichkeit wird dadurch etwas geschmälert. Wenn also eine Klasse lediglich dazu dient Funktionsblöcke zusammenzufassen, dann sollte, der Übersicht halber, mit direkten Klassenaufrufen gearbeitet werden.

## 2.2 Softwareaufbau

In diesem Kapitel ist dargelegt wie das Roviointerface funktioniert.

### 2.2.1 Rovio im Netzwerk Finden

Wenn in der späteren Anwendung mit festen IP`s gearbeitet wird ist diese Funktionalität nicht nötig. Allerdings kann es durchaus sinnvoll sein die Software so zu entwickeln, dass sie, mit der zur Verfügung stehenden Anzahl an Robotern, arbeitet. Dies würde die Robustheit der Anwendung deutlich steigern. Für diesen Fall ist eine Suchfunktion im Netzwerk unerlässlich.
Wie kann also ein Rovio im Netzwerk gefunden werden? Der Hersteller hat sich ebenso mit dieser Problematik auseinandergesetzt und bietet zur Lösung dieses Problems ein eigenes Programm an. Der Suchmechanismus wurde hingegen nicht in der API[2] dokumentiert. Um dennoch trotzdem diese Funktionalität in das Interface einbinden zu können, wurde mit Wireshark [1]die auftretende Netzwerkkommunikation mitgeschnitten und analysiert.

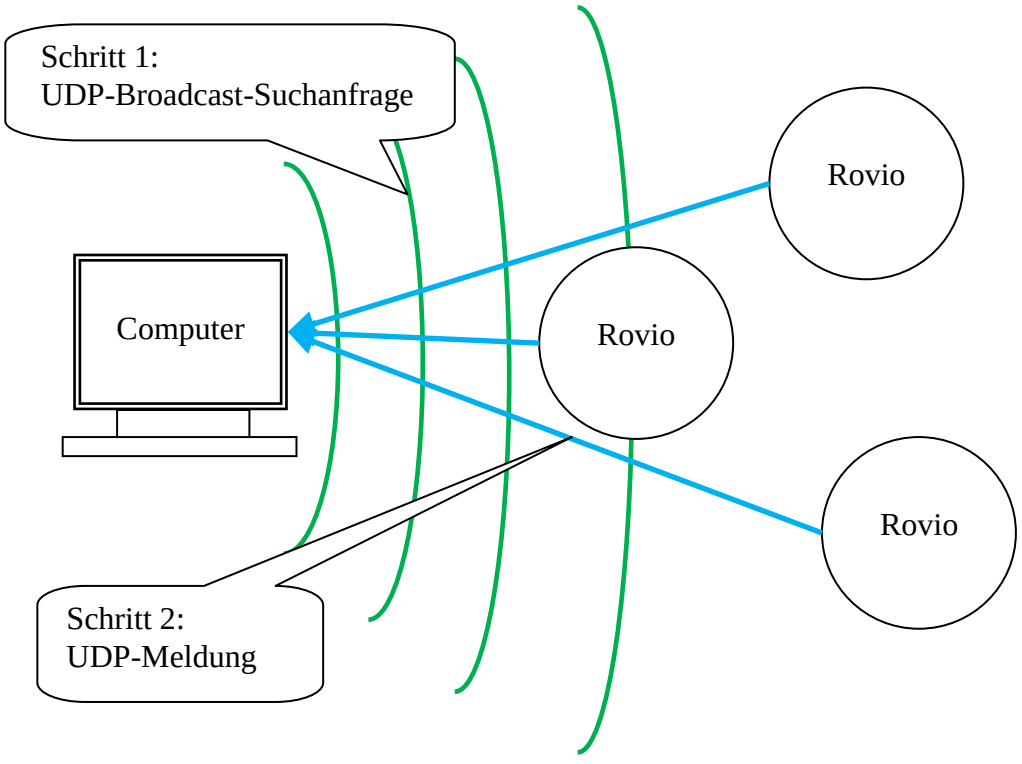


Abbildung 3: Netzwerk-Suchschema

In Abbildung 3 wird die grundsätzliche Funktionsweiße des Roviosuchmechanismuses dargestellt. Das Verfahren ist in zwei Schritten aufgebaut. Im ersten Schritt sendet der Computer, welcher die Rovios suchen möchte, ein spezielles Broadcastpacket. Jeder

[1] Wireshark ist ein Tool das es ermöglicht alle Paket mitzuschneiden welche die im PC verbauten Netzwerkkarten empfangen. Danach ist eine Analyse dieser Pakete möglich. (http://www.wireshark.org/)

Rovio, der dieses Packet empfängt, reagiert darauf in dem er eine UDP-Meldung schickt. Diese Nachricht wird direkt an den suchenden Computer gesendet. So teilt jeder im Netzwerk aktive Roboter dem Computer seine IP-Adresse mit. Im Interface ist diese Funktionalität als statische Methode realisiert. Dies hat den Vorteil, dass keine Instanz der Interfaceklasse erzeugt werden muss, um diese aufzurufen.

```
1: public static String[] findRovios() throws IOException{
2:      DatagramSocket socket = new DatagramSocket(4000);
3:      byte message[] = {0x49, 0x50, 0x45, 0x4E, 0x49, 0x53, 0x02};
4:      DatagramPacket p = new DatagramPacket(
5:                      message
6:                      ,message.length
7:                      ,new InetSocketAddress("255.255.255.255", 4000));
8:      socket.send(p);
9:      byte incommingMassage[] = new byte[1024];
10:     p = new DatagramPacket(incommingMassage, incommingMassage.length);
11:     socket.setSoTimeout(400);
12:     String lokalIp = InetAddress.getLocalHost().getHostAddress();
13:     Vector<String> rovios = new Vector<String>();
14:     try{
15:             while(true){
16:             socket.receive(p);
17:             if(p.getAddress().getHostAddress().compareTo(lokalIp)!=0){
18:                     if(!rovios.contains(p.getAddress().getHostAddress())){
19:                             rovios.add(p.getAddress().getHostAddress());
20:                     }
21:             }
22:             }
23:     } catch (IOException e) {
24:             ...
25:     }
26:     String array[] = new String[rovios.size()];
28:     return rovios.toArray(array);
29: }
```

Abbildung 4: Quellcode findRovios()

In Abbildung 4 ist der Code der Suchfunktion dargestellt. Wichtig für eine erfolgreiche Suche ist es, die Nachricht zu kennen, welche die Rovios dazu veranlasst sich beim Computer zu melden. Diese ist in Zeile 3 fest in das Programm eingebettet. In einen String übersetzt bedeutet sie „IPENIS“. Nachdem die Nachricht in ein Datagramm eingebunden wurde, welches als Zieladresse die Broadcastadresse 255.255.255.255 trägt, wird sie gesendet. Um die von den Robotern stammenden Meldungen zu erhaltenen, muss zuerst ein Datenpaket erstellt werden, in welches die empfangenen Daten geschrieben werden. Des Weiteren wurde ein Timer gesetzt, welcher eine Exception auslöst, wenn in 400 ms kein Paket empfangen wurde. Der Vector[2] rovios dient zum aufsammeln der gefundenen Rovio-Ip-Adressen. Die variable Datenstruktur ist nötig, da nicht bekannt ist wie viele Rovios sich im Netzwerk befinden. Nun kann die Verarbeitung der ankommenden Datenpakete betrachtet werden. Hierzu sind Zeile 14-25 wichtig. In Zeile 16 wird versucht ein Datenpaket zu empfangen. Ist dies innerhalb von 400ms möglich, so wird in Zeile 17 überprüft, ob das Datenpaket das gesendete Broadcast-Paket ist. Dieses wird auch vom Sendenden Rechner Empfangen. In Zeile 18 wird noch überprüft ob der Rovio mit dieser IP schon in dem Vector(rovios) enthalten ist. Fällt diese Prüfung negativ aus, so wird die IP-Adresse des Empfangenen Paketes

[2] Java API[3] java.util.Vector

dem Vector hinzugefügt. Ist es nicht möglich innerhalb von 400ms ein Paket zu empfangen, so wird eine Exception ausgelöst, welche die in Zeile 15 begonnene Endlosschleife beendet. Im Catch-Block, der nach Beendigung der Endlosschleife aufgerufen wird, ist der Broacast-Empfangs-Mechanismus ein zweites Mal Implementiert, um eventuelle Paketverluste bei der ungesicherten Datenübertragung mit UDP auszugleichen. Aus Platzgründen wurde auf eine Darstellung dieser Redundanz verzichtet.

### 2.2.2 Verbindungsaufbau

Um eine Verbindung zum Rovio aufzubauen muss zu allererst eine Standard HTTP-Authentifizierung durchgeführt werden. Hiermit soll von Seiten des Herstellers ein gewisses Mindestmaß an Sicherheit bereitgestellt werden. Das Java Framework bietet hierfür eine sehr komfortable Lösung. Es ist möglich einen Authenticator zu erstellen, welcher, nachdem er als Standard-Authenticator gesetzt ist, die HTTP-Authentifizierung übernimmt. Dieser Mechanismus ist in der Java API[3] (java.net.Authenticator) ausführlich beschrieben. Desweiteren ist es sinnvoll, wenn eine Authentifizierung durchgeführt wurde, die Verbindung zu halten, sodass es nicht nötig ist sich für jede Anfrage neu anzumelden. Diese Funktionalität wird auch vom Framework bereitgestellt. Der komplette Verbindungsaufbau kann somit im Konstruktor des Rovio-Interfaces stattfinden. Der Konstruktor selbst setzt nur den Authenticator und erstellt die Funktionsblöcke wie es in Abbildung 5 deutlich wird.

```
1: public class Rovio {
2:
3:      public Move move;
4:      public ContMove contMove;
5:
6:      public Video video;
7:      public ContVideo contVideo;
8:
9:      public Status status;
10:     public ContStatus contStatus;
11:
12:     public Rovio(String ip, String user, String password)
                    throws MalformedURLException{
13:             Authenticator.setDefault(new RovioAuthenticator(user, password));
14:
15:             move = new Move(ip);
16:             contMove = new ContMove(move);
17:
18:             video = new Video(ip);
19:             contVideo = new ContVideo(video);
20:
21:             status = new Status(ip);
22:             contStatus = new ContStatus(ip);
23:     }
24:     ...
25:}
```

Abbildung 5: Konstruktor und Attributdefinition des Roviointerfaces

Es ist ersichtlich wie einfach das Problem der Authentifizierung gelöst werden konnte. Allerdings ist es nötig selbst eine Authenticator-Klasse zu implementieren.

```
1: public class RovioAuthenticator extends Authenticator{
2:     private PasswordAuthentication pa;
3:
4:     public RovioAuthenticator(String user, String password){
5:             pa = new PasswordAuthentication (user, password.toCharArray());
6:     }
7:     @Override
8:     protected PasswordAuthentication getPasswordAuthentication() {
9:     return pa;
10:    }
11: }
```

Abbildung 6: RovioAuthenticator

RovioAuthenticator leitet sich von Authenticator ab. Was bedeutet dass alle Attribute und Methoden von Authenticator auch im RovioAuthenticator zur Verfügung stehen. Der eigentliche Mechanismus zur Passwordübergabe (getPasswordAuthentication()) wird hingegen überschrieben und durch einen neuen ersetzt. So wird eine Instanz der PasswordAuthentication, welche nur zur Speicherung des Benutzernamen und des Passwortes dient. Der Benutzername und das Passwort werden als Klartext im Arbeitsspeicher hinterlegt. Diese Vorgehensweise kann es einem Angreifer ermöglichen unter Umständen die Zugangsdaten auszuspionieren. Die geplanten Projekte jedoch, sind nicht so sicherheitskritisch, dass dies ein Problem darstellen würde.

### 2.2.3 Aufruf eines CGI-Befehls

Um die Rovio-API[2] nutzen zu können, muss einen Möglichkeit gefunden werden, mit Java CGI-Befehle aufzurufen. In „Java ist auch eine Insel“[4], wird die Verwendung der Klassen URL und URLConnection [3] zur Behandlung von CGI-Abfragen vorgeschlagen.

```
1:    URL call = new URL("http://192.168.2.126/rev.cgi?Cmd=nav&action=18");
2:    URLConnection uc = call.openConnection();
3:    uc.setDoOutput(true);
4:    InputStream in = uc.getInputStream();
```

Abbildung 7: Beispil CGI-Aufruf

Um einen CGI-Befehl von Java aus aufzurufen sind vier Operationen durchzuführen. In Abbildung 7 ist ein Beispielcode zu sehen, der diese ausführt. Als erstes muss eine URL erzeugt werden. An den Konstruktor ist ein String zu übergeben werden. Dieser enthält: die Adresse des Roboters, den Namen des CGI-Skriptes und die Attribute, welche für den Aufruf nötig sind. Anschließend kann die Methode openConnection() der Instanz der Klasse URL aufgerufen werden. Der Name dieser Methode ist leicht irreführend, da mit dem Aufruf dieser, noch keine Verbindung hergestellt wird. Es wird lediglich eine Instanz der Klasse URLConnection erzeugt, welche in der Lage ist, eine Verbindung zu etablieren. Um Ausgaben in der Verbindung zu ermöglichen ist setDoOutput(true) aufzurufen. Nachdem die Connection den Bedürfnissen entsprechend konfiguriert ist, kann die Methode getInputStream() aufgerufen werden. Durch diesen Funktionsaufruf werden einige Mechanismen angetriggert. Zum einen wird die TCP-Verbindung hergestellt, wenn dies der erste Aufruf an die Zieladresse ist. Beim Aufbau der TCP-

[3] URL und URLConnection sind in java.net Paket enthalten

Verbindung, wird auch die Authentifizierung durchgeführt. Ist es nicht der erste Aufruf, wird auf eine, im Hintergrund am Leben erhaltene, TCP-Verbindung zurückgegriffen. Dieser Mechanismus vermeidet unnötigen Overhead. Außerdem wird über die TCP-Verbindung eine http-Anfrage geschickt, welche das CGI-Skript anfordert und somit aufruft. Anschließend kann der Stream ausgewertet werden. Dieser enthält die Antwort des CGI-Skriptes. Erste Steuerversuche zeigen, dass die Rovio-API[2] eine Schwachstelle aufweist. So ist es nicht möglich, für die Bewegungsfunktionen absolute Steuerbefehle abzusetzen, also eine Strecke oder eine Zeitdauer und Geschwindigkeit vorzugeben. Der Steuerbefehl versetzt den Roboter für ca. 200ms in den gewünschten Bewegungszustand. Wird kein weiterer Befehl übergeben hört er auf, sich zu bewegen. Wird hingegen innerhalb dieses Zeitraums ein weiterer Steuerbefehl abgesetzt, so bewegt sich der Roboter ohne Unterbrechung weiter. Für einen kontinuierlichen Betrieb wäre es also nötig, dass ein steuerndes Programm eine Zykluszeit kleiner 200ms aufweist. Um es zu ermöglichen, das Interface bei Programmen zu nutzen die mehr Zeit benötigen um Steuerentscheidungen zu treffen. Also ist es unumgänglich Nebenläufigkeiten einzusetzen. Java bietet hierzu zwei Standardlösungen an, welche im folgenden Kapitel vorgestellt werden sollen.

### 2.2.4 Einschub Thread oder Task

In Java existieren zwei Arten von Nebenläufigkeiten. Es gibt gezeitete (TimerTasks) und ungezeitete (Threads). Für das Interface ist nur der Schleifenbetrieb dieser beiden Nebenläufigkeiten interessant.

```
1: public class MyThread extends Thread{
2:      Object sync;
3:      public MyThread(){
4:              sync = new Object();
5:
6:      public void wakeup(){
7:              synchronized( sync ){
8:                      sync.notify();
9:              }
10:     }
11:     @Override
12:     public void run() {
13:             while(!this.isInterrupted()){
14:                     //do something
15:
16:                     try {
17:                             sync.wait(1000);
18:                     } catch (Exception e) {}
19:             }
20:     }
21:}
```

Abbildung 8: Beispiel Thread

In Abbildung 8 ist eine Threadstruktur schematisiert. Die gewünschte Funktionalität der Nebenläufigkeit muss, sowohl bei einem Thread, als auch bei einem Task, in der run() Methode implementiert werden. Wird bei einem Thread das Ende der Methode erreicht, so endet damit auch der Thread. Aus diesem Grund wird in einem Thread eine While-Schleifen-Konstruktion verwendet, welche mit isInterrupted() abfragt, ob die Methode interrupt() aufgerufen wurde. So ist es möglich die While-Schleife sauber von außen zu beenden. Für die meisten Threads ist es wichtig, eine warte Funktion einzubauen. Diese

wurde bewusst mit wait(1000) und nicht mit sleep(1000) realisiert. Da wait() noch zusätzlich die Möglichkeit bietet, den Thread vor Ablauf der eingestellten Wartezeit aufzuwecken. Diese Funktionalität ist als Wait-Nodify Konstruktion bekannt. Wichtig für dieses Konstrukt ist das Vorhandensein eines Objektes, welches mit wait() beobachtet wird. Es existieren zwei Möglichkeiten um wait() zu verlassen. Es wird entweder gewartet bis die übergebene Zeit abgelaufen ist, oder bis ein Aufruf der Funktion nodify() erfolgt ist. Dieses Konstrukt ermöglicht es, einen periodischen Ablauf zu implementiert, dessen Wartezeit durch asynchron auftretende Ereignisse unterbrochen werden kann. Das Zeitverhalten dieses Threads ist sehr stark vom Scheduler des Betriebssystems abhängig. Für zeitkritische Anwendungen in denen eine genaue Periodendauer eingehalten werden soll, ist diese Konstruktion also weniger geeignet.

```
1: public class MyTask extends TimerTask{
2:
3:      @Override
4:      public void run() {
5:              //do something
6:      }
7: }
```

Abbildung 9: Beispiel TimerTask

Der in Abbildung 9 dargestellte TimerTask ist wesentlich einfacher aufgebaut. Es ist nur die gewünschte Funktionalität in der run() Methode zu implementieren. Alle Peripherie die nötig ist, wird von einem Timer bereitgestellt. So muss keine Schleife erzeugt werden, um einen kontinuierlichen Betrieb sicher zu stellen. Ist ein solcher Betrieb gewünscht, so kann der Timer angewiesen werden, dieses Verhalten zu erzeugen. Der Timer ermöglicht es, genaue Periodendauern zu realisieren. So ist dieser für zeitkritische Anwendungen besonders geeignet.

```
1: public class Call {
2:
3:      public static void main(String[] args) {
4:
5:              MyThread myThread = new MyThread();
6:              myThread.start();
7:
8:              Timer timer = new Timer();
9:              MyTask myTask = new MyTask();
10:             timer.schedule(myTask, 0, 1000);
11:
12:             myThread.interrupt();
13:             myTask.cancel();
14:     }
15:
16: }
```

Abbildung 10: Beispiel Aufruf

Nach dem die Möglichkeiten von Thread und Task erörtert wurden, soll nun betrachtet werden, wie die Nebenläufigkeiten erzeugt, gestartet und beendet werden. Im Falle des Threads ist dieser Ablauf relativ einfach zu beschreiben. Als erstes muss eine Instanz des Threads erzeugt werden. Im nächsten Schritt wird diese mit start() ausgeführt. Um den

Thread sauber zu beenden wird nicht etwa die Metode stop() verwendet, da diese den Thread beendet, ohne darauf zu achten, ob zum Beispiel Streams oder ähnliches noch zu schließen sind. So ist der Aufruf der Methode interrupt in Kombination mit der While-Schleife eine gute Möglichkeit den Thread sauber zu terminieren. Einen TimerTask zu starten bedarf zweierlei Dinge: Die Erstellung einer Taskinstanz und zusätzlich einer Timerinstanz, welche den Task scheduled. Hierbei bietet der Timer mehrere Schedule-Verhalten an. In dieser Arbeit soll nur auf das, in Abbildung 9 dargestellte, Verhalten

```
timer.schedule(myTask, 0, 1000)
```

eingegangen werden. Um den Timer anzuweisen einen Task zu verarbeiten muss dieser mit schedule übergeben werden. Der erste Integerwert steht für die Zeit, welche gewartet werden muss, bis der Task zum ersten Mal ausgeführt wird. Der zweite steht für die periodische Wiederholungszeit des Tasks. Um einen Task zu beenden kann die Methode cancel() aufgerufen werden.

### 2.2.5 Bewegung

In diesem Kapitel soll erörtert werden welche Eigenschaften die Bewegungsfunktionen der Rovio-API[2] aufweisen und, wie ein Interface diese sinnvoll nutzen kann.

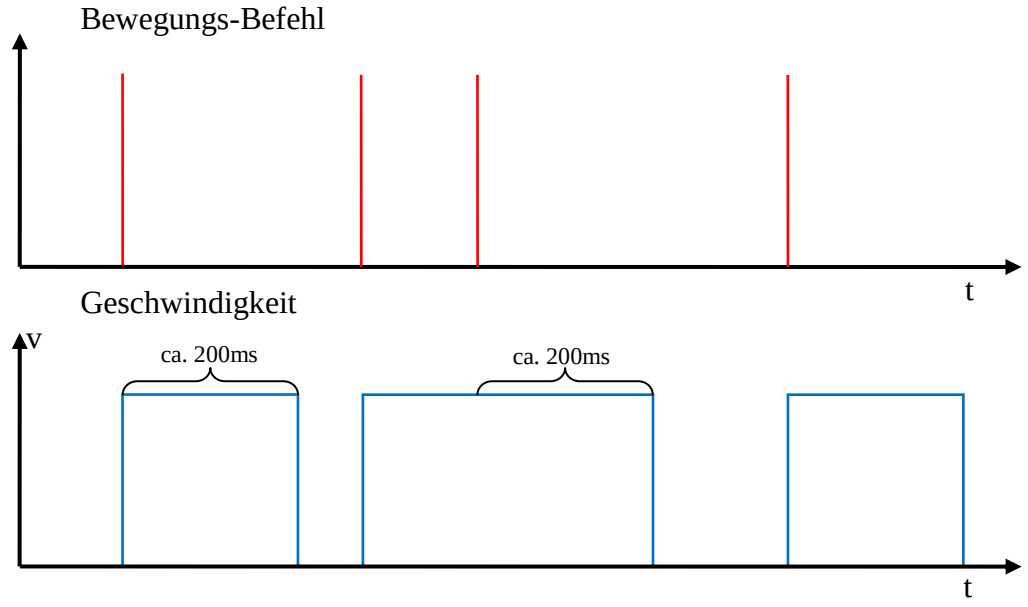


Abbildung 11: Bewegungsmechanismus

In Abbildung 11 ist die Reaktion des Rovio auf einen erhaltenen Bewegungsbefehl, wie er in der API dokumentiert ist, dargestellt. Wird ein solcher Befehl abgesetzt, so bewegt sich Rovio, 200 ms lang, in die Richtung, die der Befehl vorschreibt. Ebenso kann der Zeitraum durch einen weiteren Befehl nachgetriggert, also verlängert werden. Dies bedeutet, dass für eine kontinuierliche Fahrt spätestens alle 200ms ein neuer Bewegungsbefehl gesendet werden muss. Ein Regelalgorithmus, der auf Bildverarbeitung beruht, braucht möglicherweise länger, um ein Steuersignal zu berechnen. Deshalb ist es sinnvoll, neben dem direkten Befehlsaufruf auch einen kontinuierlichen Betrieb zu ermöglichen. Um diesen kontinuierlichen Betrieb zu

ermöglichen, muss also eine Nebenläufigkeit mindestens alle 200ms, die ihr gegebene Anweisung, wiederholen. Dies sollte solange geschehen bis eine neue Anweisung gegeben wird. Eine Anweisungsänderung soll sich möglichst sofort auswirken. Wie in Kapitel 2.2.4 diskutiert, lässt sich ein solches Verhalten am besten mit einem Thread, der eine Wait-Nodify-Struktur enthält, realisieren. Auf diese Art wird es möglich asynchrone Befehlsaufrufe zu realisieren.

```
1: public class ContMoveThread extends Thread{
2:      Object sync;
3:      boolean contMove;
4:      Move move;
5:      ContMod modus;
6:      int speed;
7:
8:      public ContMoveThread(Move move){
9:              sync = new Object();
10:             this.move = move;
11:             modus = ContMod.Stop;
12:             contMove = false;
13:     }
14:     public void StartContMoveTask(){
15:             contMove = true;
16:             synchronized( sync ){
17:                     sync.notify();
18:             }
19:     }
20:     public void StopContMoveTask(){
21:             contMove = false;
22:             synchronized( sync ){
23:                     sync.notify();
24:             }
25:     }
26:     …
27:     public synchronized void setModus(ContMod modus,int speed){
28:             this.modus = modus;
29:             this.speed = speed;
30:             synchronized( sync ){
31:                     sync.notify();
32:             }
33:     }
34:     …
35:     @Override public void run(){
36:             while(!this.isInterrupted()){
37:                     if(contMove){
38:                             switch(modus){
39:                             case Stop:
40:                                     break;
41:                             case Forward:
42:                                     move.forward(speed);
43:                                     break;
44:                             …
45:                             case RotateRight:
46:                                     move.rotateRightBySpeed(speed);
47:                                     break;
48:                             }
49:                             synchronized( sync ){
50:                                     sync.wait(50);
51:                             }
52:                     }
53:                     else{
54:                             synchronized( sync ){
55:                                     sync.wait(50);
56:                             }
57:                     }
58:             }
59:     }
60:}
```

Abbildung 12: Bewegungsthread Struktur

In Abbildung 12 ist ein Auszug aus dem Quellcode des Bewegungsthread zu sehen. Um die gewünschte kontinuierliche Befehlswiederholung zu realisieren, wurde eine einfache Status-Maschine implementiert. In der Variable Modus ist der aktuelle Status der Maschine gespeichert. Diese Variabel ist vom Typ ContMod, was nichts weiter als einer Enumeration entspricht, welche alle gewünschten Zustände namentlich enthält. Die Switch-Case-Anweisung, in der run() Methode des Threads, stellt sicher, dass dem Status des Systems entsprechend, ein Befehl an den Rovio gesendet wird. Durch wait(50) wird eine Periodendauer von 50ms realisiert. Diese Periodendauer wurde gewählt, um mit einer möglichst großen Sicherheit den kontinuierlichen Betrieb aufrecht erhalten zu können. Es kann nicht davon ausgegangen werden, dass der Thread eine reale Wiederholdauer von 50ms aufweist. Schließlich liegt das Zeitmanagement im Zuständigkeitsbereich des betriebssystemeigenen Scheduler. Die Methode setModus(…) wurde so aufgebaut, dass eine Statusänderung dazu führt, den Befehl wait(50) zu unterbrechen und die neue Anweisung sofort an Rovio zu übermitteln. Der im Hintergrund laufende Thread wurde noch in einer zweiten Klasse gekapselt. In dieser sind nur die Methoden angelegt, welche auch für das Interface von Bedeutung sind. Dies erhöht die Übersichtlichkeit und macht es somit einfacher, ein eigenes Programm mit diesem Interface zu entwickeln.

```
1: public class ContMove{
2:      Move move;
3:      ContMoveThread task;
4:
5:      public ContMove(Move move) throws MalformedURLException {
6:              this.move = move;
7:              task = new ContMoveThread(move);
8:              task.start();
9:      }
10:     public void startContMove(){
11:             task.StartContMoveTask();
12:     }
13:     public void stopContMove(){
14:             task.StopContMoveTask();
15:     }
16:     public void close(){
17:             task.interrupt();
18:     }
19:     public void stop() throws IOException{
20:             task.setModus(ContMod.Stop);
21:     }
22:     public void forward(){
23:             task.setModus(ContMod.Forward);
24:     }
25:     public void backward(){
26:             task.setModus(ContMod.Backward);
27:     }
28:     …
29:     public void rotateRight(int speed){
30:             task.setModus(ContMod.RotateRight,speed);
31:     }
32: }
```

Abbildung 13: Kapselung des Threads

Die Klasse ContMove, wie sie in Abbildung 13 zu sehen ist, wurde zur Kapselung des Threads erstellt. Es ist zu erkennen, dass der Konstruktor automatisch eine Instanz des ContMoveThread`s erstellt und diese startet. Mit startContMove() und stopContMove() lässt sich der Thread aktiviren, bzw. in einen Wartezustand setzen. Close() dient der endgültigen Terminierung des Threads. Alle weiteren Funktionen bilden die Bewegungsmöglichkeiten des Roboters ab.

### 2.2.6 Video

Wird der CGI-Befehl aufgerufen, welcher den Rovio dazu veranlasst ein JPG zu übertragen, so benötigt die Übertragung, des Bildes und die Umwandlung von einem JPG zu einem Java BufferedImage[4], eine gewisse Zeit. Diese Zeitverzögerung wurde statistisch erfasst und die Durchschnittswerte und Standartabweichung in Tabelle 3 auf Seite 28 für verschiedene Auflösungen und Qualitätsstufen festgehalten. Es kann allerding möglich sein, dass für manche Anwendungen diese Verzögerungszeit zu groß ist. Zu diesem Zweck soll eine Möglichkeit geschaffen werden, kontinuierlich Bilder vom Rovio herunterzuladen. Außerdem soll immer das letzte, vollständig heruntergeladene Bild asynchron bereitgestellt werden. Zur Erfüllung dieser Forderung wurde ein TimerTask gewählt.

```
1: public class ContVideoTask  extends TimerTask{
2:      BufferedImage image;
3:      double time1, time2;
4:
5:      Video video;
6:
7:      public ContVideoTask(Video video) {
8:              this.video = video;
9:              try {
10:                     image = video.getImage();
11:             } catch (IOException e) {
12:                     // TODO Auto-generated catch block
13:                     e.printStackTrace();
14:             }
15:     }
16:     public synchronized BufferedImage getContImage(){
17:             return image;
18:     }
19:     public synchronized double getAktuellFramesPerSecond(){
20:             return (1/((double)(time2 - time1)/1000000000));
21:     }
22:     private synchronized void setTimes(){
23:             time1 = time2;
24:             time2 = System.nanoTime();
25:     }
26:     private synchronized void setContImage(BufferedImage image){
27:             this.image = image;
28:     }
29:     @Override public void run(){
30:             setTimes();
31:             try {
32:                     setContImage(video.getImage());
33:             }
34:             catch (IOException e) {
35:                     // TODO Auto-generated catch block
36:                     e.printStackTrace();
37:             }
38:     }
39: }
```

Abbildung 14: Video Task

Durch die run() Methode des Tasks wird noch einmal deutlich wie einfach ein Task aufgebaut ist, im Vergleich zu einem Thread. Die Methode setTimes() dient der Bestimmung der Zykluszeit des Tasks. Durch getAktuellFramesPerSecond() kann aus der Zykluszeit die Framerate berechnet werden. Die Hauptfunktionalität wird in Zeile 32 erzeugt. Durch getImage() wird ein Bild angefordert und wenn dieses vollständig geladen und umgewandelt wurde, wird die Referenz auf das Bild mit setContImage() in

[4] Enthalten im Java-Packet java.awt.image.BufferedImage

der Variable image gespeichert. Es wurde bewusst auf eine direkte Zuweisung verzichtet, um eventuelle Synchronisationsprobleme zu umgehen. Der Videotask wurde aus Gründen der Übersicht auch in einer weiteren Klasse gekapselt. Eine Besonderheit im Vergleich zu BewegungsTask ist die Tatsache, dass beim Start des Tasks eine Framerate übergeben werden kann, die beim nicht Überschreiten der maximalen Framerate, relativ genau eingehalten wird.

### 2.2.7 Status

Für das Abrufen der Statusinformationen existieren auch zwei Möglichkeiten. Einmal das direkte Abrufen, welches mit einer Zeitverzögerung behaftet ist. Und der kontinuierliche Modus, welcher einer Framerate entsprechend, die Daten lädt. Es wurde, wie bei den Video Funktionen ein Task verwendet. Der Vorteil der Kontinuierlichen Variante im Vergleich zur Direkten ist der geringere Overhead, der Auftritt, weil nicht jede Information einzeln geladen werden muss. Dieser Vorteil kommt nur dann zum Tragen, wenn viele verschiedene Daten in einem Berechnungsschritt benötigt werden. Um den Wert des Infrarotsensors zu erhalten war es unerlässlich, ein separates Datenpaket anzufordern. Deswegen wurde die Funktion nur in die direkte Funktionsgruppe eingegliedert.

### 2.2.8 Verbindungsende

Um ein sauberes Terminieren des Programms zu gewährleisten, welches das Interface nutzt, wurde die close() Methode implementiert. Durch den Aufruf dieser Methode werden alle Nebenläufigkeiten, die zum Betrieb des Rovio nötig sind, beendet.

## 2.3 Interfaceübersicht

Zur Darstellung der Einzelfunktionen des Interfaces soll eine Instanz der Rovio Klasse mit dem Namen rovio verwendet werden.

### 2.3.1 Bewegung

Als erstes sollen die Bewegungsfunktionen demonstriert werden. Hierzu soll die schematische Abbildung 15 verwendet werden. Diese zeigt die möglichen Bewegungsrichtungen des Roboters.

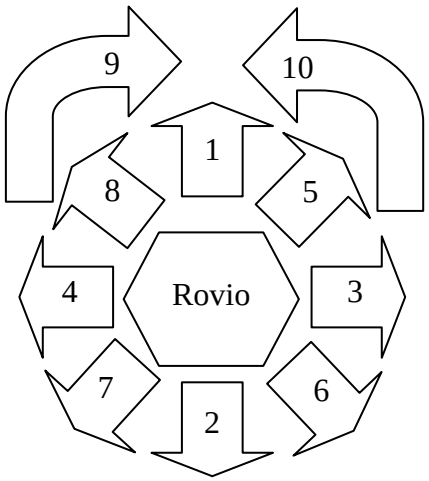

Abbildung 15: Bewegungsrichtungen

Einzelbefehle (ca. 200ms Ausführzeit):

Die Variable Speed ermöglicht eine Geschwindigkeitsvariation von 1 langsam bis 10 schnell.

1. `rovio.move.forward(int speed)`
2. `rovio.move.backward(int speed)`
3. `rovio.move.straightRight(int speed)`
4. `rovio.move.straightLeft(int speed)`

5. `rovio.move.forwardRight(int speed)`
6. `rovio.move.backwardRight(int speed)`
7. `rovio.move.backwardLeft(int speed)`
8. `rovio.move.forwardLeft(int speed)`
9. `rovio.move.rotateLeftBySpeed(int speed)`
10. `rovio.move.rotateRightBySpeed(int speed)`

Des Weiteren ist es möglich die rotatorischen Bewegungen 9 und 10 winkelabhängig auszuführen. So führt ein einmaliger Funktionsaufruf zu einer Drehung um 20 Grad.

9. `rovio.move.rotateLeftBy20degree()`
10. `rovio.move.rotateRight20degree()`

Außerdem kann die Funktion stop() eine aktuell ausgeführte Aktion unterbrechen.

Kontinuierliche Befehle:

Die Variable Speed ermöglicht eine Geschwindigkeitsvariation von 1 langsam bis 10 schnell.

1. `rovio.contMove.forward(int speed)`
2. `rovio.contMove.backward(int speed)`
3. `rovio.contMove.straightRight(int speed)`
4. `rovio.contMove.straightLeft(int speed)`
5. `rovio.contMove.forwardRight(int speed)`
6. `rovio.contMove.backwardRight(int speed)`
7. `rovio.contMove.backwardLeft(int speed)`
8. `rovio.contMove.forwardLeft(int speed)`
9. `rovio.contMove.rotateLeftBySpeed(int speed)`
10. `rovio.contMove.rotateRightBySpeed(int speed)`

Es ist auch möglich, keine Geschwindigkeit zu übergeben, dann wird der als letztes übergebene Geschwindikeitswert weiter verwendet.

1. `rovio.contMove.forward()`
2. `rovio.contMove.backward()`
3. `rovio.contMove.straightRight()`
4. `rovio.contMove.straightLeft()`
5. `rovio.contMove.forwardRight()`
6. `rovio.contMove.backwardRight()`
7. `rovio.contMove.backwardLeft()`
8. `rovio.contMove.forwardLeft()`
9. `rovio.contMove.rotateLeftBySpeed()`
10. `rovio.contMove.rotateRightBySpeed()`

Außerdem kann die Funktion stop() eine aktuell ausgeführte Aktion unterbrechen

### 2.3.2 Video

Es wurde die Möglichkeit geschaffen, direkt ein Bild vom Rovio zu laden. Hierbei tritt allerdings eine Verzögerung auf (Tabelle 4 S.28)

```
rovio.video.getImage()
```

Die zweite Möglichkeit ist die Verwendung von ContVideo. Da im Hintergrund kontinuierlich Bilder geladen werden und dass zuletzt vollständig Geladene zur Verfügung gestellt wird, ist kaum eine Verzögerungszeit vorhanden.

```
rovio.contVideo.getImage()
```

In beiden Befehlsgruppen kann die Position des Kameraarms mit folgenden Befehlen variiert werden.

```
Oben:       rovio.video.haedUp()
Mitte:      rovio.video.haedMiddle()
Unten:      rovio.video.haedDown()


Oben:       rovio.contVideo.haedUp()
Mitte:      rovio.contVideo.haedMiddle()
Unten:      rovio.contVideo.haedDown()
```

### 2.3.3 Status

Die einzelnen Statusbefehle, die zur Verfügung stehen, sind im Folgenden aufgelistet. Es stehen alle Befehle bis auf `getIRValue()` sowohl in `status`, als auch in `contStatus` zur Verfügung.

```
rovio.status.getPosition()
```

```
rovio.contStatus.getPosition()
```

Liefert die Klasse RovioPosition, welche die Werte X,Y, Theta und RoomID mit getX() getY(),getTheta() und getRoomID() bereitstellt. Diese Werte werden mit dem Truetrack Navigationssystem ermittelt.

getX():

x Abweichung zum Bezugspunkt

getY():

y Abweichung zum Bezugspunk

getTheta():

Ausrichtung im Raum

getRoomID():

ID des Bezugspunktes

```
rovio.status.getWLanSignalStrength()
rovio.contStatus.getWLanSignalStrength()
```

Liefert einen Integerwert zurück, der zwischen 0 und 254 liegt. Wobei 0 für schlechte und 254 gute WLan-Signalqualität bedeutet

```
rovio.status.getHeadPosition()
rovio.contStatus.getHeadPosition()
```

Liefert einen Integerwert zurück, welcher die Position des Kameraarms repräsentiert.

Oben: 65

Mitte: 135-140

Unten: 204

```
rovio.status.getBatteryStatus()
rovio.contStatus.getBatteryStatus()
```

Es wird ein Integerwert übergeben, welcher für den Ladezustand der Batterie steht. Der Wertebereich für einen normalen Ladezustand reicht von 106-127. Die Marke von 106 sollte allerdings nicht unterschritten werden, da sich Rovio sonst selbst deaktivieret, oder versucht zur Ladestation zurück zu kehren.

```
rovio.status.getChargingStatus()
rovio.contStatus.getChargingStatus()
```

Wird eine 80 zurück gegeben, so ist der Roboter gerade damit beschäftigt seine Batterie aufzuladen.

```
rovio.status.getIRValue()
```

Es wird ein boolean übergeben. Bei true ist vom IR-Sensor ein Hindernis erkannt worden. Bei false konnte kein Hindernis erkannt werden.

# 3 Technische Erweiterbarkeit

Die technische Erweiterbarkeit des Rovio ist nach offiziellem Informationsstand der Firma WooWee sehr begrenzt. So sind in der API keine Funktionen aufgeführt, welche es erlauben auf Datenleitungen wie zum Beispiel einem Com Port oder einem I2C Bus zuzugreifen. Allerdings ist schon bekannt, dass versteckte Funktionen existieren, welche nicht in der API dokumentiert wurden. Eine realistische Möglichkeit zur Nutzung des Systems als Trägersystem wäre: die Nutzung eines separaten WLan-Moduls, um einen zusätzlichen Datenkanal zu schaffen. Über diesen Datenkanal könnte zusätzliche Peripherie angesteuert werden. Es wäre ebenso möglich zusätzliche Sensordaten über diesen Kanal zu empfangen. Nach aktuellem Informationsstand scheint dies die praktikabelste Lösung zu sein. Sollte der Aufwand hierfür zu groß sein, ist der Einsatz als Trägersystem fraglich.

# 4 Analyse technischer Eckdaten

Dieses Kapitel soll einen Überblick über die Leistungsfähigkeit des Rovio in Kombination mit dem Interface liefern.

## 4.1 Bandbreitenbedarf

Zur Vermessung der Bandbreite wurde der in Windows7 integrierte Ressourcenmonitor verwendet. Dieser ermöglicht es den Bandbreitenbedarf einzelner Anwendungen zu vermessen. In Tabelle 1 wurde die benötige Bandbreite bei maximaler Framerate bestimmt.

| | Auflösung | | | | |
|---|---|---|---|---|---|
| Qualität | | **176x144** | **320x240** | **352x288** | **640x480** |
| | **Niedrig** | 2,1MBit/s (29fps) | 3,8MBit/s (29fps) | 3,7MBit/s (20fps) | 5MBit/s (12fps) |
| | **Mittel** | 2,2MBit/s (29fps) | 3,9MBit/s (28fps) | 3,7MBit /s (19fps) | 5,4MBit /s (12fps) |
| | **Hoch** | 2,4MBit/s (29fps) | 4MBit/s (26fps) | 4,1MBit /s (17fps) | 5,4MBit /s (10fps) |

Tabelle 1: Bandbreitenbedarf bei Maximaler Framerate

In Tabelle 2 wurde die benötigte Bandbreite bei einer festen Framerate von 6 Frames per Second (fps) bestimmt.

| | Auflösung | | | | |
|---|---|---|---|---|---|
| Qualität | Bei 6fps | **176x144** | **320x240** | **352x288** | **640x480** |
| | **Niedrig** | 0,5MBit/s | 0,8MBit/s | 1,2MBit/s | 2,6MBit/s |
| | **Mittel** | 0,5MBit/s | 0,9MBit/s | 1,3MBit /s | 2,7MBit /s |
| | **Hoch** | 0,5MBit/s | 1MBit/s | 1,4MBit /s | 3,2MBit /s |

Tabelle 2: Bandbreitenbedarf bei 6 FPS

## 4.2 Videoverzögerungszeit

Die Verzögerungszeit für den direkten Bildladebefehl video.getImage() wurde durch eine einfache Zeitmessung, vor und nach Ausführung der Methode, bestimmt. So kann

eine einzelne Verzögerungszeit errechnet werden. Da diese Zeit allerdings nicht konstant ist, wurden die Methode tausendmal vermessen und die Standartabweichung S bestimmt. Zur Berechnung wurden folgende Formeln verwendet.

$$S = \sqrt[2]{\frac{1}{n-1}\sum_{i=0}^{n}(X_i - \bar{X})^2}$$

$$\bar{X} = \frac{1}{n}\sum_{i=0}^{n} X_i$$

| | Auflösung | | | | |
|---|---|---|---|---|---|
| Qualität | | **176x144** | **320x240** | **352x288** | **640x480** |
| | **Niedrig** | 34,9ms / 2,2ms | 34,1ms / 7,5ms | 47,9ms / 31,7ms | 81,4ms / 10,7ms |
| | **Mittel** | 35,6ms / 3,8ms | 33,8ms / 6,8ms | 52,1ms / 14,9ms | 86,4ms / 14,3ms |
| | **Hoch** | 34,7ms / 2,5ms | 34,2ms / 8,1ms | 68,9ms / 14,3ms | 90,1ms / 34,2ms |

Tabelle 3: Mittelwert und Standartabweichung von getImage()

Tabelle 3 enthält die Statistisch relevanten Werte. Es ist zu erkennen, dass diese unplausibel sind. So führt zum Beispiel eine Erhöhung der Qualität bei einer Auflösung von 176x144 Pixeln nicht immer zu einem Anstieg der Mittleren Verzögerungszeit. Dies ist dadurch zu erklären, dass keine Beleuchtungsanlage zur Verfügung stand, welche für konstante Lichtverhältnisse sorgte. Diese Lichtschwankungen beeinflussen das Aufgenommene Bild welches sich dadurch besser oder schlechter mit Jpg komprimieren lässt. Dadurch variiert die Datenmenge, die übertragen werden muss. Es kann mit statistischen Mittel aus diesen werten eine Abschätzung getroffen werden wie lange die maximale Verzögerungszeit ist. Die Webcam arbeitet auf 30fps Basis. Dies bedeutet im schlimmsten Fall wird gerade dann ein Bild angefordert wenn die Kamera im Begriff ist ein neues anzufertigen. Daraus folgt die das die Kamera eine maximale Verzögerung von 33,3ms aufweist. Die Standardabweichung gilt so wie sie in Tabelle 3 angegeben ist für ein Intervall von 68,3%. Was bedeutet, dass ein gemessener Wert mit einer Wahrscheinlichkeit von 68,3%. in $\bar{X} \pm S$ liegt. Um ein Intervall von 99,7% zu erreichen muss S mit drei multipliziert werden. Für die maximale Verzögerungszeit gilt also:

$$t_{max} = 33{,}3ms + \bar{X} + 3S$$

| | Auflösung | | | | |
|---|---|---|---|---|---|
| Qualität | | **176x144** | **320x240** | **352x288** | **640x480** |
| | **Niedrig** | 75,8ms | 89,9ms | 176,3ms | 146,8ms |
| | **Mittel** | 80,3ms | 87,5ms | 130,1ms | 162,6ms |
| | **Hoch** | 75,5ms | 91,8ms | 145,1ms | 226,0ms |

Tabelle 4: Maximale Verzögerungszeit

## 4.3 Bewegungsgenauigkeit

Die Bewegungsgenauigkeit wurde mit Hilfe einer Tafel vermessen. Der Rovio legt dabei die beiden entgegengesetzten Strecken zurück; zum Beispiel vorwärts und rückwärts. Anschließend wird die Abweichung zur Ausgangsposition vermessen. Deswegen muss kein Mittelwert gebildet werden und die Standartabweichung berechnet sich wie folgt.

$$S = \sqrt[2]{\frac{1}{n-1}\sum_{i=0}^{n} X_i^{\,2}}$$

Die Standartabweichung setzt sich nun allerdings aus zwei Fehlern zusammen. Um den Einzelfehler zu erhalten wird kann die Standartabweichung einfach halbiert werden. Wie schon bei der Vermessung der Verzögerungszeit gilt auch dieser Wert für ein Intervall von 68,3%. Er muss ebenfalls verdreifacht werden um eine Gültigkeit für ein Intervall von 99,7% zu erreichen.

| Nr. | foreward() backward() | forwardRight() backwardLeft() | forwardLeft() backwardRight() | straightRight() straightLeft() | rotateLeftBySpeed () rotateRightBySpeed() |
|---|---|---|---|---|---|
| 1 | 3 | 1 | 2 | 0 | 0 |
| 2 | 2,5 | 0,5 | 2,5 | 0,5 | 2 |
| 3 | 0 | 1 | 4,5 | 3 | 0,5 |
| 4 | 0 | 0 | 1,5 | 0,5 | 0,5 |
| 5 | 2 | 2 | 1,5 | 1 | 0,5 |
| 6 | 5 | 2,5 | 3,5 | 0 | 1 |
| 7 | 6 | 6 | 3,5 | 1 | 3 |
| 8 | 2 | 2 | 4,5 | 0,5 | 2 |
| 9 | 0 | 3 | 3,5 | 3 | 0 |
| 10 | 0,5 | 1,5 | 2,5 | 0,5 | 1,5 |
| Standartabweichung | 3,1 | 2,7 | 3,3 | 1,5 | 1,5 |
| 0,5 * Standartabweichung | 1,5 | 1,3 | 1,7 | 0,8 | 0,8 |
| Intervall 99,7% | 4,6 | 4,0 | 5,0 | 3,3 | 3,3 |

Tabelle 5: Bewegungsgenauigkeit in cm

Die Messungen In Tabelle 5 wurden bei der niedrigsten Geschwindigkeitsstufe und einer angestrebten Fahrdauer von einer Sekunde durchgeführt. Es ist zu erkennen, dass eine Fahrgenauigkeit von fünf Zentimetern erreicht werden kann. Für die Untersuchung der rotatorichen Bewegungsgenauigkeit macht die Angabe in Zentimetern wenig Sinn. Die Tatsache, dass der Messpunkt sich ungefähr 15cm entfernt von der Drehachse des Roboters befindet, ermöglicht es, durch trigonometrische Rechnung, den Wert annähernd in Grad umzuwandeln. Siehe Tabelle 6.

| Nr. | rotateLeftBySpeed ()<br>rotateRightBySpeed() |
|---|---|
| 1 | 0 |
| 2 | 7,14 |
| 3 | 1,79 |
| 4 | 1,79 |
| 5 | 1,79 |
| 6 | 3,58 |
| 7 | 10,68 |
| 8 | 7,14 |
| 9 | 0 |
| 10 | 5,36 |
| Standartabweichung | 5,4 |
| 0,5 * Standartabweichung | 2,7 |
| Intervall 99,7% | 8,2 |

Tabelle 6: Rotatorische Bewegungsfunktion in Grad

# 5 HowTo Use

In diesem Kapitel soll eine Kurzeinführung gegeben werden, wie das Interface zu benutzen ist. Diese ist in Deutch und in Englisch angefertigt worden.

## 5.1 Deutsch

### 5.1.1 Initialisierung

Die Initialisierung des Rovio läuft relativ einfach ab. Es muss nur die IP-Adresse der Benutzername und das Password dem Konstruktor übergeben werden. All diese Werte müssen vom Typ String sein.

```
Rovio rovio = new Rovio(String ip, String benutzer, String password);
```

Beispiel:

```
Rovio rovio = new Rovio("192.168.2.126","admin","rovio2010");
```

Wenn nicht ganz klar ist wie die Ip-Adressen der Rovios lauten, weil sie z.B. von einem DHCP vergeben wurden, kann eine statische Methode verwendet werden. Diese liefert ein Array von Ip-Adressen zurück. So sind auch Implementierungen möglich, die auf einer variablen Anzahl von Robotern basieren.

```
String ips[] = Rovio.findRovios();
```

### 5.1.2 Benutzung

Das Interface ist in Funktionsgruppen unterteilt. Es existieren zwei verschieden Typen dieser Gruppen. Einmal die Direkten:

```
rovio.video
```

```
rovio.status
```

```
rovio.move
```

Diese Gruppen beinhalten Methoden die einen direkten Aufruf auf dem Rovio auslösen. Dies bedeutet: bei Funktionen die Informationen anfordern, ist mit einer Verzögerungszeit zu rechnen (video, status). Für die Bewegungsfunktion heißt es, dass mindestens alle 200ms ein Bewegungsbefehl (move) geschickt werden muss, um eine kontinuierliche Fahrt sicher zu stellen.

Laden eines Bildes:

```
BufferedImage myImage = rovio.video.getImage();
```

WLan-Status abfragen:

```
int wLanSignalStrength = rovio.status.getWLanSignalStrength();
```

Position Abfragen:

```
RovioPosition myPosition = rovio.status.getPosition()
int x = myPosition.getX()
int y = myPosition.getY()
int theta = myPosition.getTheta()
```

Wobei x und y die vom Navigationssystem bestimmten Koordinaten sind und Theta angibt, in welche Richtung die Front des Rovio zeigt.

Bewegung:

```
rovio.move.forward();
rovio.move.backward();
```

Neben den direkten Befehlsgruppen wurden auch kontinuierliche Befehlsgruppen implementiert:

```
rovio.contVideo
rovio.contStatus
rovio.contMove
```

Bei den Informationen anfordernden Gruppen contVideo und contStatus, wurde ein Mechanismus implementiert, welcher ermöglicht eine Framerate festzulegen, mit der Informationen aktualisiert werden können. Der eigentliche Informationsaufruf hat dadurch kaum Verzögerungszeit. Bei den Bewegungsbefehlen contMove führt ein Befehlsaufruf dazu, dass der Rovio kontinuierlich in die gewünscht Richtung fährt, bis er einen anderen Befehl erhält.

Laden eines Bildes:

```
rovio.contVideo.startContVideo(5);// Start für 5 FPS
BufferedImage myImage = rovio.contVideo.getImage();
rovio.contVideo.stopContVideo();// zum pausieren
```

Status Abfrage:

```
rovio.contStatus.startContStatus(6);// Start für 6 FPS
int wLanSignalStrength = rovio.contStatus.getWLanSignalStrength();
RovioPosition myPosition = rovio.contStatus.getPosition()
rovio.contStatus.stopContStatus();// zum pausieren
```

Bewegungs Befehle:

```
rovio.contMove.startContMove() // Start
rovio.contMove.forward();
rovio.contMove.backward();
rovio.contMove.stopContMove();// zum pausieren
```

### 5.1.3 Beenden

Um die Verbindung zum Rovio zu beenden muss nur die close() Methode ausgeführt werden. Dadurch werden alle Nebenläufigkeiten beendet.

```
rovio.close()
```

## 5.2 English

### 5.2.1 Initialization

The initialization of the Rovio runs relatively simple. It only needs the IP address; the username and the password which have to pass to the constructor. All these values must be of type String.

```
Rovio rovio = new Rovio(String ip, String username, String password);
```

Example:

```
Rovio rovio = new Rovio("192.168.2.126","admin","rovio2010");
```

If not entirely clear what IP addresses Rovios have, for example because it were assigned by a DHCP, a static method can be used. This method provides an array of IP addresses. Implementations are possible based on a variable number of robots.

```
String ips[] = Rovio.findRovios();
```

### 5.2.2 Usage

The interface is divided into functional groups. There are two different types of these groups. First the direct:

```
rovio.video
```

```
rovio.status
```

```
rovio.move
```

These groups include the methods to trigger a direct call to the Rovio. This means: Request information functions have a delay time (video, status). The motion feature means that at least every 200ms a movement command (move) will be sent in order to provide a continuous drive safely.

Loading an image:

```
BufferedImage myImage = rovio.video.getImage();
```

WiFi Status call:

```
int wLanSignalStrength = rovio.status.getWLanSignalStrength();
```

Position call:

```
RovioPosition myPosition = rovio.status.getPosition()
int x = myPosition.getX()
int y = myPosition.getY()
int theta = myPosition.getTheta()
```

Offers the coordinates given by the navigation system x and y and theta indicates the direction in which shows the front of the Rovio.

Motion:

```
rovio.move.forward();
rovio.move.backward();
```

Second the continuous command groups:

```
rovio.contVideo
rovio.contStatus
rovio.contMove
```

The information and requesting groups contVideo contStatus, a mechanism was implemented that allows setting a frame rate to regulate the information updated rate. The interface information call has thus little lag time. The movement command group contMove provides the functionality that the Rovio runs continuously in one direction until it receives another command.

Loading an image:

```
rovio.contVideo.startContVideo(5);// Start für 5 FPS
BufferedImage myImage = rovio.contVideo.getImage();
rovio.contVideo.stopContVideo();// zum pausieren
```

Status call:

```
rovio.contStatus.startContStatus(6);// Start für 6 FPS
int wLanSignalStrength = rovio.contStatus.getWLanSignalStrength();
RovioPosition myPosition = rovio.contStatus.getPosition()
rovio.contStatus.stopContStatus();// zum pausieren
```

Motion:

```
rovio.contMove.startContMove() // Start
rovio.contMove.forward();
rovio.contMove.backward();
rovio.contMove.stopContMove();// zum pausieren
```

### 5.2.3 Close

The Rovio connection can be closed by using the close () method. This will shut down all concurrencies.

```
rovio.close()
```

# 6 Ausblick, Zusammenfassung

Zusammenfasend ist festzustellen, das Interface weist ein gutes Reaktionsverhalten, für die Bewegungsfunktionen, auf. Der Bandbreitenbedarf ermöglicht einen gleichzeitigen Betrieb von 5 bis 50 Robotern; je nach Grafikeinstellungen und sonstiger Last im WLan. Die Bewegungsgenauigkeit… . Für Projekte die eine bewegliche Einheit benötigen, die visuellen Daten ihrer Umgebung liefert, ist der Rovio gut geeignet. Der Einsatz als Trägersystem für Zusatzhardware gestaltet sich allerdings schwierig. So existiert keine offizielle Möglichkeit eine Hardwareerweiterung durchzuführen. Allerdings wäre es möglich, einen zusätzlichen Datenkanal für Erweiterungen über ein separates WLan-Modul zu realisieren. Somit kann der Rovio auch als Trägersystem verwendet werden.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis

[1] Schaubild 2009 WowWee www.wowwee.com
[2] API Specification for Rovio, Version 1.2 ,8 Oktober 2008
http://www.wowwee.com/static/support/rovio/manuals/Rovio_API_Specifications_v1.2.pdf
[3] Javadoc API Version 7
http://java.sun.com/javase/7/docs/api/
[4] Christian Ullenboom, Java ist auch eine Insel (8. Auflage), Galileo Press, 2009, Kapitel: 18.3 Die Klasse URLConnection
[5] Conradt, J., Simon, P., Pescatore, M., and Verschure, PFMJ. (2002). Saliency Maps Operating on Stereo Images Detect Landmarks and their Distance, Internaional Conference on Artificial Neural Networks (ICANN2002), p. 795-800, Madrid, Spain.