

Projektseminar

Eigenentwicklung eines angepassten Motor Shield

Ausgabedatum: 17. Oktober 2012

Abgabedatum: 31. März 2013

verantwortlicher Professor: Prof. Dr.-Ing. habil. Armin Zimmermann

wissenschaftlicher Betreuer: M.Sc. Thomas Dietrich

Eingereicht von: Benjamin Behlau, Lars Vogel

Matrikel-Nr. 46448, 46478

benjamin.behlau@tu-ilmenau.de

lars.vogel@tu-ilmenau.de

Danksagung

Herrn M.Sc. Thomas Dietrich möchten wir für die Betreuung und hilfreiche Unterstützung in allen Phasen des Projektseminars danken. Insbesondere die Freiheiten in der Gestaltung unserer Zeiten sowie dem Aufbau neuer Prototypen mit der einhergehenden Versorgung an jeglichen Arbeitsmaterialen sind hervorzuheben. Zum maßgeblichen Gelingen trug die stets kompetente Begleitung in zahlreichen Angelegenheiten bei.

Besonderer Dank geht an unsere Familienangehörigen und Freunde, die selbst bei außergewöhnlichen Arbeitszeiten wertvollen unterstützenden Beistand geleistet haben und mit Ausdauer sowie Ruhe in jeglicher Situation zum Gelingen beigetragen haben.

Zusammenfassung

Ein Arduino-System (d.h. bestehend aus Arduino ADK und Adafruit Motor Shield), welches zur Steuerung eines handelsüblichen funkferngesteuerten Modellautos eingesetzt wird, soll zur Kostenminimierung durch ein Arduino Uno R3 sowie eine eigens entwickelte Ansteuerung für die beiden Motoren ersetzt werden. Besondere Beachtung ist der Ansteuerungscharakteristik und Dimensionierung der elektrischen Bauteile zu schenken.

Für die vollumfängliche Nutzung der entwickelten Hardware werden die bereits vorhandenen Implementierungen entweder entsprechend angepasst oder bei Notwendigkeit neu entworfen. Das Projekt kann dabei als abgeschlossen betrachtet werden, wenn in einem ausreichenden Testzyklus die Funktionsfähigkeit bestätigt werden kann.

Abstract

An Arduino system (i.e., consisting of Arduino ADK and Adafruit Motor Shield), which will be used to control a commercially available radio-controlled model car, shall be replaced by an Arduino Uno R3 and a specially developed control for the two motors to minimize costs. Special attention has to be paid towards the control characteristics and the dimensioning of electrical components.

For complete usage of the developed hardware already existing implementations will be either adjusted accordingly or redesigned if needed. The project can be considered as completed if the functionality can be confirmed in a sufficient test cycle.

Inhaltsverzeichnis

1 Einleitung und Motivation	1
1.1 Allgemein	1
1.2 Hardware	1
1.2.1 Arduino	1
1.2.2 Arduino Mega ADK	2
1.2.3 Arduino Uno	2
1.3 Software	3
1.3.1 Java-Programm	3
1.3.2 Android Applikation	4
2 Zustandsanalyse	7
2.1 Hardware	7
2.2 Software	8
2.2.1 Arduino Programmierung (C/C++)	8
2.2.2 Applikationen	8
3 Motoransteuerung	11
3.1 Hardwareumsetzung	11
3.1.1 Einleitung	11
3.1.2 Motortreiber	11
3.1.3 Ansteuerung	13
3.2 Arduino	14
3.2.1 Kommunikation	14
3.2.2 Lenkung	16
3.2.3 Geschwindigkeitsregelung	17
4 Applikation	19
4.1 Android	19
4.1.1 USB-Kommunikation	19
4.1.2 Werteparameter	23

4.2 Steuerkonsole	24
4.2.1 Werteparameter	25
5 Fehleranalyse	27
5.1 Kamera- und Auflösungswechsel	27
5.2 Netzwerkverbindung bei wechselnden Access-Points	28
5.3 USB-Verbindungserkennung am Arduino	28
5.4 Positionierung Smartphone	29
6 Zusammenfassung und Ausblick	31
Abbildungsverzeichnis	I
Verzeichnis der Quelltexte	II
Literaturverzeichnis	IV

1 Einleitung und Motivation

1.1 Allgemein

Im Rahmen des Softwareprojekts wurde eine Applikation zur Steuerung eines funkgesteuerten Modellautos unter der Verwendung eines PCs und Smartphones entwickelt. Die Befehle, welche zur Steuerung des Autos vom PC erzeugt werden, sollen die Motoren des Autos ansteuern und schließlich ein Steuern des Fahrzeuges über eine Internetverbindung ermöglichen. Die erzeugten Befehle sollen mithilfe eines Arduinos verarbeitet und als Pegel an die Motoren des Fahrzeuges ausgegeben werden. Um eine Motoransteuerung über Arduino zu ermöglichen, besteht die Möglichkeit, wie in diesem Fall, ein Motorshield zu verwenden. Da am Arduino nur sehr geringe Ströme von einigen wenigen mA ausgegeben werden können, muss das Motorshield diese über einen Motortreiber in verwertbare Signale wandeln. In Kapitel 2.1 wird erläutert, aus welchem Grund der ursprüngliche Aufbau nicht geeignet ist um die im Auto vorhandenen Motoren zu steuern. In den weiteren Kapiteln wird näher auf die Art der Kommunikation zwischen Smartphone und Arduino eingegangen sowie über die Probleme und endgültige Umsetzung der Motoransteuerung. Ergebnis des Projektseminars ist ein selbstentworfenes Motorshield, welches die Ansteuerung des Fahrzeuges und somit die Verwendung als Drohne ermöglicht.

1.2 Hardware

1.2.1 Arduino

Arduino ist eine Open Source-Plattform mit einfach zu nutzender Hard- und Software. Es besteht im Allgemeinen aus einem Microcontroller, welcher die Möglichkeit der Programmierung bietet um das Verhalten für einen gewünschten Zweck anzupassen sowie Ein- und Ausgabe-Pins zur Kommunikation mit seiner Umgebung. Über die Ausgabe von Signalen können Motoren, Lampen oder ähnliches gesteuert werden.

1.2.2 Arduino Mega ADK

Das Arduino Mega ADK baut auf einem Atmel ATmega2560 auf. Android basierte Smartphones bzw. Tablets können über das USB-Host-Interface an das Board gekoppelt werden. Es verfügt über 54 digitale In- und Output-Pins, 16 analoge Input-Pins sowie 4 UART Ports. 14 der digitalen Output-Pins können verwendet werden um PWM-Signale zu erzeugen. Die Betriebsspannung beträgt 5 Volt. Um das Board zu betreiben sollte eine Spannung von 7 bis 12 Volt angelegt werden und dessen Pins liefern selbst einen Gleichstrom von 40 mA. Der Microcontroller arbeitet mit einem 16 MHz-Prozessor und bietet 256 KB zur Programmierung. Um die Stromversorgung des Arduino-Boards sicherzustellen kann entweder die USB-Verbindung oder eine externe Stromversorgung genutzt werden. Falls mehrere Möglichkeiten der Stromversorgung gegeben sind, wird die Quelle automatisch ausgewählt.

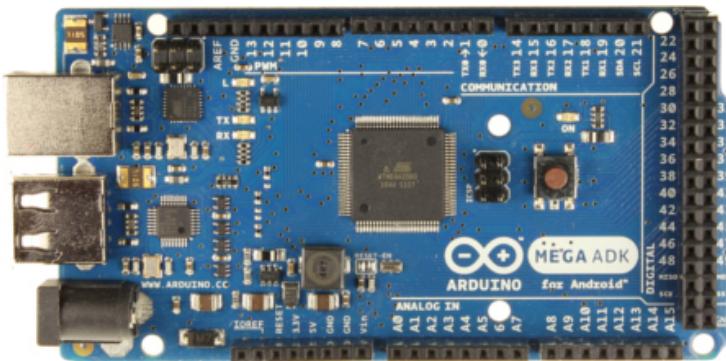


Abbildung 1.1 – Arduino Mega ADK

1.2.3 Arduino Uno

Das Arduino Uno basiert auf einem ATmega328. Es besitzt 14 digitale In- und Output-Pins, davon sind 6 als PWM-Ausgang nutzbar. Dieses Arduino Board verfügt nicht, wie die restlichen Boards, über eine FTDI USB-to-serial Schnittstelle. Stattdessen wird ein Atmega8U2 als USB-to-serial Konverter eingesetzt. In unserem Projekt haben wir das Arduino Uno R3 (Revision3) verwendet. Dieses Board verwendet anstatt eines Atmega8U2 einen Atmega16U2 als USB-to-serial Konverter. Die Betriebsspannung liegt bei 5 Volt. Der Bereich der Eingangsspannung ist mit 6-20 Volt begrenzt, der empfohlene Spannungsbereich liegt zwischen 7 und 12 Volt. An den I/O Pins liegt ein Gleichstrom von 40 mA an. Das

1.3 Software

Board verfügt über einen 16 MHz-Prozessor und bietet hingegen nur 32 KB Speicher zur Programmierung.

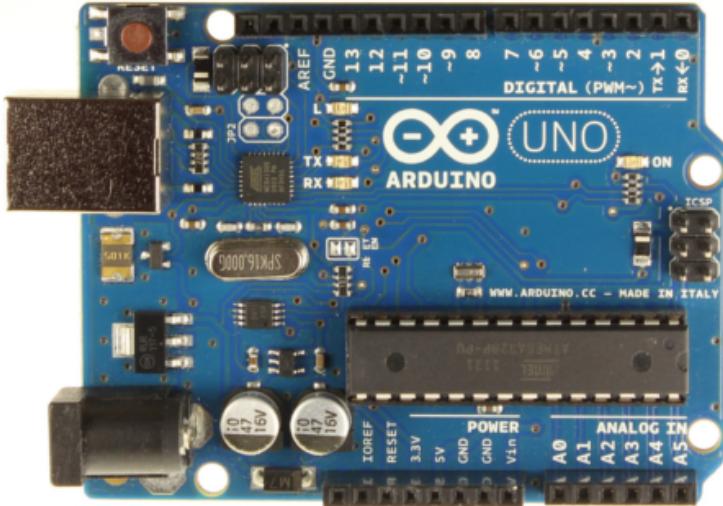


Abbildung 1.2 – Arduino UNO

1.3 Software

Da dieses Projektseminar auf einem Softwareprojekt basiert, wird auch die damit verbundene Software verwendet. Diese Software umfasst 2 Bestandteile: zum einen das Java-Programm, welches auf Seite des PCs Anwendung findet, und zum anderen die Android-Applikation um die Kommunikation zwischen Modellauto und PC zu ermöglichen.

1.3.1 Java-Programm

Das Java-Programm besteht aus einer grafischen Benutzeroberfläche, welche zur Fernsteuerung des Fahrzeuges über das Internet dient. Die größte Fläche nimmt dabei die Darstellung der Bilddaten ein, welche von der Smartphonekamera an den PC gesendet werden. Weiterhin bietet die Software die Möglichkeit Lenkung sowie Geschwindigkeit zu steuern. Die Anpassung wird über Schieberegler realisiert, welche per Maus oder zur Vereinfachung per Tastendruck eingestellt werden können. Um die Geschwindigkeit zu regeln kann man die Tasten 1 bis 5 verwenden, wodurch eine Art Gangschaltung simuliert wird. Mit den Tasten Q und W kann der Lenkwinkel verändert werden. Des Weiteren bietet die Software die Möglichkeit die Auflösung sowie Qualität der zu übertragenden Bilder zu ändern,

um die Datenmenge an die Verbindungsqualität anzupassen. Da das Fahrzeug mit einem Smartphone verbunden ist, besteht die Möglichkeit eine Tonaufnahme am Fahrzeug durchzuführen sowie die Lampe des Telefons, wenn vorhanden, zur Beleuchtung zu nutzen. Um die Position des Fahrzeuges bestimmen zu können, werden die GPS-Daten ausgelesen und können über Tastendruck sogar auf Google-Maps dargestellt werden. Zusätzlich kann am Fahrzeug ein Signalton wiedergegeben werden.

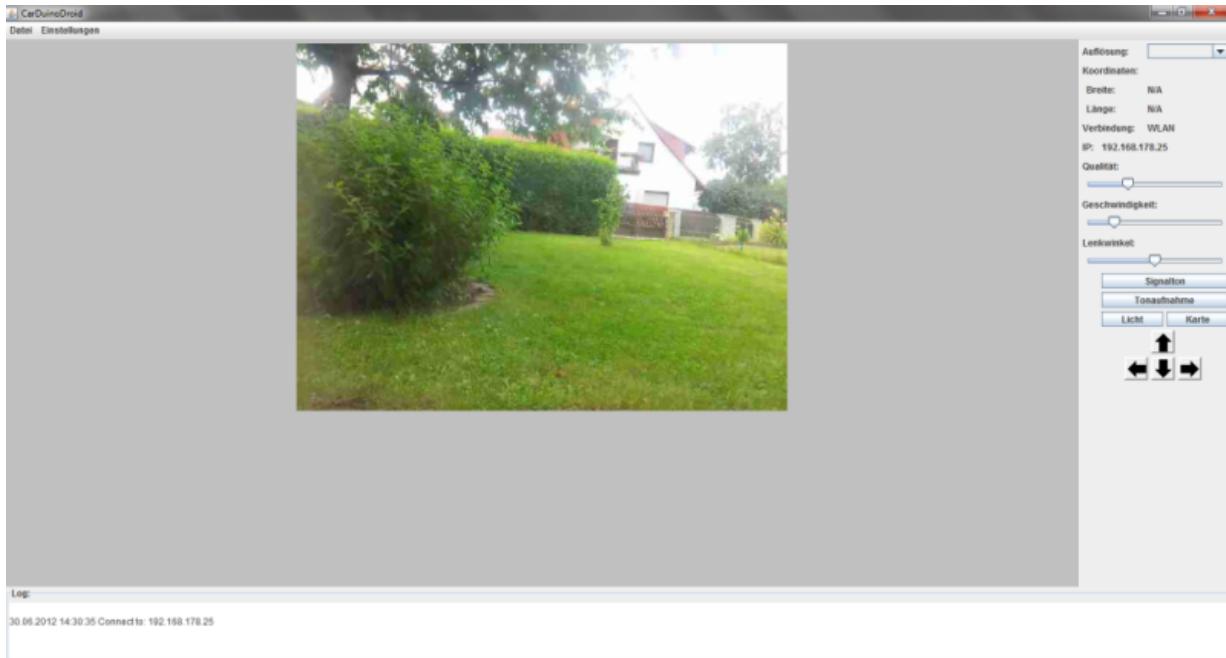


Abbildung 1.3 – Benutzeroberfläche

1.3.2 Android Applikation

Die Hauptaufgaben der Android-Applikation sind sowohl die Steuerdaten, welche vom PC gesendet werden, an das Arduino Board weiterzuleiten, als auch die Bilddaten des Smartphones zum PC zu senden. Zusammengefasst ist die Applikation die Steuerzentrale für die Kommunikation zwischen Auto und PC. Das Telefon stellt Mikrofon und Lampe zur Verfügung, welche über die Software verwendet werden können. Die Steuerdaten werden über den USB-Port an das Board weitergegeben, welches mithilfe dieser Daten die 2 Motoren ansteuern kann. Die Netzwerk-Kommunikation wird über 3 Sockets realisiert. Da die Steuerdaten über den Downlink und die Bilddaten über den Uplink gesendet werden, kann es im Regelfall nicht zu einer gegenseitigen Behinderung kommen. Die Applikation sendet Bestätigungen an das Java-Programm zurück, ob die gesendeten Befehle korrekt

1.3 Software

ausgeführt wurden. Im Laufe des Projektseminars musste diese Applikation an das Arduino Uno angepasst werden, da dieses, wie oben erwähnt, nicht über eine FTDI USB-to-serial Schnittstelle verfügt. Das hat zur Folge, dass den USB-Host das Smartphone darstellt.



Abbildung 1.4 – Carduinodroid-App

2 Zustandsanalyse

Zur Ermittlung der grundlegenden Vorgehensweise ist eine eingehende Auseinandersetzung mit dem Ist- sowie folglich Soll-Zustand notwendig. Die Analyse der derzeitigen Situation dient als ausschlaggebender Punkt zur Ausarbeitung des Zeitmanagements in den folgend angeführten Themenkomplexen. Mit einer Abweichungsanalyse werden alle Änderungen klassifiziert.

2.1 Hardware

Zur weiteren Kostenminimierung erfolgt das Ersetzen eines Arduino ADK durch ein Arduino Uno sowie die Bereitstellung einer eigens entwickelten Ansteuerung der Motoren. Für die Mikrocontroller-Umstellung wird das Hauptaugenmerk auf die Art des verwendeten USB-Treibers gelegt. Eine Veränderung vom „FTDI USB-to-serial“-Chip-Treiber mit Arduino als USB-Host hin zu einem „USB-to-serial“-Konverter wird softwareseitige Anpassungen zur Folge haben (siehe Kapitel [2.2](#)).

Die bisherige Umsetzung zur Ansteuerung der Servo- und DC-Motoren wurde über ein handelsübliches Arduino „Motor Shield“, welches mit einem L293D als Motortreiber ausgestattet ist, gewährleistet. Bedingt durch die niedrigen Kenngrößen im Bereich der Dauer- und Spitzenstrombelastung von höchstens 600 mA bzw. 1.2 A überhitzt der Baustein und weist veränderte Eigenschaften auf. Neben der Geschwindigkeitssteuerung wird weiterhin eine reibungsfreie Manövriertfähigkeit benötigt, die bereits in der vorhandenen Umsetzung in vollem Umfang gestellt wird. Die Entwicklung und Konzeption einer Erweiterungsplatine für das Arduino Uno, welche die vorher genannten Eigenschaften aufweist, steht im Fokus des Projektseminars.

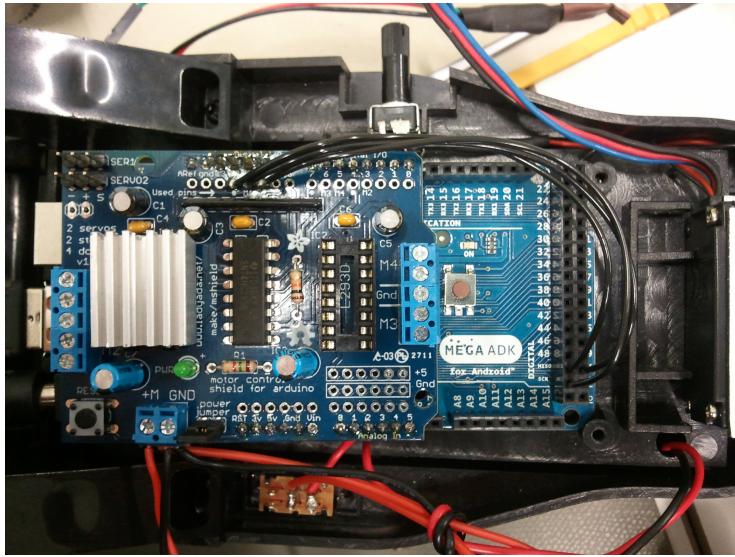


Abbildung 2.1 – Arduino Mega ADK mit Motor Shield

2.2 Software

2.2.1 Arduino Programmierung (C/C++)

Ein Quelltext mit den grundlegenden Funktionen hinsichtlich der Arduino Programmierung im Bereich digitaler sowie analoger Signale wird durch das Fachgebiet zur Verfügung gestellt.

Die Anpassung muss hinsichtlich einer USB-Informationsübertragung und deren Auswertung erfolgen, aber auch alle hardwareseitigen Inputs bedürfen einer Regulierung. Mit einer persistenten Prüfung der seriellen Datenverbindung erfolgt die Sicherstellung eines Notstopps bei Verbindungsverlust. Weiterhin soll der Faktor zur einfachen Anpassung auf Motoren mit spezifischen Werten im Vordergrund stehen. Eine Variabilisierung ist somit von Vorteil.

2.2.2 Applikationen

Android

Die bisherige Kommunikation zwischen dem androidfähigen Mobiltelefon und Arduino ADK wurde über die USB-Host-Funktion seitens des Mikrocontrollers gewährleistet. Zu-

2.2 Software

dem bietet das Accessory Development Kit (Abk. ADK) alle möglichen Bibliotheken, die wichtige Implementierungsdetails übernehmen.

```
1 import com.android.future.usb.UsbAccessory;
2 import com.android.future.usb.UsbManager;
3
4 mUSBManager = UsbManager.getInstance(this);
5 UsbAccessory acc = mUSBManager.getAccessoryList()[0];
6
7 if (!mUSBManager.hasPermission(acc)) return;
```

Quelltext 2.1 – ADK Initialisierung

Während der Erstellung eines USB-Manager-Objektes (Siehe Quelltext 2.1) werden alle wichtigen Übertragungsdetails festgelegt, wobei man sich auf Defaultwerte (bspw. Baudrate 9600) einigt oder bestimmte eigene Abstimmungen vornehmen kann. Wenn die Verbindung aufgebaut ist, erfolgt bei der Accessory-Kommunikation der Datenaustausch per FileInputStream- und FileOutputStream (Siehe Quelltext 2.2).

```
1 ParcelFileDescriptor mFD = mUSBManager.openAccessory(acc);
2
3 if (mFD != null) {
4     FileDescriptor fd = mFD.getFileDescriptor();
5     mIS = new FileInputStream(fd); // use this to receive messages
6     mOS = new FileOutputStream(fd); // use this to send commands
7 }
```

Quelltext 2.2 – ADK FileStream

Mit der Umstellung auf ein Arduino Uno muss die Funktion des zentralen Host-Controllers auf das Mobiltelefon übergehen. Mit USB On-The-Go (Abk. USB OTG) ausgerüstete Geräte werden, wie die Anpassung der Android-Version auf 4.0.3 (3.0 bei Tablet ausreichend), ein wichtiger Bestandteil für die Integration des USB-Host-Modus.

Java Steuerkonsole

Durch das vorangegangene Softwareprojekt „CarDuinoDroid“ konnte ein ausreichend dokumentierter Quelltext zur Verfügung gestellt werden. Der Grundfunktionsumfang zur Steuerung der Drohne kann in seiner Gesamtheit beibehalten werden. Einzig einige Parameter bezüglich der Geschwindigkeit und Lenkwinkel bedürfen einer Anpassung.

3 Motoransteuerung

3.1 Hardwareumsetzung

3.1.1 Einleitung

Um das Fahrzeug zu steuern, müssen beide Fahrzeugmotoren vom Arduino-Board angesteuert werden. Die Ansteuerung für die Lenkung kann direkt umgesetzt werden. Diese läuft über einen Servo-Motor, welcher über den Arduino angesteuert werden kann. Das eigentliche Problem stellt die Ansteuerung des DC-Motors für die Geschwindigkeitsregulierung dar. Die Problematik besteht darin, dass das Arduino-Board nur Ströme im mA-Bereich ausgibt, was jedoch nicht ausreicht um das Fahrzeug in Bewegung zu versetzen. Der ursprüngliche Versuch dieses Problem zu lösen war die Verwendung eines Arduino Motor Shield. Diese Lösung erwies sich allerdings als ungeeignet. Der Grund wurde im Laufe des Projektseminars durch Ausmessen der Ströme klar. Als weitere Realisierungsmöglichkeiten wären eine Optokoppler-Schaltung oder eine selbst entworfene Transistorschaltung möglich gewesen. Wir haben uns dafür entschieden das Grundkonzept des gegebenen Motor Shields beizubehalten und demzufolge einen geeigneten Motortreiber mit entsprechender Schaltung zu finden und einzubinden.

3.1.2 Motortreiber

Das im Softwareprojekt verwendete Motor Shield ist mit einem Motortreiber des Typs L293D von ST Microelectronics ausgestattet. Jedoch ist dieser Motortreiber für den vorhandenen DC-Motor nicht geeignet. Der Motor benötigt im normalen Betrieb einen Dauerstrom von 1400-1900 mA und bei Beschleunigungs- sowie Bremsvorgängen bis zu 3 A bei einer Betriebsspannung von bis zu ca. 9 V. Der L293D ist allerdings nur für Dauerstrombelastung von 600 mA und Stromspitzen bis 1,2 A pro Kanal ausgelegt. Demzufolge können die benötigten Werte auch nicht per Parallelschaltung der internen Transistoren erreicht werden, weshalb eine geeignetere Dimensionierung gefunden werden muss. Eine

erhebliche Einschränkung stellt die Versorgungsspannung von bis zu ca. 9 V dar, da die Ströme bei einer solch geringen Spannung eher ungewöhnlich sind, aber vom verwendeten Akkumulator nur zur Verfügung gestellt werden. Nach einiger Suche konnten wir die Auswahl auf zwei Bauteile begrenzen: den L298N von ST und den Si9987 von Vishay. Aus dem Datenblatt kann man folgern, dass man mindestens drei parallelgeschaltete Si9987 benötigt um den Motor anzusteuern. Laut Datenblatt liefert der Si9987 einen maximalen Spitzenstrom von 1,5 A und 1 A Dauerstrom, welcher jedoch mit steigender Temperatur abnimmt. Bei unseren Versuchen den Motor mit diesem Treiber anzusteuern hat das Bauteil allerdings eine Temperatur erreicht, bei der das elektrische Verhalten nicht mehr den Vorgaben entsprach. Aus diesem Grund ist die endgültige Wahl schließlich auf den L298N gefallen. Der Vorteil bei dieser Lösung ist, dass nur ein Bauteil benötigt wird. Dieser Treiber besitzt zwei spiegelgleiche H-Brückenschaltungen, welche im Normalfall zur Ansteuerung von zwei separaten Motoren gedacht sind. Allerdings bietet er zudem die Möglichkeit beide Schaltkreise parallel zu verbinden und somit einen Motor mit mehr Strom zu versorgen. Die Parallelschaltung der beiden Schaltkreise ergibt einen zulässigen Spitzenstrom von bis zu 4 A.

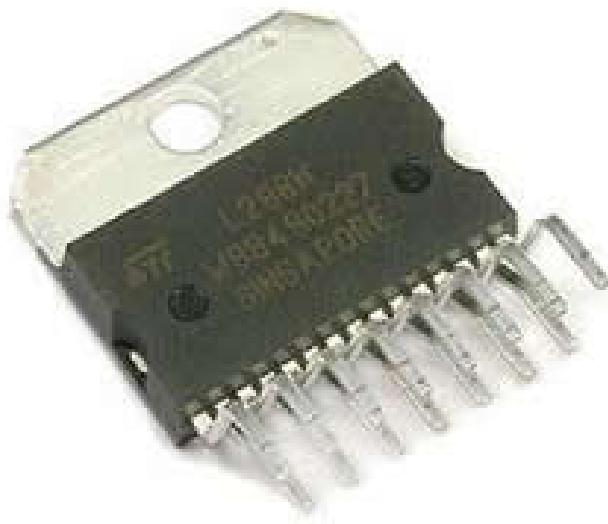


Abbildung 3.1 – L298N

3.1 Hardwareumsetzung

3.1.3 Ansteuerung

Die Ansteuerung des Servo-Motors für die Lenkung lässt sich relativ einfach realisieren. Der Motor muss einerseits mit der 5V-Spannung des Arduino-Boards versorgt und andererseits der Lenkwinkel per PWM-Ausgang eingestellt werden. Zur Vervollständigung muss nur noch das Erdungskabel mit dem GND-Anschluss am Arduino verbunden werden. Um den DC-Motor anzusteuern verwendeten wir letztendlich, wie oben angeführt, den L298N. Um diesen parallelgeschaltet zu verwenden, müssen mehrere Pins paarweise zusammengeschaltet werden (Pin-Belegung siehe Bild): Output 1 mit Output 4, Output 2 mit Output 3, Input 1 mit Input 4 sowie Enable A mit Enable B. Sense A und Sense B müssen zu Masse geführt werden. Die Steuersignale des Motors werden an den Outputs ausgegeben, welche demzufolge an die Anschlüsse des Motors geführt werden müssen. Zum Vorwärtsfahren müssen an den Inputeingängen 1 und 4 ein High-Pegel und an den Eingängen 2 und 3 ein Low-Pegel anliegen, zum Abbremsen oder Rückwärtsfahren müssen die Pegel hingegen vertauscht werden. Die Steuerungssignale zur Geschwindigkeitsregulierung müssen allerdings an die Enable-Eingänge gelegt werden. Vs und Vss dienen zur Spannungsversorgung.

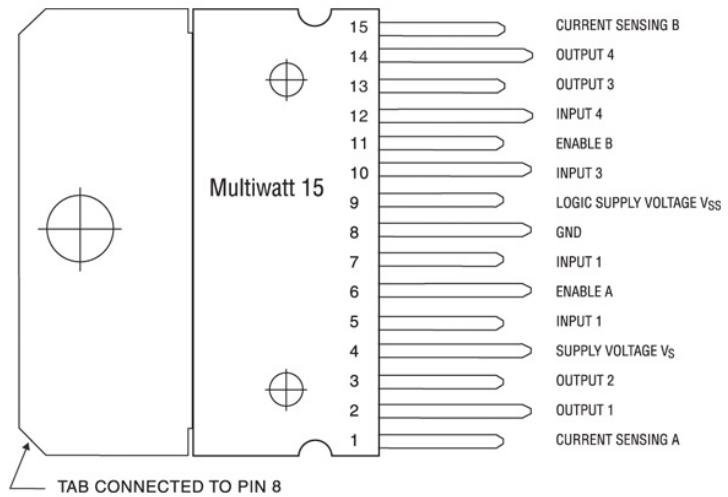


Abbildung 3.2 – L298N Beschaltung

Da in diesem Bauteil selbst keine Schutzdiode integriert sind, müssen zwischen Motor und Motortreiber noch zusätzlich vier Freilaufdioden geschaltet werden um auftretende Induktionsströme abzufangen und insbesondere die In-/Outputs des Arduino vor Schäden zu bewahren.

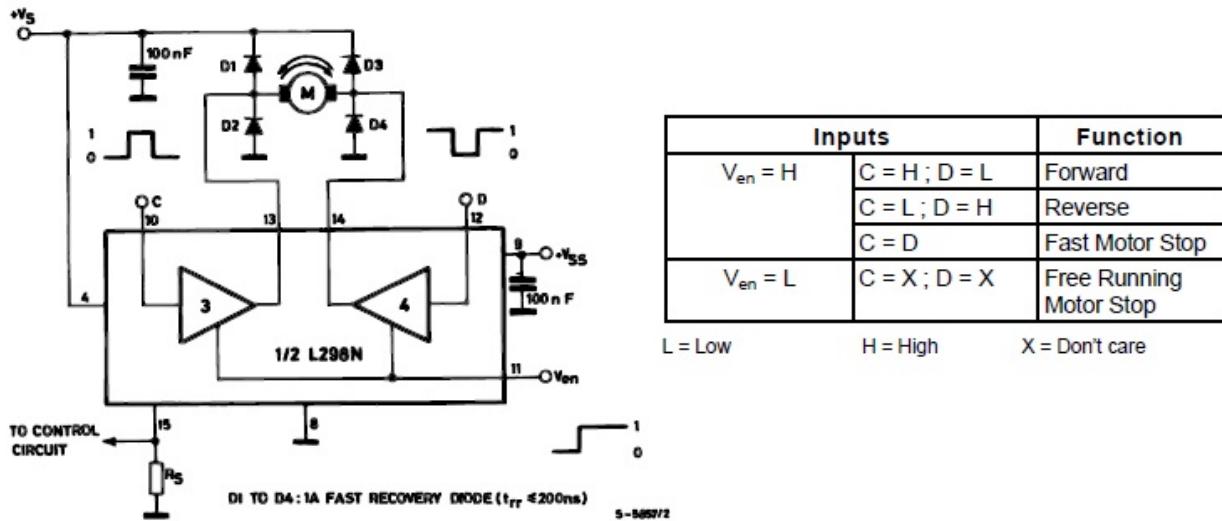


Abbildung 3.3 – Beschaltung von L298N und DC-Motor

3.2 Arduino

3.2.1 Kommunikation

Die hier betrachtete Kommunikation (siehe Quelltext 3.1) findet zwischen dem Arduino-Board und den Motoren statt. Die gesendeten Zahlenwerte vom Smartphone müssen an dieser Stelle in für die Motoren verwertbare Signale umgewandelt werden. D.h. sie werden sinnvoll umgerechnet um mit ihrer Hilfe ein geeignetes PWM-Signal erzeugen zu können. Die Werte werden als Bitstream an das Board übertragen und dort in ein Array geschrieben. Im Quelltext des Arduinos müssen zunächst einige Festlegungen getroffen werden. Mit `int m1Pin=2` wird beispielsweise der Wert dieser Variable an den Pin 2 des Boards gelegt.

Im folgenden Programmteil (siehe Quelltext 3.2) wird die Baudrate auf 9600 festgelegt, da es in der Android-Applikation so vorgegeben wird. Der Servo-Motor erhält seine Signale entsprechend der Festlegung über den Pin 6 am Arduino. Der Winkel, der standardmäßig

3.2 Arduino

```
1 void set_direction_Car(int direction, int driving_distance);
2
3 int trans_pos = 8;
4 int min_degree = 50; //Min Degree 50(-90) - 90 means the wheels are aligned
5 int max_degree = 130; //Max Degree (90-)130 - both sides with an angle of 40
6 degree
7
8 int step_degree;
9 int stepsizes_dc;
10 int m1Pin = 2;
11 int m2Pin = 4;
12 int m3Pin = 3;
13 int transition_dc = 9;
```

Quelltext 3.1 – Initialisierung Arduino

beim Einschalten des Arduinos eingestellt wird, beträgt 87° . Dieser Wert ergab sich durch ausgiebige Testfahrten, da wir feststellen mussten, dass die Räder des Fahrzeugs nicht exakt gerade stehen. Wir testeten das Fahrverhalten mit verschiedenen Einstellungen und legten schließlich den Wert 87 als Standardeinstellung fest, bedingt durch die geringste Abweichung von einer Optimalgeraden. Weiterhin wird definiert, wie stark der Lenkwinkel pro Schritt geändert wird. Aus Sicherheitsgründen wurde zudem eine Methode zum Stoppen des Fahrzeugs implementiert, die nach dem Setup eine bewegungsfreie Drohne gewährleistet.

```
1 void setup()
2 {
3     Serial.begin(9600); //
4     servo_motor.attach(6);
5     servo_motor.write(87);
6
7     step_degree = (90 - min_degree)/(trans_pos);
8
9     stop_Car();
10 }
```

Quelltext 3.2 – Setup Arduino

Um die Motoren steuern zu können muss der empfangene Bitstream weiter verarbeitet werden. Zu Anfang wird geprüft, ob eine serielle Verbindung vorliegt. Sofern das nicht der Fall ist, wird das Fahrzeug gestoppt. Ist eine Verbindung vorhanden, wird gewartet, bis 4 Bit im Puffer zur Verfügung stehen. Diese werden dann zur Weiterverarbeitung in ein Array geschrieben. Die Bedeutung der einzelnen Stellen des Arrays wird in den jeweiligen Abschnitten erläutert.

```

1 void loop()
2 {
3     if(Serial){
4         int x;
5         int i=0;
6         int values[4];
7         // motorStop(); // Motor Stop
8         if(Serial.available() >= 4)
9         {
10             while (i<4) {
11                 values[i] = Serial.read();
12                 i++;
13                 // Serial.println(values[i-1]);
14             }
15             set_direction_Car(values[3], values[2]);
16             set_speed_DC(values[1], values[0]);
17             // Serial.println(atoi(direction)+atoi(Value));
18         }
19     }else{stop_Car();}
20 }
```

Quelltext 3.3 – Hauptroutine – Loop

3.2.2 Lenkung

Dieser Teil des Codes realisiert die Einstellung des Lenkwinkels durch einen Servo-Motor. Die Stellen 2 und 3 des Arrays dienen zur Steuerung der Lenkung. Diese 2 Parameter unterteilen sich wie folgt: Stelle 2 für den Lenkwinkel und Stelle 3 für die Richtung, also Rechts oder Links. Liegt ein Signal an, wird der Winkel entsprechend der Parameter variiert. Sobald kein Lenksignal (d.h. beide Parameter besitzen den Wert 0) mehr übertragen wird, stellt der Motor die Ausgangslage ein.

```

1 void set_direction_Car(int direction, int steps)
2 {
3     if(abs(steps)<=8)
4     {
5         if(direction == 1)
6             servo_motor.write(87 + steps*step_degree);
7         else
8             servo_motor.write(87 - steps*step_degree);
9     }
10    else
11        servo_motor.write(87);
12 }
```

Quelltext 3.4 – Lenkung

3.2.3 Geschwindigkeitsregelung

Ähnlich wie die Lenkung funktioniert auch die Ansteuerung des DC-Motors. Die Stellen 0 und 1 repräsentieren die Werteparameter für die Geschwindigkeit mit Richtung. Die 0. Stelle des Arrays gibt den Wert für die Beschleunigung an, die 1. Stelle dagegen die Richtung, vorwärts oder rückwärts, an. Liegt ein Signal an, wird die Richtung und Geschwindigkeit gemäß der Parameter geändert. Liegt kein Signal an (d.h. beide Parameter besitzen den Wert 0), wird das Fahrzeug gestoppt, was über ein Ausrollen zum Schutz des Motortreiber geschieht. Dazu müssen alle Pins, die der Motor abgreift, auf Low gesetzt werden. M1Pin und m2Pin liegen an den jeweiligen Inputs des Motortreibers an, m3Pin dagegen gibt das Steuersignal in Form eines PWM-Signals auf den Enable-Eingang des Treibers.

```

1 void set_speed_DC( int direction , int speed ){
2     if (( speed > 9) || (speed < 1))
3     {
4         stop_Car();
5         return;
6     }
7
8     if (direction == 1)
9     {
10        digitalWrite(m1Pin,HIGH);
11        digitalWrite(m2Pin,LOW);
12        analogWrite(m3Pin,75+speed*20);
13    }
14    else
15    {
16        digitalWrite(m1Pin,LOW);
17        digitalWrite(m2Pin,HIGH);
18        analogWrite(m3Pin,75+speed*20);
19    }
20}
21
22
23 void stop_Car()
24 {
25     digitalWrite(m1Pin,LOW);
26     digitalWrite(m2Pin,LOW);      // stopped
27     digitalWrite(m3Pin,LOW);
28 }
```

Quelltext 3.5 – Geschwindigkeitsregelung

4 Applikation

Die Symbiose zwischen Android-Applikation und Java Steuerkonsole ist ein wichtiger Bestandteil für die Funktionsfähigkeit der Drohne aus Konsumelektronik. Anhand einer Zustandsanalyse wurden die notwendigen Veränderungen herausgearbeitet und die Gestaltungsgrundlage durch das Hauptseminar „Elektronische Ein- und Ausgabe mit Arduino“ (Kühne, Brehme 2012) gelegt.

4.1 Android

Als Herzstück und zugleich Steuerzentrale befindet sich die Android-Applikation im Mittelpunkt jeglicher Datenallokation sowie -übermittlung. Nicht nur die Steuerkommandos, sondern auch Bilddaten, GPS-Auswertung und Tonaufnahmen werden verwaltet.

4.1.1 USB-Kommunikation

Bei der sinnvollen Integration des USB-Controller-Objektes wird der erste Versuch zur Verarbeitung von Bewegungsinformation im „Controller__Android“ als Ausgangspunkt genutzt. Dabei war es uns wichtig, keine vorzeitige Instanziierung vorzunehmen, um einer Verschwendungen von Ressourcen vorzubeugen (siehe Quelltext 4.1).

```
1 public class Controller__Android
2 {
3     private static UsbController sUsbController;
4     ... case 1:{           if(sUsbController == null){
5         sUsbController = new UsbController(ACTIVITY,
6             mConnectionHandler , VID, PID, log , this );}
```

Quelltext 4.1 – Platzierung der Objekterstellung

Zudem verhindert die Fallabfrage zum Vorhandensein eines „sUsbController“ eine weitere

Erzeugung des Objektes und ermöglicht die punktgenaue Nutzung. Mit der Festlegung der Instanziierung des Objektes „mConnectionHandler“ muss auch jenes für die Verwendung beschrieben werden. Es dient der Abfrage des Zustandes unserer USB-Verbindung während der Erstellung, die aus diversen Gründen nicht funktionsfähig sein könnte (siehe Kapitel 5.3). Unter anderem betrachten wir dabei die Fälle eines fehlenden USB-Endgerätes (hier: Arduino Uno R3), den Ausfall bzw. das vorzeitige Stoppen oder Vorhandensein einer aufgebauten USB-Verbindung (siehe Quelltext 4.2).

```

1 private final IUsbConnectionHandler mConnectionHandler = new
2     IUsbConnectionHandler() {
3         public void onUsbStopped() {
4             log.write(LOG.INFO, "USB Connection gestoppt");
5         }
6
6 public void onErrorLooperRunningAlready() {
7     log.write(LOG.INFO, "Looper läuft schon");
8 }
9
10 public void onDeviceNotFound() {
11     if(sUsbController != null){
12         sUsbController.stop();
13         sUsbController = null;
14     }
15     log.write(LOG.INFO, "Device nicht gefunden");

```

Quelltext 4.2 – mConnectionHandler Definition

Dementsprechend muss das Interface „IUsbConnectionHandler“ die jeweiligen Methoden abstrakt vordeklarieren um die Instanziierung realisieren zu können (siehe Quelltext 4.3).

```

1 public interface IUsbConnectionHandler {
2
3     void onUsbStopped();
4     void onErrorLooperRunningAlready();
5     void onDeviceNotFound();

```

Quelltext 4.3 – Objekt IUsbConnectionHandler

Als wichtigster Bestandteil zur Gewährleistung einer fehlerfreien USB-Kommunikation auf Basis des USB-Host-Modus (Abk. USB OTG) dient die Klasse „USBController“. Der enthaltene Konstruktor (siehe Quelltext 4.4) weist einerseits alle Parameter für die Suche des richtigen USB-Gerätes zu (Product ID - Abk. PID und Vendor ID - Abk. VID) und initialisiert wichtige Funktionen. Darunter befindet sich „getApplicationContext()“. Es bietet die Möglichkeit der Nutzung von Broadcasting, aber auch die Implementierung eines Vendor

4.1 Android

Service wie in unserem Beispiel. Allgemeiner ermöglicht es den Zugriff auf anwendungs-spezifische Ressourcen und Klassen sowie „up-calls“ für Applikations-Level-Operationen wie beispielsweise Start-Aktivitäten oder das Senden und Empfangen von Anweisungen. Die nachfolgende Anweisung „getSystemService(Context.USB_SERVICE)“ erlaubt zudem einen Zugang zum USB-Manager mit Zugriff auf USB-Geräte als Host, der für unsere Kom-munikation essentiell ist.

```
1 public UsbController(Activity parentActivity, IUsbConnectionHandler
2   connectionHandler, int vid, int pid, LOG Log, Controller.Android
3   controller_android) {
4     controller.Android = controller_android;
5     mApplicationContext = parentActivity.getApplicationContext();
6     mUsbManager = (UsbManager) mApplicationContext.getSystemService(Context.
7       USB_SERVICE);
```

Quelltext 4.4 – Konstruktor USBController

Für den Zugriff auf ein gewisses USB-Gerät muss dieses nicht nur in einer Geräteliste zur Verfügung stehen, sondern es erfordert auch die Vergabe von Zugriffsrechten (siehe Quell-text 4.5). Folglich nutzen wir die Methode „enumerate“ (zu deutsch: aufzählen) als Suche des geforderten Arduino Uno R3 über die VID sowie PID nach der Rechtevergabe. Somit erreichen wir nicht nur einen logischen Verbund von Suche und Zuweisung, sondern auch eine entsprechend richtige Abarbeitungsreihenfolge.

```
1 enumerate(new IPermissionListener() {
2   public void onPermissionDenied(UsbDevice d) {
3     UsbManager usbman = (UsbManager) mApplicationContext.
4       getSystemService(Context.USB_SERVICE);
5     PendingIntent pi = PendingIntent.getBroadcast(
6       mApplicationContext, 0, new Intent(ACTION_USB_PERMISSION),
7       0);
8     mApplicationContext.registerReceiver(mPermissionReceiver,
9       new IntentFilter(ACTION_USB_PERMISSION));
10    usbman.requestPermission(d, pi);
```

Quelltext 4.5 – Rechteanforderung

Die Festlegung des zu nutzenden Arduino Uno R3 wird über eine HashMap realisiert. Mit dem USB-Manager können alle angeschlossenen Geräte abgefragt und anschließend in einer verketteten Liste zusammengeführt werden. Um schrittweise alle Einträge zu über-prüfen wird eine while-Schleife verwendet, welche iterativ nach den gesuchten PID und VID

Angaben abfragt. Bei erfolgreichem Durchlauf wird ein Handler gestartet, der für unsere Datenübertragungsroutine verantwortlich sein wird (siehe Quelltext 4.6).

```

1 HashMap<String , UsbDevice> devlist = mUsbManager . getDeviceList () ;
2 Iterator<UsbDevice> deviter = devlist . values () . iterator () ;
3
4 while ( deviter . hasNext ()) {
5     UsbDevice d = deviter . next () ;
6     if ( d . getVendorId () == VID && d . getProductId () == PID ) {
```

Quelltext 4.6 – HashMap zur USB-Geräte Wahl

Gemäß Kühne und Brehme (2012: 29 f.) „(. . .) ist es wichtig, dass die Kommunikationsroutine nicht im selben Thread wie die Hauptroutine läuft. Dadurch werden Verklemmungen vermieden. Mit Hilfe der Klassen Thread und Runnable kann ein neuer Thread erstellt werden, der ausschließlich für die Kommunikation verantwortlich ist.“ Auf Basis dessen erfolgt eine Ausgliederung der Übertragung von Kommunikationsdaten (siehe Quelltext 4.7).

```

1 private void startHandler ( UsbDevice d ) {
2     if ( mLoop != null ) {
3         mConnectionHandler . onErrorLooperRunningAlready () ;
4         return ;
5     }
6     mLoop = new UsbRunnable ( d ) ;
7     mUsbThread = new Thread ( mLoop ) ;
8     mUsbThread . start () ;
9 }
```

Quelltext 4.7 – Handler starten

Das Objekt „USBRunnable“ wird als „mLoop“ instanziert um eindeutig einen Verweis auf die ablaufende Endlosschleife der Datenübertragung zu erreichen (siehe Quelltext 4.8). Als wichtiger Parameter wird das gewünschte Arduino Uno R3 als „UsbDevice“ übergeben, damit Daten und Nachrichten sowohl gesendet, als auch empfangen werden können.

```

1 UsbRunnable ( UsbDevice dev ) {
2     mDevice = dev ;
3 }
```

Quelltext 4.8 – Thread für Datenübertragung instanzieren

Nach dem Öffnen der USB-Verbindung erfolgt das Versenden eines „ControlTransfer“-Strings, welcher an den gewählten Endpunkt einen kontrollierten Datenstrom versendet

4.1 Android

(siehe Quelltext 4.9). Dieser dient zudem zur Synchronisation der Übertragungsgeschwindigkeit auf 9200 Baud. Abschließend wird der genaue Kommunikationsendpunkt definiert, damit der Datenstrom zielgerichtet ankommen kann.

```
1 public void run() {
2     UsbDeviceConnection conn = mUsbManager.openDevice(mDevice);
3
4     conn.controlTransfer(0x21, 34, 0, 0, null, 0, 0);
5     conn.controlTransfer(0x21, 32, 0, 0, new byte[] { (byte) 0x80,
6                     0x25, 0x00, 0x00, 0x00, 0x00, 0x08 }, 7, 0)
7
8     UsbEndpoint epOUT = null;
9     UsbInterface usbIf = mDevice.getInterface(1);
10    ...
11    epOUT = usbIf.getEndpoint(i);
```

Quelltext 4.9 – Festlegung Baudrate und Geräteendpunkt

In einer weiteren internen For-Schleife wird nach der Initialisierung der Übertragungsgeräte und -punkte die Verbindung realisiert. Dabei nutzt man eine Variable „mData“, die über eine Methode an die zu vermittelnden Informationen angepasst und per BulkTranser übertragen werden kann (siehe Quelltext 4.10). Einerseits erfolgt eine Überprüfung, ob die gewünschten Daten bereits übermittelt wurden, und ob der Transfer abgeschlossen ist. Das sichert eine Überlagerung sowie Änderung von Daten während der USB-Kommunikation und die mögliche unerwünschte Doppelung von Datenaustauschvorgängen.

```
1 if (!controller.Android.GetSent()){
2     controller.Android.SetSent(true);
3     conn.bulkTransfer(epOUT, mData, mData.length, 0);
4 }
```

Quelltext 4.10 – Übertragung von Daten

4.1.2 Werteparameter

Die einzelnen Daten auf Seiten des Arduino werden im Byte-Format empfangen. Um den komplizierten Umwandlungsprozess auf der Hardware zu vermeiden, wird vor der Zusammensetzung entweder eine „Byte.parseByte()“-Umwandlung für Zahlenwerte oder „(byte)(BoolWert?1:0)“ für boolsche Werte durchgeführt (siehe Quelltext 4.11).

```

1 comando[0] = Byte.parseByte(parts[1]);
2 comando[1] = (byte)(front?1:0);

```

Quelltext 4.11 – Umwandlung nach Byte

4.2 Steuerkonsole

Die grafische Benutzeroberfläche (Abk. GUI) dient als zentraler Mittelpunkt zur Steuerung der Drohne. Insbesondere die Anpassung der wichtigen Bewegungsparameter zur Sicherstellung der USB-Datenübertragung zwischen dem androidfähigen Mobiltelefon und Arduino Uno R3 ist ein wichtiger Bestandteil.

Zur Veranschaulichung des weitreichenden Problems muss die Art der Kommunikation zwischen den jeweiligen Bestandteilen beleuchtet werden. Über die BufferedReader sowie -Writer-Funktion wird die Netzwerk-Übertragung zwischen der Steuerkonsole und der Android-Applikation initialisiert. Dabei ist es möglich Worte in jeglicher Kombination aus dem ASCII-Code zu versenden. Mit der Verbindung von Comma-separated Values (Abk. CSV) als strukturierte Übertragung von Daten besteht kein Problem der vollständigen Zuordnung.

Dem gegenüber steht die USB-Verbindung zum Arduino Uno R3, die entsprechend ihrer Namensgebung digitale Daten bitweise hintereinander überträgt. Der festgelegte Puffer am Arduino kann anhand des Füllstandes neuer Daten abgefragt werden, wobei im voraus nicht zu erkennen ist, welchen Inhalt diese enthalten werden. Somit erfolgt unsererseits eine Normierung der Datenmenge auf 4 Bit und foglich die Anpassung der gesamten Parameter (siehe Quelltext 4.12) auf jeweils eine Länge von 1 Bit seitens der Steuerkonsole (speed, up, angle, right).

```

1 private void send_controlsignal(int speed, int angle){
2     if (controller_computer.network.send_controlsignal(speed+";"+up+";"+
3         angle+";"+right))
4         feedback_output();
}

```

Quelltext 4.12 – Parameter

4.2.1 Werteparameter

Die Berechnung der Werteparameter erfolgt über einen einfachen Grundsatz:

$$\frac{100}{X} \leq Y$$

Man definiere X als die zu suchende Variable und Y als die Anzahl der Stufen, die man wählen möchte. Zudem wird der Definitionssraum von Y auf $[1,y]$ mit $y \in \text{natürliche Zahlen}$ beschränkt (in unserem spezifischen Fall $y \leq 9$). Die einhergehende Umstellung der Formel zur Berechnung der Stufen nach X ermöglicht die Festlegung des genannten Parameters anhand der Anzahl der gewünschten Stufen.

```
1 private int SpeedCalculation(int speed)
2 {
3     if (up || down)
4         return (int) ((speed / 12.5) + 1); // Geschwindigkeitsstufen 1-9
5     else return (int) (0);
6 }
```

Quelltext 4.13 – Geschwindigkeitskalkulation

Der Quelltext 4.13 wird in der Klasse „Car_Controller“ verwendet, um die vorher beschriebene Vorgehensweise zur Festlegung der Stufen einzubinden. Sofern eine höhere Stufenanzahl benötigt wird, bestände auch die Möglichkeit Zahlen mit vorangestellter Null darzustellen, um eben die Festlegung auf eine genannte Bit-Länge zu gewährleisten.

5 Fehleranalyse

Während eines ausführlichen Testzyklus konnten einige Probleme hinsichtlich der Programmierung und Sicherheit festgestellt werden. Das Vorgehen während der Analyse zur Sicherstellung einer einwandfreien Funktionsfähigkeit begann mit der Erweiterung des bereits vorhandenen Testzyklus aus dem vorangegangenen Softwareprojekt „CarDuinoDroid“ (SS12). Bereiche der Erweiterung sind:

- Kamera- und Auflösungswechsel
- Netzwerkverbindung bei wechselnden Access-Points
- USB-Verbindungserkennung Arduino
- Positionierung Smartphone

5.1 Kamera- und Auflösungswechsel

Insbesondere auf Geräten des Herstellers Samsung können Probleme in der Umstellung der Auflösung festgestellt werden (im Speziellen: Abstürze). Bei der ersten Version der Software führte bereits der Aufbau einer Bildübertragung zum Absturz der Android-Applikation, wobei die Nutzung auf Sony- sowie HTC-spezifischen Testsystemen zu keinerlei ähnlichen Anzeichen führte. Zur Analyse der Fehlerherkunft wurde der Ablauf zum Aufbau und zur Änderung der Bildübertragung schrittweise ausgewertet. Dabei konnte festgestellt werden, dass der „Resolution_Listener“ bei Befüllung aller möglichen Auflösungsvarianten ein „ActionEvent“ erkannte. Folgend wurde die darin enthaltene Implementierung ausgeführt, die wiederum zu einer Änderung der voreingestellten Auflösung führte. Mit der Abfrage, ob ein Befüllungsprozess durchgeführt wurde, wird genau diese Aktivierung des ActionListeners verhindert (siehe Quelltext [5.1](#)).

```
1 if (!controller_Computer.gui_computer.getFilled()){  
2     controller_Computer.camera_settings.send_change_resolution(resolution);}  
3 controller_Computer.gui_computer.setFilled(false);
```

Quelltext 5.1 – Befüllungsprozess sichern

5.2 Netzwerkverbindung bei wechselnden Access-Points

In der Testumgebung des Zusebaus an der Technischen Universität Ilmenau erfolgt die Wahl eines Access Points entsprechend der Empfangsqualität. Diese Wechsel können gemäß der Auslastung im Netzwerk problematisch sein. Zudem gibt es einige undefinierte Bereiche, in denen scheinbar kein Empfang möglich ist, die aber im Weiteren nicht beleuchtet werden. Sofern die Verbindung nicht aufrecht erhalten werden kann, erfolgt in unregelmäßigen Abständen ein nicht bewusst reproduzierbarer Absturz der Android-Applikation.

5.3 USB-Verbindungserkennung am Arduino

In der ersten Version der Umsetzung zur Ansteuerung einer Drohne aus Konsumelektronik wird der verbaute Nickel-Metallhybrid-Akkumulator nicht nur zur Versorgung der Motoren, sondern auch des Arduino Uno R3 benötigt. Die größere Belastung zeigt besonders hinsichtlich der Nutzungsdauer Wirkung. Mit ca. 30 bis 45 Minuten Dauerbelastung erreicht der elektrische Speicher seine Grenzen.

In 90 Prozent der Fälle tritt das vorher genannte Problem eines entladenen Akkumulator auf und führt zum Stillstand des „CarDuinoDroid“. Die restlichen 10 Prozent werden durch einen entladenen Mobiltelefon-Akkumulator abgedeckt. Die Folgen sind dabei so weitreichend, dass durch fehlende Steuerungsbefehle eine Endlosbewegung zum letzten bekannten Signal erfolgt. Der Grund liegt in einem nicht zu erkennenden USB-Verbindungsabbruch auf dem Arduino, der weiterhin seine serielle Übertragung als „gegeben“ annimmt und keinen Motorstopp durchführt. Die möglichen Gefahren für Umgebung und die Drohne an sich besitzen ein hohes Potential.

5.4 Positionierung Smartphone

Aus eigener Erfahrung muss für die weitere Entwicklung die Positionierung des Smartphones in den Vordergrund gerückt werden. Mit der bisherigen Lösung über einen Klettverschluss am Heck des Modellautos als Befestigungsmittel kann bei Rückwärtsfahrten das Schadensrisiko nicht auf ein minimales Maß reduziert werden. Oftmals hohe Geschwindigkeiten und optische Beeinträchtigung durch die Nutzung der Kameras führen zu Fehleinschätzungen im Bremsvorgang. Foglich besteht keine Sicherung eines optimalen Abstandes zu Objekten.

6 Zusammenfassung und Ausblick

Der Grundgedanke zur Herstellung einer Drohne aus Konsumelektronik wird insbesondere mit der Umstellung auf die kostengünstigere Variante eines Arduino Uno R3 sowie der vollumfänglichen Bewegungssteuerung realisiert. Dieser Weg zur Minimierung hoher Produktionskosten um ein Vielfaches sowie die Reduzierung des Eigenanteils in der hardwareseitigen Aufbereitung auf die Einbringung eines Vierquadrantenstellers (bzw. Motortreiber) zeugt von der Einfachheit einer eigenen Umsetzung. Laut Larry Page (CEO Google) sind aktuell mehr als 750 Millionen Geräte weltweit aktiviert (Google Official Blog 2013) und das Potential zur Nutzung dieser Umsetzungsvariante gegeben. Die Entwicklung von stark angepassten, eingebetteten Systemen für Dronennutzer benötigt nicht nur ein hohes Maß an Kapital, sondern auch die Weiterentwicklung sowie Integration neuer Möglichkeiten und Funktionen. Mit einem androidfähigen Mobiltelefon wird jedem Nutzer zugleich ein großes Repertoire an Sensoren und Funktionen gegeben. Mit der Ausbreitung von Mobilfunknetzen und deren Standards gibt es für viele Gebiete eine Kommunikationsmöglichkeit die dem Satelliten-Standard mit ca. 4 bis 10 MBit in nichts nachstehen wird (außen vor gelassen: die Exklusivität von Dronenverbindungen im Einsatz bspw. der NASA und folglichen Geschwindigkeiten). Mit der Einbeziehung der 4. Generation des Mobilfunkstandards, zu dem auch LTE gezählt wird, erreicht man Übertragungsraten von ca. 100 MBit. Wenn man abschließend diese Raten noch dem High Definition Format von 1080p (1920 * 1080, 30 Bilder/s) gegenüberstellt, so wird ersichtlich, dass mit voranschreitendem, flächendeckendem Ausbau des LTE-Netzes selbst die ehemals notwendigen Qualitätsminderungen zur Sicherung von Geschwindigkeiten ad absurdum geführt werden.

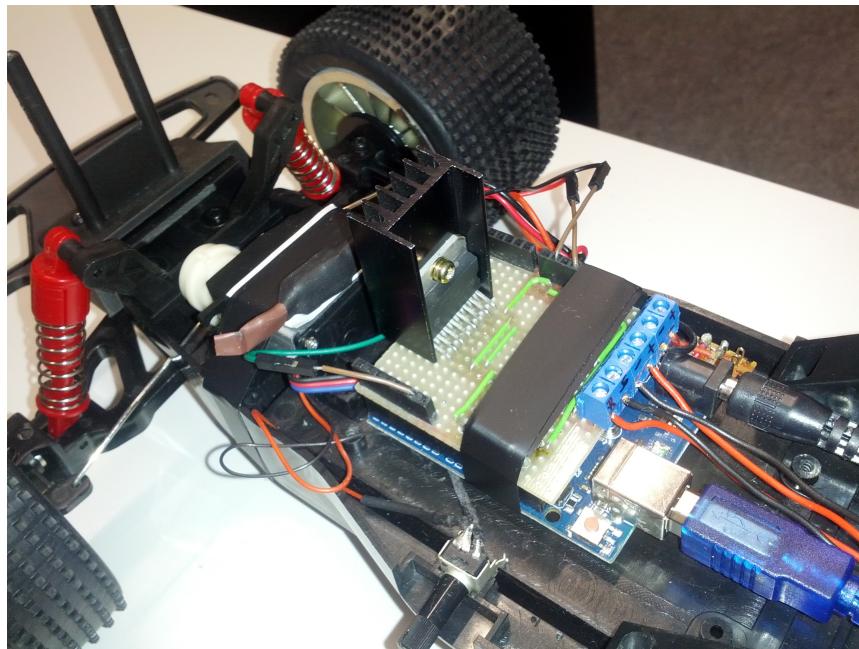


Abbildung 6.1 – Entwurf des eigenen Motor Shields

Ein denkbarer Ansatz zur Erweiterung wäre der Aspekt von Sensorik wie Ultraschallverfahren zur Entfernungsmessung oder Beschleunigungssensoren für Feedback in Form von Vibration. Setzt man den Schwerpunkt dagegen softwareseitig, wäre eine Steuerung über ein weiteres Mobiltelefon erstrebenswert. Die Einsparung in Transportgewicht und der Gewinn an Mobilität sind dabei nicht zu verachten. Somit sind für zukünftige Weiterentwicklungen auf Basis der bisherigen Umsetzung des CarDuinoDroid die Grenzen nur durch das Mobiltelefon selbst gesetzt.

Abbildungsverzeichnis

1.1	Arduino Mega ADK	2
1.2	Arduino UNO	3
1.3	Benutzeroberfläche	4
1.4	Carduinodroid-App	5
2.1	Arduino Mega ADK mit Motor Shield	8
3.1	L298N	12
3.2	L298N Beschaltung	13
3.3	Beschaltung von L298N und DC-Motor	14
6.1	Entwurf des eigenen Motor Shields	32

Verzeichnis der Quelltexte

2.1	ADK Initialisierung	9
2.2	ADK Filestream	9
3.1	Initialisierung Arduino	15
3.2	Setup Arduino	15
3.3	Hauptroutine - Loop	16
3.4	Lenkung	16
3.5	Geschwindigkeitsregelung	17
4.1	Platzierung der Objekterstellung	19
4.2	mConnectionHandler Definition	20
4.3	Objekt IUsbConnectionHandler	20
4.4	Konstruktor USBController	21
4.5	Rechteanforderung	21
4.6	HashMap zur USB-Geräte Wahl	22
4.7	Handler starten	22
4.8	Thread für Datenübertragung instanzieren	22
4.9	Festlegung Baudrate und Geräteendpunkt	23
4.10	Übertragung von Daten	23
4.11	Umwandlung nach Byte	24
4.12	Parameter	24
4.13	Geschwindigkeitskalkulation	25
5.1	Befüllungsprozess sichern	28

Literaturverzeichnis

- [A1313a] ARDUINO: Arduino ADK. <http://arduino.cc/en/Main/ArduinoBoardADK>. Version: März 2013
- [A1313b] ARDUINO: Arduino Uno. <http://arduino.cc/en/Main/ArduinoBoardUno>. Version: März 2013
- [A1313c] ARDUINO: Serial. <http://arduino.cc/en/Reference/Serial>. Version: März 2013
- [A1313d] ARDUINO: Tutorial PWM. <http://www.arduino.cc/en/Tutorial/PWM>. Version: März 2013
- [AD113a] ANDROID DEVELOPERS: Accessory Development Kit 2012 Guide. <http://developer.android.com/tools/adk/adk2.html>. Version: März 2013
- [AD113b] ANDROID DEVELOPERS: USB Host. <http://developer.android.com/guide/topics/connectivity/usb/host.html#discovering-d>. Version: März 2013
- [AP113] ANDROID PLAYGROUND: Arduino and Java. <http://playground.arduino.cc/Interfacing/Java>. Version: März 2013
- [APH13] ANTIPASTO HARDWARE WIKI: How-to use a serial port with Android. <http://antipastohw.pbworks.com/w/page/41729078/How-to%20use%20a%20serial%20port%20with%20Android,%20Liquidware%20Ambrosia%20edition>. Version: März 2013
- [AS113a] ANDROID SERVERBOX: Android USB Host + Arduino: How to communicate without rooting your Android Tablet or Phone. <http://android.serverbox.ch/?p=549>. Version: März 2013
- [AS113b] ANDROID SERVERBOX: Creating a Serial to USB driver using the Android USB-Host API. <http://android.serverbox.ch/?p=370>. Version: März 2013

Literaturverzeichnis

- [AT113a] *ARDUINO TUTORIAL: Motorsteuerung direkt per Arduino.* <http://www.arduino-tutorial.de/2010/06/motorsteuerung-direkt-per-arduino/>. Version: März 2013
- [AT113b] *ARDUINO TUTORIAL: Motorsteuerung mit einer MOS-FET Brücke.* <http://www.arduino-tutorial.de/2010/06/motorsteuerung-mit-einer-mos-fet-brucke/>. Version: März 2013
- [BK113] *BLAIR KELLY: Arduino Wifly Mini.* <http://www.blairkelly.ca/2012/04/20/arduino-wifly-mini/>. Version: März 2013
- [CH113] *CHRISTOPH'S HIMBEERE: Motorsteuerung mit L298 und Raspberry Pi.* <http://www.christophs-himbeere.de/tutorials/motorsteuerung-mit-l298-und-raspberry-pi/>. Version: März 2013
- [DCN13] *DAMEN CNC: Bild L298N.* <http://www.damencnc.com/images/products/L298N.jpg>. Version: März 2013
- [EC113] *ELECTRO-CIRCUIT: Schema L298N.* <http://electro-circuit.net/wp-content/uploads/2011/10/Konfigurasi-Driver-Hbridge-L298.jpg>. Version: März 2013
- [GC113] *GOOGLECODE: usb-serial-for-android.* <https://code.google.com/p/usb-serial-for-android/source/browse/README.md>. Version: März 2013
- [GH113] *GITHUB: Arduino-Communicator.* <https://github.com/jeppsson/Arduino-Communicator/tree/master/src/com/primavera/arduino/listener>. Version: März 2013
- [GOB13] *GOOGLE OFFICIAL BLOG: Update from the CEO.* <http://googleblog.blogspot.ca/2013/03/update-from-ceo.html>. Version: März 2013
- [HE113] *HOBBY ELECTRONICS: Arduino UNO Tutorial 2 - Servos.* <http://www.hobbytronics.co.uk/arduino-tutorial2-servos>. Version: März 2013
- [IPC13] *ITP PHYSICAL COMPUTING: DC Motor Control Using an H-Bridge.* <http://itp.nyu.edu/physcomp/Labs/DCMotorControl>. Version: März 2013
- [JW113] *JAYWAY: Arduino UNO and the Google ADK – part I.* <http://www.jayway.com/?p=9419/>. Version: März 2013
- [KB12] *KÜHNE, Sascha ; BREHME, Sebastian: Hauptseminar - Elektronische Ein- und Ausgabe mit Arduino.* TU Ilmenau, 2012

- [MC113] *MIKROCONTROLLER: H-Brücken Übersicht.* http://www.mikrocontroller.net/articles/H-Br%C3%BCcken_%C3%9Cbersicht. Version: März 2013
- [RT113] *ROBOTER-TEILE: Datasheet L298N.* <http://www.roboter-teile.de/datasheets/l298N.pdf>. Version: März 2013
- [SO113] *STACKOVERFLOW: Communicating with Arduino from Android.* <http://stackoverflow.com/questions/9768296/communicating-with-arduino-from-android>. Version: März 2013
- [STM13] *STMELECTRONICS: Datasheet L293D.* <http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000059.pdf>. Version: März 2013
- [VS113] *VISHAY SILICONIX: Datasheet SI9987.* <http://www.vishay.com/docs/70864/si9987.pdf>. Version: März 2013

Eidesstattliche Erklärung

Wir, Benjamin Behlau, Lars Vogel, Matrikel-Nr. 46448, 46478, versichern hiermit, dass wir unser Projektseminar mit dem Thema

Eigenentwicklung eines angepassten Motor Shield

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben, wobei wir alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet haben. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ilmenau, den 2013/03/25

BENJAMIN BEHLAU, LARS VOGEL