Bau und Programmierung einer Fernsteuerung

mit zwei Arduinos und Funkmodulen als Transmitter und Receiver

Felix Viola, im Fach Informatik

Herr Faßbender, IF Leistungskurs, 2016

Inhaltsverzeichnis

1. Einleitung

Nachdem meine letzte Fernsteuerung für mein Fernlenkauto / Modellauto kaputt gegangen war, überlegte ich, durch welche Fernsteuerung ich die alte ersetzte. Dabei suchte ich eine Fernsteuerung mit möglichst vielen Anschlüssen, um zum Beispiel nicht nur den Fahrtenregler und den Lenk-Servo, wie bei der alten anzuschließen, sondern auch noch zwei weitere Servos, wodurch zum Beispiel eine Kamera oder in Hinsicht auf noch kommende Projekte insgesamt vier Motoren gleichzeitig gesteuert werden könnten. Genauso sollte die neue Fernsteuerung mehrere Lichter ein- bzw. ausschalten können, wodurch auch die Möglichkeit offen bleibt, mit diesem Ein-/Aus-Signal die Aufnahme mit einer Kamera zu starten beziehungsweise zu stoppen.
Leider führte eine kurze Recherche im Internet zu einer Enttäuschung, da es eine den Wünschen entsprechende Fernsteuerung erst ab mindestens 100€ zu erwerben gibt. Daraufhin überlegte ich mir, ob es nicht eine günstigere Alternative gäbe und kam zu dem Entschluss mir selber eine Fernsteuerung zu bauen und programmieren. Dabei war ich natürlich auch an der Frage interessiert, wie der Bau und die Programmierung einer Fernsteuerung mithilfe Drahtloser Kommunikation aussähe. Also fing ich mit einer einfachen Recherche an und fand heraus, dass ich für die Drahtlose Kommunikation zwischen dem Transmitter und dem Empfänger nur zwei günstige Funkmodule zu kaufen bräuchte und für die restliche Umsetzung besaß ich bereits die meisten Teile, jedoch brauchte ich noch zwei Arduinos, um die Funkmodule auch ansteuern zu können. Eine Idee, wie man das programmieren könnte, hatte ich auch schon.
Meine genauen Kriterien für die Fernsteuerung waren, dass sie mindestens vier Servos / Fahrtenregler ansteuern und vier Lichter ein- bzw. ausschalten können muss. Dazu sollte dieses Projekt nicht teurer als 30€ werden. Außerdem würde ich mir wünschen, dass die Fernsteuerung einfach reproduzierbar ist, um diese bei möglichen zukünftigen Projekten nutzen zu können. Eine parallele Nutzung mehrerer Fernsteuerungen wird in dieser Umsetzung leider nicht möglich sein, jedoch habe ich bereits eine Idee, wie dies möglich werden könnte. Auf diese Idee werde ich im meinem Fazit zu dem Projekt noch ein wenig genauer eingehen.

2. Theoretischer Teil

Um eine Fernsteuerung für ein Modellauto aus zwei Arduinos zu bauen, - einen zur Datenübermittlung und einen zur Umsetzung der empfangenen Daten im Auto - werden einmal die Arduinos selbst gebraucht, natürlich das Auto mit seinem Lenk-Servo und dem Fahrtenregler zur Regulierung der Geschwindigkeit, jedoch auch eine Möglichkeit der drahtlosen Übermittlung der Daten, wie schnell und in welche Richtung das Auto fahren soll.
Der Arduino kann die Signale erzeugen, um den Servo und den Fahrtenregler anzusteuern, verfügt selber aber über keine Möglichkeit eine Drahtlose Kommunikation mit einem anderen Arduino / Gerät herzustellen. Dafür bieten sich Funkmodule an, die einmal als Transmitter und einmal als Receiver fungieren. Zusätzlich besteht die Möglichkeit Anschlüsse für Lichter mit in das Projekt einzubauen, da man damit Lichter Ein und Ausschalten kann, jedoch auch Erweiterungen wie zum Beispiel einen Raspberry Pi mit Ein/Aus-Signalen (für die Videoaufzeichnung etc.) Signale geben kann.

2.1 Verwendete Module, Bauteile und deren Funktionsweisen

Im folgenden werden die einzelnen Module und Bauteile beschrieben, die benötigt werden, um eine gute Fernsteuerung zu entwickeln. Zu den Teilen gehört der Arduino, die beiden Funkmodule, der Fahrtenregler und der Lenk-Servo des Autos.

2.1.1 Arduino

Der Arduino ist ein sehr günstiges (OpenSource-) Entwicklungs-Kit, basierend auf einem Atmel-Mikroprozessor. Auf dem Board sind alle Teile bereits verbaut, die gebraucht werden um einen Mikroprozessor zu betreibenden, wodurch man sich nur um die Programmierung kümmern muss.
Den Arduino gibt es in verschiedenen Ausführungen (am populärsten ist der Arduino Uno). In diesem Fall wird der „Arduino Pro Mini“ für die Fernsteuerung verwendet, da er sehr klein, sehr gut auf einer Platine zu verbauen und schon für 7€ zu habe ist. Ein

Arduino Uno wird als die Empfänger-Einheit verwendet, da dieser vom Mikrocontroller her mehr Möglichkeiten als der Pro Mini bietet.
Der Arduino lässt sich mit der für ihn entwickelten Arduino IDE (auch OpenSource) in der Programmiersprache C/C++ programmieren.

2.1.2 Funkmodule

Die beiden verwendeten Funkmodule sind das Wichtigste am gesamten Projekt. Sie bilden die Kommunikations-Ebene der beiden Arduinos. Dabei sind diese beiden Funkmodule die günstigsten und (für den Preis) die Module mit der größten Reichweite (bis zu 100m in Gebäuden mit einer kleinen Antenne), die es auf dem Markt gibt.
Sie verwenden die Frequenz 433MHz, die in Deutschland für das Modellfahren und den Modellflug vorgesehen ist.
Als Modulationsverfahren verwenden die Module „ASK“ (Amplitude-Shift Keying). Bei diesem Verfahren wird die Amplitude der Funkwelle variiert um Bits zu versenden/empfangen. Im einfachsten Fall wird also bei einer hohen Amplitude eine „1“ und bei einer niedrigen eine „0“ übermittelt. Mit einer höheren Menge an verschiedenen Amplitudenstärken können auch mehrere Bits übermittelt werden. Bei 4 unterschiedlichen Amplituden erhält man damit 2 Bits („00“ - „01“ - „10“ und „11“).

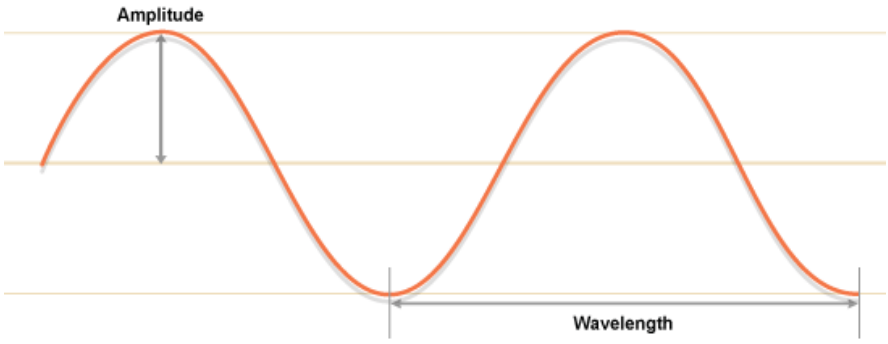

Bild 5.1

2.1.3 Lenk-Servo

Bei dem Lenk-Servo handelt es sich um einen gewöhnlichen Servo, dessen Bewegung wird durch die Mechanik im Auto in eine Lenkbewegung übersetzt.
Ein Servo besteht aus einem Elektromotor (links unten im Bild 5.1), einem Potentiometer (am rechten Rand in der Mitte) und einer Steuerungsplatine (unten rechts), wobei durch die Zahnräder eine Umdrehung des Motors so übersetzt wird, dass

sich der Servo-Arm am Ende nur ein kleines Stück bewegt, wodurch die Bewegungen besser kontrolliert werden können.

Die Steuerungsplatine bekommt über ein PWM-Signal die gewünschte Position des Servoarms übermittelt. Dieses Signal wird in eine Spannung übersetzt und dann mit der Spannung durch das Potentiometer verglichen. Der Motor wird solange nach Rechts oder Links gedreht, bis die Spannung durch das PWM-Signal mit der Spannung durch das Potentiometer übereinstimmen (Das Potentiometer wird durch den Motor mit gedreht, wodurch sich die Spannung ändert).

Der hier verwendete Servo hat eine Grundfrequenz von 20ms und einer Pulslänge von einer Millisekunde (nach links gedreht), über 1,5 Millisekunden (Mittlere Stellung) bis zwei Millisekunden (nach rechts gedreht).

Das Anschlusskabel des Servos besteht aus drei Phasen. Diese sind einmal 5V+ (rot), Ground (braun) und das Signal (orange).

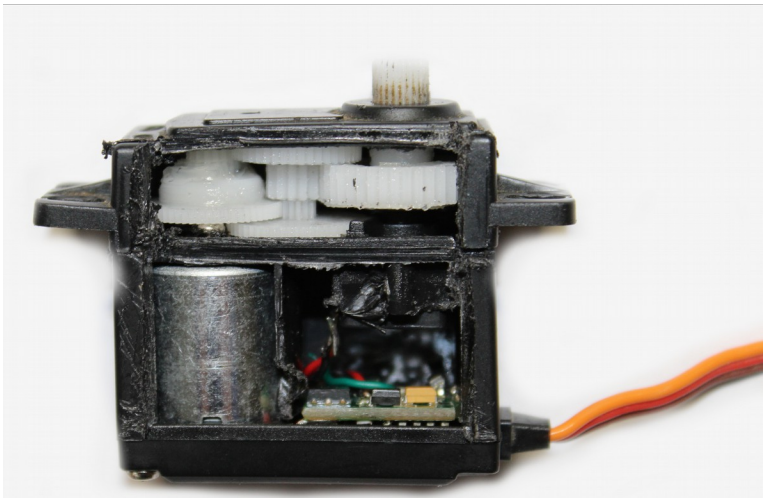

Bild 6.1

2.1.4 Fahrtenregler

Damit ein Modellauto nicht nur mit Vollgas Vorwärts/Rückwärts fahren kann (was nicht ganz soviel Spaß bei schnelleren Autos macht), gibt es Fahrtenregler, die es ermöglichen, die Spannung, die der Motor bekommt, zu regulieren. Dabei ist auch der Akku des Autos direkt an den Fahrtenregler angeschlossen. Der Fahrtenregler macht aus den 12V des Akkus 5V, die für die restliche Elektronik zur Verfügung steht.

Ein Fahrtenregler verwendet wie der Lenk-Servo ein PWM-Signal, mit dem der Stromzufluss zu dem verwendeten Motor reguliert wird.

Das Anschlusskabel ist (genau wie bei dem Servo) in drei Phasen eingeteilt. Dabei ist

5V+ wieder das rote Kabel, Ground jedoch am schwarzen und das Signal an dem weißen Kabel.

3. Praktischer Teil

Nach dem Vorstellen und Erklären von Arduino, Auto und den Funkmodulen, die für das Entwickeln einer Fernsteuerung nötig sind, macht es Sinn diese auch zusammenzuführen und zu programmieren.

3.1 Zusammenbau der Module

Der Zusammenbau lässt sich in zwei Teile unterteilen. Einmal in den Transmitter / die Fernsteuerung und einmal in den Receiver / den Empfänger im Auto.

3.1.1 Zusammenbau des Transmitters

Der Zusammenbau des Transmitters stellt sich einfach dar. Durch die Vorgabe der im Modellauto vorhandenen Standards für die Steckverbindungen müssen Dreier-Pins verwendet werden, bei den der Linke Pin das Signal ausgibt und der Mittlere und rechte Pin Vcc und Ground sind.

Da ein PWM-Signal gebraucht wird kommen die Digitalen Pins 3,5,6,9,10 und 11 in Frage, wobei ich mich hierbei für die ersten vier von den genannten entschieden habe. Vcc und Ground werden einfach mit den entsprechenden Pins vom Arduino verbunden.

Die Anschlüsse für das Licht haben die selben Standards wie die Anschlüsse der Servos/Fahrtenregler. Dabei muss jedoch kein PWM-Signal zur Verfügung gestellt werden, sondern lediglich Strom auf den mittleren Pin. Die analogen Pins bieten sich dafür an, da diese sich auf der anderen Seite des Arduinos befinden und damit der Bau einer Platine vereinfacht wird. Ausgewählt habe ich hierbei die Pins A0 bis A3.

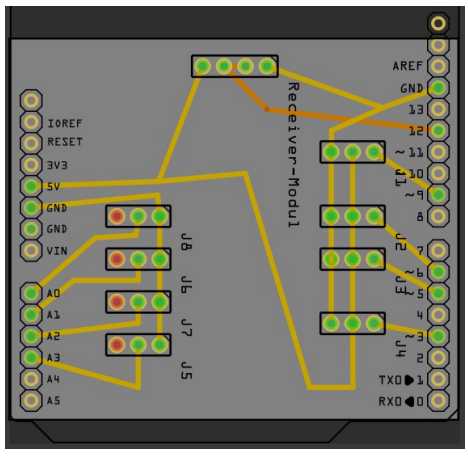

Bild 7.1

Das Receiver-Modul muss mit Vcc und Ground „versorgt“ werden. Da die beiden Data-Pins das selbe ausgeben, habe ich diese verbunden und mit dem Digitalen Pin 12 des Arduinos verbunden.

Der Strom, der gebraucht wird, um den Arduino zu betreiben, wird dabei durch den Fahrtenregler zur verfügung gestellt.

3.1.2 Zusammenbau des Receivers

Der Receiver hingegen ist ein wenig komplizierter, da hier nicht nur die Position der vier Potentiometer in zwei Joy Sticks ausgelesen werden muss, sondern auch noch die Zustände der vier Schalter, um die Lichter ein und aus zu schalten beziehungsweise den Raspberry Pi ein An-/Aussignal zu übermitteln, abgefragt werden müssen.

Die Schalter (im Bild 8.1 durch Taster dargestellt) muss erst mal mit Strom versorgt werden, um dann am (hier gewählten) digitalen Pin zu schauen, ob Strom ankommt oder nicht. Um ein genaues Signal zu bekommen ist es außerdem nötig an den „Ausgang“ des Schalters einen „Pull-Down“ Widerstand (10kOhm) mit dem Ground zu verbinden. Die Schalter habe ich in diesem Fall an die Digitalen Pins 2-5 angeschlossen.

Die Potentiometer (auch 10kOhm) werden einmal mit dem Ground und einmal mit 5V+ verbunden (an die linken und rechten Pins). Durch drehen am Potentiometer wird der

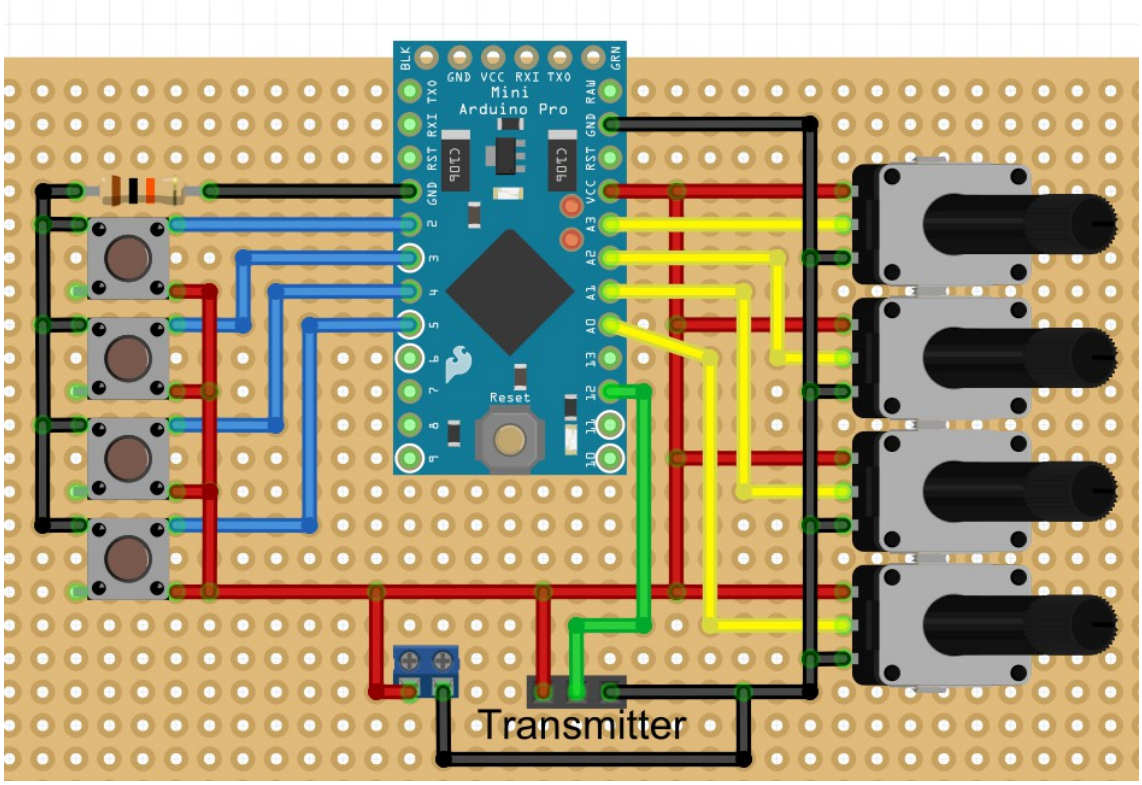

Bild 8.1

Widerstand verändert, wodurch die Spannung am mittleren Pin zwischen 0V und 5V liegt. Mit den Analogen Pins am Arduino kann man diese Spannung als einen Wert zwischen 0 und 1023 auslesen.
Der Transmitter muss nur an 5V und Ground angeschlossen werden, und dabei der mittlere „Data“ Pin mit den Arduino Pin 12 verbunden werden.
Für den Strom sorgt hierbei ein oder mehrere Akkus mit einen Spannungs-„Booster“, mit dem eine solide 5V Stromversorgung gewährleistet wird.

3.2 Programmierung

Nun muss die Hardware mit einer Software verbunden werden. Diesen Teil kann man in den Transmitter und den Receiver unterteilen.

3.2.1 Transmitter

Die Fernsteuerung zu programmieren ist der einfachste Teil. Hierzu muss die Arduino-Library VirtualWire importiert werden. Um Werte zwischenspeichern zu können, müssen Variablen erstellt werden, in diese die Werte eingespeichert werden können. Dafür ist der Datentyp „Integer“ (kurz „int“) am besten geeignet, da in diesen die eingelesenen Werte zwischen 0 und 1023 der Potentiometer auf jeden Fall reinpassen und die Werte der Schalter 0 und 1 genau so. Außerdem muss ein „int8_t“ Array erzeugt werden, in das die Werte geschrieben wird, die am Ende über das Transmitter-Modul mit Hilfe von VirtualWire gesendet werden soll. Dieser Datentyp muss verwendet werden, damit die Werte richtig übertragen werden können. „int8_t“ entspricht einem Datentyp mit einer Länge von 8bit.

In der Setup-Methode kann VirtualWire der Pin für den Transmitter übergeben werden (hier wird der Standardpin 12 verwendet – diese Methode muss also nicht ausgeführt werden), es kann ein „Push to Talk“ Pin festgelegt werden (wodurch es ermöglicht wird, dass nur solange an diesem festgelegten Pin ein „HIGH“-Signal ankommt auch etwas mit dem Transmitter-Modul gesendet wird) und wie „Push to Talk“ aktiviert werden soll (solange kein Signal oder ein Signal am entsprechenden Pin anliegt), wobei diese „Push to Talk“-Einstellungen in diesem Fall auf den Standards gelassen wird, da es nicht benötigt wird. Zuletzt muss VirtualWire die Anzahl an Bits pro Sekunde mitgeteilt

werden, wobei durch diese Methode („vw_setup(4000)“) auch die Library initialisiert wird (alle Einstellungen müssen vorher vorgenommen worden sein). In der darauf folgenden Loop-Methode, die im Gegensatz zur einmalig beim Starten des Mikrocontrollers aufgerufenen Setup-Methode solange wiederholt wird, bis der Mikrocontroller ausgeschaltet wird, wird nun alles ausgeführt, was für die Fernsteuerung essentiell ist.

Zuerst werden die Werte von den Potentiometer ausgelesen und in ihre Variablen gespeichert. Das selbe wird mit den Schaltern gemacht. Bevor die Werte nun in das Array gespeichert werden, bot es sich an, die Werte der Potentiometer aus dem Bereich von 0 bis 1023 auf einen Bereich von 0 bis 180 umzuwandeln. Dazu bietet sich die Methode „map(Wert, Min vom Wert, Max vom Wert, Min vom neuen Wert, Max vom neuen Wert)“ an. In diesem Zuge werden die „gemappten“ Werte in das Array gespeichert. Die Werte der Schalter werden direkt in das Array gespeichert. Dabei ist es natürlich wichtig, dass die Reihenfolge der gespeicherten Werte am Transmitter und am Receiver identisch ist. Zuletzt wird mit der Methode „vw_send()“ von VirtualWire das Array versendet, mit „vw_wait_tx()“ gewartet bis alles gesendet wurde und eine kurze Pause im Programm mit „delay(...)“ gemacht, um den Receiver nicht mit zu vielen Daten zu überfluten.

```
#include <VirtualWire.h>

int pwm1, pwm2, pwm3, pwm4;
int io1, io2, io3, io4;

int8_t Data[8];

void setup() {
  vw_setup(4000);
}

void loop()
  digitalWrite(13,1);
  pwm1 = analogRead(A0);
  pwm2 = analogRead(A1);
  pwm3 = analogRead(A2);
  pwm4 = analogRead(A3);
  io1 = digitalRead(0);
  io2 = digitalRead(1);
  io3 = digitalRead(2);
  io4 = digitalRead(3);

  Data[0] = map(pwm1,0,1023,0,180);
  Data[1] = map(pwm2,0,1023,0,180);
  Data[2] = map(pwm3,0,1023,0,180);
  Data[3] = map(pwm3,0,1023,0,180);
  Data[4] = map(io1,0,1023,0,1);
  Data[5] = map(io2,0,1023,0,1);
  Data[6] = map(io3,0,1023,0,1);
  Data[7] = map(io4,0,1023,0,1);

  vw_send((uint8_t *)Data, 8);
  vw_wait_tx();
  digitalWrite(13,0);
  delay(100);
}
```

Bild 10.1

3.2.2 Receiver

Den Receiver zu programmieren ist komplizierter als die Fernsteuerung, was unter anderem auch daran liegt, dass der Receiver den Fahrtenregler „aktivieren“ muss und er damit umgehen können muss, dass der Transmitter zum Beispiel auch mal keine Daten schickt.

Der Receiver hat in seinem Programm erst mal VirtualWire importiert, um die Daten empfangen zu können, „ServoTimer2“ um die Servos / Fahrtenregler ansteuern zu können (Es kann nicht die normale Servo.h-Library verwendet werden, da diese den selben Timer1 wie VirtualWire des Arduinos verwendet) und die Arduino-Library EEPROM um Daten auf dem Arduino (wie auf einer Festplatte zu speichern). Nach dem Importieren müssen ServoTimer2 Objekte erzeugt werden.

In der Setup-Methode wird zuerst VirtualWire die Bitrate (Bits pro Sekunde) übermittelt („vw_setup(4000)“) und mit „vw_rx_start()“ wird der Receiver-Prozess aktiviert. Den zuvor erzeugten ServoTimer2 Objekten muss nun ein Pin zugewiesen werden (siehe „3.1.1 Zusammenbau des Transmitters“) und die PinModi der für das Licht vorgesehenen Pins muss auf Output gestellt werden. Danach wird ein mittlerer Wert (experimentell herausgefunden) an alle Servos (/ den Fahrtenregler) mit einem PWM-Signal geschrieben um falsche Werte zu verhindern / den Leerlauf-Wert an den Fahrtenregler zu schicken, damit dieser nicht spontan Gas gibt. Daraufhin werden (eine kleine Spielerei) die letzten Werte der Lichter – ob sie an sind oder nicht – aus dem EEPROM (die „Festplatte“ des Arduinos) ausgelesen und die Ausgänge entsprechend ein- oder ausgeschaltet. Mit einer Pause von ungefähr vier bis fünf Sekunden im Programm wird bewirkt, dass der Fahrtenregler aktiviert wird (da er erst mal für kurze Zeit ein bestimmtes Signal braucht, um überhaupt was zu machen).

In der darauf folgenden Loop-Methode werden zuerst ein paar Variablen erzeugt, um später die empfangen Werte darin zwischenzuspeichern. Danach wird ein Array („buf“) vom Datentyp „unit8_t“ (ein „unsigned“ 8Bit Datentyp) mit der Länge der Nachricht erstellt. Wenn nun VirtualWire eine Nachricht hat, werden durch die Methode „vw_get_message()“ die Werte der Nachrichten das Array „buf“ gespeichert und man kann auf die gesendeten Werte zugreifen. Hier werden die Werte nun wieder auf für das Auto entsprechende Wertebereiche gemappt und an die Servos geschrieben / die Lichter ein- oder ausgeschaltet. Danach wird noch abgeglichen ob der Wert, der für die Lichter

im EEPROM gespeichert ist von dem aktuellen abweicht, und wenn dies der Fall ist wird der neue Wert gespeichert (diese Abfrage ist sinnvoll, da der EEPROM nur eine Max Lebensdauer hat und diese durch unnötige Schreibvorgänge stark verkürzt wird).

Sollte VirtualWire keine Nachricht empfangen haben, werden alle Servos auf den Mittelwert gesetzt werden, da ansonsten die letzten Werte beibehalten würden, womit ein Crash des Autos vorprogrammiert wäre.

```
#include <VirtualWire.h>
#include <ServoTimer2.h>
#include <EEPROM.h>

ServoTimer2 pwm1, pwm2, pwm3, pwm4;

int io1 = A0;
int io2 = A1;
int io3 = A3;
int io4 = A4;


void setup(){
   vw_set_rx_pin(12);
   vw_setup(4000);
   vw_set_ptt_inverted(true);
   vw_rx_start();

   pwm1.attach(3);
   pwm2.attach(5);
   pwm3.attach(6);
   pwm4.attach(9);
   pinMode(A0, OUTPUT);
   pinMode(A1, OUTPUT);
   pinMode(A2, OUTPUT);
   pinMode(A3, OUTPUT);

   pwm1.write(map(107,0,180,1000,2000));
   pwm2.write(map(107,0,180,1000,2000));
   pwm3.write(map(107,0,180,1000,2000));
   pwm4.write(map(107,0,180,1000,2000));
   digitalWrite(A0, EEPROM.read(0));
   digitalWrite(A1, EEPROM.read(1));
   digitalWrite(A2, EEPROM.read(2));
   digitalWrite(A3, EEPROM.read(3));
   delay(4000);
}

```

```
void loop(){
   int PWM1, PWM2, PWM3, PWM4;
   int IO1, IO2, IO3, IO4;

   uint8_t buf[VW_MAX_MESSAGE_LEN];
   uint8_t buflen = VW_MAX_MESSAGE_LEN;
   if(vw_have_message() == HIGH)
   {
     vw_get_message(buf, &buflen);

     PWM1 = map(buf[0],0,180,1000,2000);
     PWM2 = map(buf[1],0,180,1000,2000);
     PWM3 = map(buf[1],0,180,1000,2000);
     PWM4 = map(buf[1],0,180,1000,2000);
     IO1 = buf[4];
     IO2 = buf[5];
     IO3 = buf[6];
     IO4 = buf[7];

     pwm1.write(PWM1);
     pwm2.write(PWM2);
     pwm3.write(PWM3);
     pwm4.write(PWM4);
     digitalWrite(A0, IO1);
     digitalWrite(A1, IO2);
     digitalWrite(A2, IO3);
     digitalWrite(A3, IO4);

     if(EEPROM.read(0)!=IO1){EEPROM.write(
     if(EEPROM.read(1)!=IO2){EEPROM.write(
     if(EEPROM.read(2)!=IO3){EEPROM.write(
     if(EEPROM.read(3)!=IO4){EEPROM.write(
     delay(100);
   }
   else{
     pwm1.write(map(107,0,180,1000,2000));
     pwm2.write(map(107,0,180,1000,2000));
     pwm3.write(map(107,0,180,1000,2000));
     pwm4.write(map(107,0,180,1000,2000));
   }
}
```

4. Fazit

Abschließend kann man das Projekt als erfolgreich bezeichnen. Die Fernsteuerung erfüllt ihren Zweck, sie funktioniert und ist meinen Wünschen entsprechend. Alle vier Kanäle für die Servos und die Lichter funktionieren wie geplant. Die Reproduzierbarkeit ist auch gegeben, da die Komponenten und das Platinen-Layout einfach gestaltet ist. Das Preislimit wurde bei diesem Projekt nicht überschritten, was aber daran liegt, dass die meisten Teile bereits vorhanden waren. Als Einziges mussten die Funkmodule für 7€ und ein Arduino Pro Mini für 5€ gekauft werden. Der Arduino Uno stammt von einem anderen Projekt, die Platinen genauso wie „Male“ und „Female“ Pins sind aus dem eigenen Bestand und die verwendeten JoySticks sind aus einer alten Fernsteuerung heraus gebaut worden. Insgesamt würde die Fernsteuerung aber beim Kauf aller Komponenten um die 50 bis 60€ kosten. Wobei der Preis durch ein anderes Arduino-Modell für den Empfänger verringert werden könnte. Ich musste auf einen Arduino Uno zurückgreifen, da die Library ServoTimer2 nicht auf dem Arduino Pro Mini zu funktionieren scheint. Dies liegt wahrscheinlich daran, dass der verwendete Timer2 der Library nicht von dem Pro Mini unterstützt wird. Ob es ein günstigeres Model mit Unterstützung für ServoTimer2 gibt, weiß ich zum aktuellen Zeitpunkt nicht.
Um eine parallele Nutzung mehrerer Fernsteuerung zu ermöglichen, muss jede Fernsteuerung und jeder Empfänger eine eindeutige ID haben. Die Fernsteuerung würde diese dann mitsenden und der Empfänger überprüfen, ob diese mit seiner eigenen übereinstimmt. Diese ID ließe sich entweder fest einprogrammieren oder Variabel mit sogenannten DIP-Schaltern einstellen.
Weitere Möglichkeiten für eine Erweiterungen des Autos wären zum Beispiel Abstandssensoren, um einen Zusammenstoß mit anderen Objekten zu verhindern oder ein GPS gestütztes autonomes Fahren, bei der man über die Drahtlose Kommunikation den gewünschten Punkt übermittelt und das Auto den GPS-Werten entsprechend zu dem gewünschten Ort fährt. Dabei sollten auch die Abstandssensoren zum Einsatz kommen.

5. Literaturverzeichnis

1. „Arduino - EEPROM“ - https://www.arduino.cc/en/Reference/EEPROM – 13.03.2016

2. „Arduino - EEPROMRead“ - https://www.arduino.cc/en/Tutorial/EEPROMRead – 13.03.2016

3. „Arduino - EEPROMWrite“ - https://www.arduino.cc/en/Tutorial/EEPROMWrite – 13.03.2016

4. „Servos – Homofaciens“ - http://www.homofaciens.de/technics-base-circuits-servos_ge_navion.htm – 22.03.2016

5. „VirtualWire Library“ - https://www.pjrc.com/teensy/td_libs_VirtualWire.html – 27.02.2016

6. Quellenangaben der Abbildungen

Bild 5.1: http://www.bbc.co.uk/staticarchive/566d1059d04772c2c60b8ed7779edba3afe6a97d.gif – 22.03.2016

Weitere Abbildungen sind selbst erstellt worden.

7. Erklärung der eigenständigen Arbeit

Hiermit erkläre ich, dass ich diese Facharbeit ohne fremde Hilfe angefertigt habe und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel verwendet habe.