Abschlussarbeit

zum Hauptseminar

Sensor-Aktor-Integration auf einer mobilen Arduino-Plattform

Sommersemester 2013

Bearbeiter: B.Eng. Harald Funk
Studiengang: Master Ingenieurinformatik

Betreuer: M.Sc. Thomas Dietrich
Fachgebiet SSE

Inhaltsverzeichnis

Abstract

Ein handelsübliches Modellfahrzeug Modell Street Breaker der Firma Carson wurde im Vorfeld des Projekts auf die elektronische Ansteuerung durch ein Arduino-System umgebaut. Das System bestand aus einem Arduino Uno und einer provisorisch gelöteten Platine zur Motorsteuerung.
Ziel des Projekts war es, weitere Sensoren und Aktuatoren sowohl hardware- als auch softwaretechnisch zu implementieren, sowie eine stabile Halterung für das, den Arduino steuernde, Android-Smartphone zu entwickeln. Des Weiteren sollten die provisorisch ausgeführten Befestigungen und Verdrahtungen verbessert beziehungsweise erneuert werden.
Im Bereich der Sensoren wurde eine Schaltung samt Platinenlayout zur Messung der aktuellen Akkukapazität auf Basis eines DS2745 „Low-Cost Battery Management“ Chips der Firma Maxim entwickelt und über eine I^2C Schnittstelle in der Software des Arduino verarbeitet.
Des Weiteren wurden zwei Ultraschallsensoren des Typs HC-SR04 zur Hinderniserkennung an der Vorder- und Rückseite des Modellfahrzeugs angebracht, um einen restriktiven Eingriff in die Steuerbefehle zu ermöglichen.
Im Bereich der Aktuatoren wurde eine Platine auf Basis des 350mA Treibers ILD4035 der Firma Infinion entwickelt und aufgebaut, welche zwei 1W High-Power LEDs der Firma Avago antreibt, um die Kamera des Smartphones auch bei schlechtem Umgebungslicht funktionsfähig zu halten.
Weiterhin wurden 2 Status-LEDs angebracht, welche den aktuellen Zustand des Modellautos in Bezug auf Stromversorgung und Verbindung zum Steuerrechner anzeigen, sowie eine weitere LED, welche vom Steuerrechner ein- und ausgeschaltet werden kann.
Im Rahmen dieser Anbauten war es außerdem nötig, die bisher provisorisch zusammengelötete Motorsteuerungsplatine komplett durch eine neu geroutete Steuerplatine zu ersetzen, welche neben der reinen Motorsteuerung auch die Anschlüsse für die Sensoren und Aktuatoren enthält.

Abbildung 1 Aussehen des Modellfahrzeuges vor Beginn (links) und nach Fertigstellung (rechts) des Projekts (noch ohne Smartphone-Halterung)

Vorhandener Aufbau und Motivation des Projekts

Vor Beginn dieser Arbeit wurde im Rahmen eines anderen Projektes das Modellfahrzeug bereits auf eine Ansteuerung der Motoren mittels Arduino umgebaut. Hierfür wurde eine Motortreiberplatine gelötet, welche mittels eines L298N Motortreibers die Steuerbefehle, welcher der Arduino über ein Android-Smartphone erhält, auf den Antriebsmotor weiterleitet.
Diese Motorplatine war auf die Kontaktleisten des Arduino gesteckt und mit Klebeband an das darunterliegende Arduino-Board montiert, da durch die schnellen Bewegungsänderungen des Modellfahrzeuges beim Fahren und dem einseitigen Schwerpunkt des L298N samt Kühler ein Herausrutschen der Kontaktstifte anders nicht zu verhindern war. Ein Foto des vorgefundenen Zustandes der Steuerplatine ist in Abbildung 2 zu sehen.

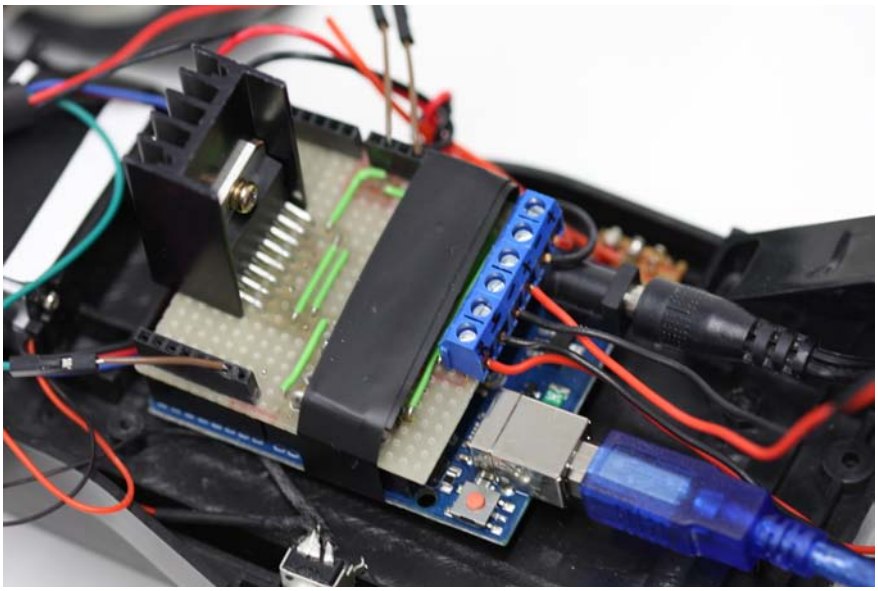


Abbildung 2 Aussehen der verbauten Steuerplatine vor Beginn des Projekts

Die gesamte Steuereinheit wurde mittels einer einzelnen Schraube auf dem Fahrgestell des Modellfahrzeuges befestigt und war hierdurch unsicher montiert und kurzschlussgefährdet.
Ebenso waren sämtliche Leitungen umständlich und über mehrere Adapter zwischen den einzelnen Platinen und Motoren verlegt.
Das den Arduino steuernde Android-Smartphone wurde an das Gehäuse des Modellfahrzeuges mittels eines Klettklebebandes befestigt und durch ein langes Adapterkabel mit dem Arduino verbunden. Diese Anbringung ist zwar funktional, jedoch nicht sonderlich schön und sicher.

Neben der Beseitigung dieser offensichtlichen Schwächen des vorhandenen Aufbaues sollen weitere Aktoren und Sensoren an das Modell angebracht, verdrahtet und softwaretechnisch ausgewertet werden. Hierdurch soll ein einfacherer und sichererer Betrieb des Fahrzeuges während des Einsatzes möglich werden.

Zur Überwachung des aktuellen Ladezustandes und einer Vorhersage der noch verbleibenden Restkapazität soll eine Möglichkeit geschaffen werden, die Restkapazität des Akkus jederzeit abzufragen und auf dem Steuerrechner angezeigt zu bekommen.

Dies soll ein rechtzeitiges Zurückkehren zur Ladestation ermöglichen, so dass unerwartete Ausfälle aufgrund zu geringer Akkuladung ausgeschlossen werden können.
Des Weiteren soll mittels Ultraschallsensoren die Umgebung des Fahrzeuges in Fahrtrichtung abgetastet werden, um einen restriktiven Eingriff in die Steuerbefehle zu ermöglichen.
Fährt das Modellfahrzeug mit hoher Geschwindigkeit in Richtung eines Hindernisses, soll der Ultraschallsensor dies erkennen und die vom Steuerrechner erhaltenen Geschwindigkeitsbefehle so weit reduzieren, dass durch einen Crash mit dem Hindernis keine schwerwiegenden Folgen für das Modellfahrzeug auftreten können.
Neben diesen sensorischen Auswertungen soll zusätzlich für eine bessere Sicht der Handykamera bei schlechten Lichtbedingungen gesorgt werden. Hierfür sollen High-Power LEDs an das Fahrgestell angebracht werden und vom Steuerrechner aus bedienbar sein, um diese bei schlechter werdenden Lichtverhältnissen bedarfsgerecht zuschalten zu können.
Außerdem soll der aktuelle Status des Fahrzeuges, also das Vorhandensein von Versorgungsspannung und die Verbindung mit dem Steuerrechner mittels LEDs angezeigt werden können und eine weitere LED bereitstehen, welche man vom Steuerrechner aus ein- und ausschalten kann.

Erweiterung der Hard- und Software des Modellfahrzeuges

Aufbau und Schaltung der Motorsteuerungsplatine

Vor Beginn dieses Projektes wurde die Ansteuerung des Motors über einen L298N Motortreiber-IC der Firma STMicroelectronics in Kombination mit vier Schottky-Dioden als Freilaufdioden durchgeführt, welche auf einem Stück Lochrasterplatine mittels kurzer Drähte zusammengelötet waren.

Der L298N verfügt über zwei unabhängige H-Brücken Treiber, deren Ausgänge mittels zweier Enable-Signale jeweils an- und ausgeschaltet, sowie mittels eines pulsweitenmodulierten Signals (PWM) jeweils einzeln in Ihrer Ausgangsspannung gesteuert werden können.
Der genaue Schaltplan war nicht mehr vorhanden und wurde daher im Rahmen des Projektes rekonstruiert. Die Schaltung ist in Abbildung 3 als Schaltplan zu sehen.

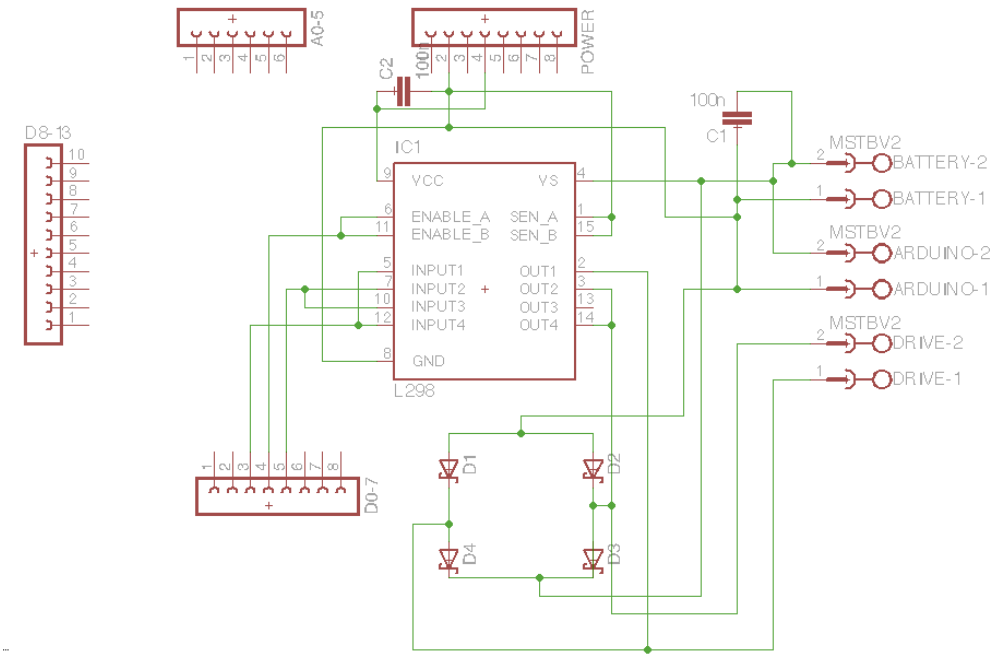


Abbildung 3 Schaltplan der vorhandenen Motorsteuerung

Zu erkennen ist, dass die beiden unabhängigen Treiber antiparallel verschalten wurden, um die Leistung des ICs zu erhöhen. Hierdurch wird es möglich bis zu 3 Ampere, anstatt der sonst möglichen 2 Ampere, zu schalten.
Aufgrund der benötigten Erweiterungen sowie dem allgemeinen Zustand der Platine wurde entschieden, die Motorsteuerungsplatine komplett neu aufzubauen und zusammen mit den für die Sensoren und Aktoren benötigten Steckverbindern als komplettes Shield[1] für den Arduino neu zu entwerfen.
Die Grundschaltung des Motorbrückentreibers wurde beibehalten, jedoch um die im Datenblatt geforderten Kondensatoren erweitert um mögliche hochfrequente Störungen abzublocken.

[1] Ein auf die Abmaße eines Arduino abgestimmtes und pinkompatibles Erweiterungsboard, welches die Funktionen des Arduino nach außen führt und erweitert.

Auf der Platine wurden außerdem Leitungen für die beiden Spannungs–versorgungspins Vin und GND, die digitalen I/O-Pins für die Ultraschallsensoren, die Status LEDs und die Steuerung der High-Power-LEDs sowie für zwei analoge Pins für den I²C-Bus verlegt und auf Steckverbinder geschaltet.
Das Layout der neuen Motorsteuerungsplatine ist in Abbildung 4 dargestellt. Der zugehörige Schaltplan ist in Anhang 1 zu finden. Der Schaltplan und das Layout der Motorsteuerungsplatine, sowie aller anderen in dieser Arbeit gezeigten Layouts und Schaltpläne, wurden mit der Software Eagle in einer freien, nichtkommerziellen Lizenz erstellt.

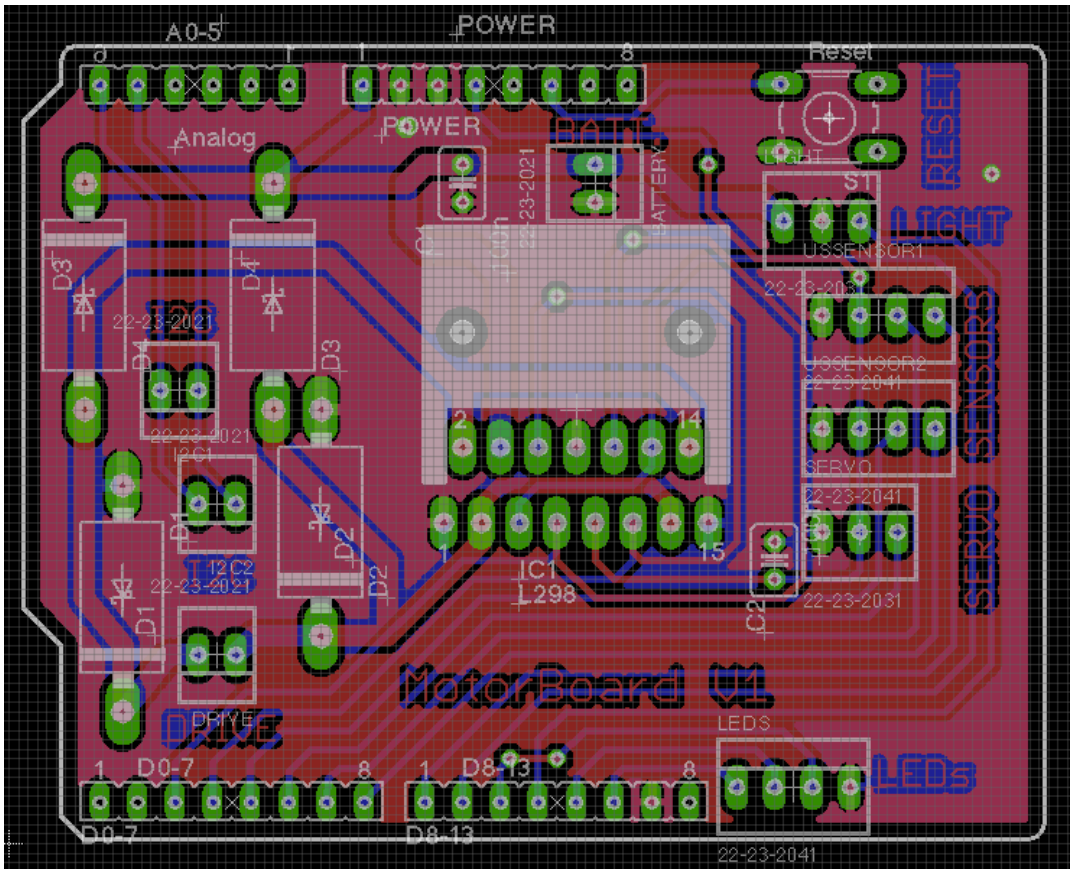


Abbildung 4 Layout der neugestalteten Motorsteuerungsplatine samt Steckverbinder

Beim Layout des Shields wurde darauf geachtet, den Schwerpunkt der Platine möglichst zentral zu halten, um Kontaktprobleme, welche durch die Fahrbewegungen des Modellfahrzeugs entstehen können, zu minimieren. Dies war bei der vorherigen Lösung ein großes Problem, weshalb die Platine mittels Klebeband am Modellfahrzeug fixiert werden musste. Der schwere Motortreiber samt Kühlkörper wurde daher möglichst mittig platziert.

Für die Spannungsversorgung der einzelnen Bauteile sind sowohl die 9,6V Nennspannung der Batterie, welche zwischen 11,2 Volt im geladenen und 7,2 Volt im ungeladenen Zustand liegt, sowie die vom Arduino ausgegebene und geregelte 5V Spannung auf dem Shield verdrahtet.
Die Batteriespannung wird an den Motortreiber, sowie an die High-Power-LED-Treiber angelegt, da diese eine hohe Leistung benötigen. Alle anderen Aktoren sind über die geregelte 5V-Spannung des Arduino versorgt.
Eine genaue Verschaltung der Pins des Arduino auf die einzelnen Bauteile des Motorboards sowie deren Funktion sind in Anhang 2 abgebildet.

Das entstandene Layout wurde mittels Laserdrucker auf eine durchsichtige Folie gedruckt und mittels UV-Licht auf eine mit Fotolack beschichtete Platine belichtet. Zur Entwicklung des Fotolacks wurde die Platine 15 Minuten in Natriumhydroxid gelegt und anschließend für ca. 30 Minuten in 60°C warmen Natriumsulfat geätzt. Anschließend wurde die Platine bestückt und auf Funktionsfähigkeit getestet. Ein Foto der fertig bestückten und angeschlossenen Platine ist in Abbildung 5 zu sehen.

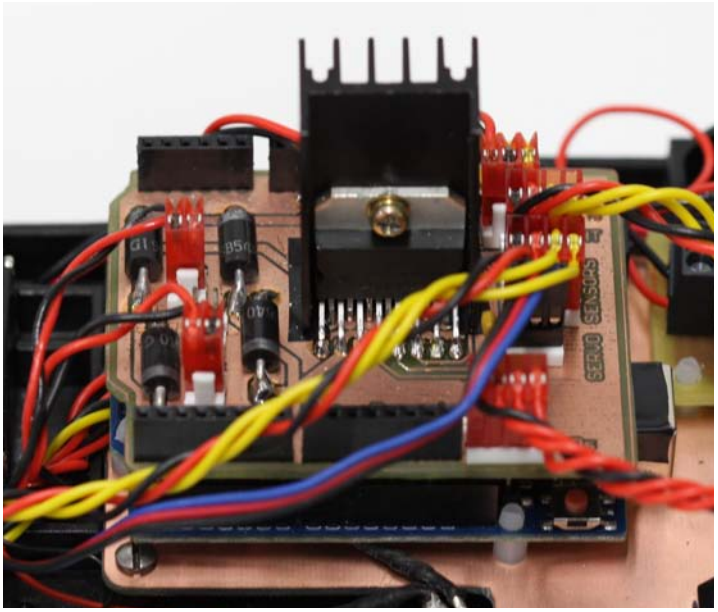


Abbildung 5 Foto des fertigen Motorboards

Messung der Akkukapazität

Zur Messung der Akkukapazität gibt es verschiedene Methoden, welche sowohl kurz aufgeführt als auch grundlegend erklärt und bewertet werden sollen. Daraufhin wird die Auswahl des benutzen ICs erklärt, dessen Verschaltung besprochen und anschließend auf die Kommunikation zwischen Arduino und dem IC eingegangen.

Vor- und Nachteile möglicher Messmethoden

Für die Messung der Kapazität eines Akkumulators (Akku) gibt es keine direkte Messmethode. Die einzige Möglichkeit besteht in der Messung sekundärer Größen und eine Schlussfolgerung von diesen auf die Kapazität des Akkus.
Die einfachste und gängigste, jedoch auch ungenauste Methode besteht im Messen der Zellenspannung eines Akkus.
Ein Akku verliert mit fallender Kapazität abhängig von den Zelleneigenschalten an Spannung. Durch eine einfache Messung der Leerlaufspannung des Akkus lässt sich somit auf den Ladungszustand ebendieses schließen. Jedoch sinkt die Leerlaufspannung nicht linear mit der Restkapazität, so dass die Messungen nachträglich linearisiert werden müssen. Zudem sind die Spannungsverluste bei den im Projekt verwendeten NiMh-Akkus über einen weiten Bereich der Entladung nur sehr klein, so dass eine sehr genaue Spannungsmessung von Nöten wäre. Eine schematische Darstellung der Entladekurve eines NiMh-Akkus ist in Abbildung 6 dargestellt.
Wie zu sehen ist, verläuft die Leerlaufspannung zwischen 50 min und 150 min fast waagerecht und ist daher nur sehr schwer und mit hoher Ungenauigkeit zu messen.

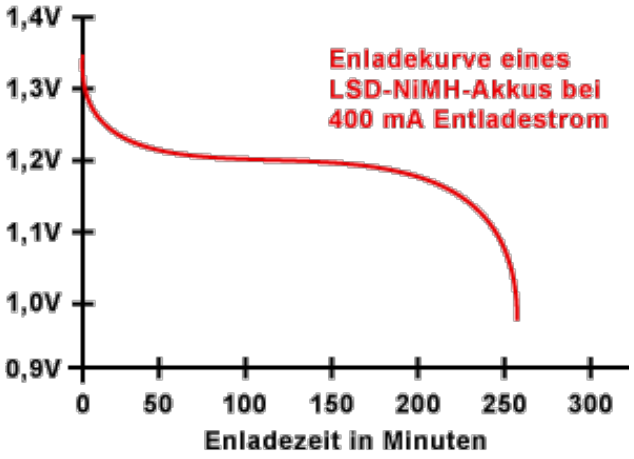


Abbildung 6 Entladekurve eines NiMh-Akkus (Quelle: elektronik-kompendium.de)

Eine weitere Methode der Kapazitätsmessung ist die Messung der in den Akku ein- und ausfliesenden Stromstärke. Hierbei wird beim Be- und Entladen des Akkus der Spannungsabfall über einen sehr kleinen Shunt-Widerstand mittels Spannungsmessung gemessen, daraus der über den Shunt fliesende Strom berechnet und über die Zeit integriert. Geht man davon aus, dass die in den Akku eingespeiste Ladungsmenge gleich der entladbaren Ladungsmenge ist, lässt sich somit eine Aussage über die Restkapazität des Akkus machen.
Messfehler ergeben sich über die im Widerstand in Hitze umgewandelte Leistung, weshalb man möglichst niederohmige Widerstände benutzt, sowie die Abtastung des Spannungssignals, wodurch kurzzeitige Stromspitzen eventuell nicht gemessen werden können. Durch die Niederohmigkeit des Widerstandes sind die gemessenen Spannungen ebenfalls sehr gering, so dass auch auf die Auflösung der Spannungsmessung Rücksicht genommen werden muss. Es bestehen daher hohe Anforderungen an die Genauigkeit des Shunt-Widerstandes und der Spannungsmessung.
Für die Messung des Spannungsabfalles und die Integration des gemessenen Stroms gibt es fertige ICs zu kaufen, welche in Kombination mit einem sehr niederohmigen Präzisionswiderstand eine sehr gute Kapazitätsmessung ermöglichen.

Auswahl des passenden ICs

Die Anforderungen an den zu verwendenden IC setzen sich wie folgt zusammen:

- Eigenständige Messung der am Widerstand abfallenden Spannung
- Eigenständige Integration der Messwerte
- Spezifiziert für NiMh-Akkus
- Einstellbarkeit der Maximalkapazität des Akkus
- Messbereich bis 3A Maximalstrom
- Einfache Kommunikation mit dem Arduino-Board
- Möglichst günstig

Sogenannte „Battery-Gauge“- oder „Battery-Monitor“-ICs gibt es von verschiedenen Herstellern und in verschiedenen Ausführungen.
Häufig sind diese jedoch nur für Lithium-Polymer- (Li-Po) Akkus spezifiziert, da sie spezielle Algorithmen zur genaueren Bestimmung der Restkapazität mit Einberechnung der physikalischen Eigenschaften der Li-Po-Akkus enthalten. Würden diese mit einem

NiMh-Akku eingesetzt werden, würde ein systematischer Fehler bei der Berechnung der Kapazität entstehen. Außerdem verwenden viele Hersteller zur Kommunikation mit den Chips ihre eigenen, herstellerspezifischen Protokolle, für die es keine fertigen Libraries für die Arduino-Boards gibt.
ICs mit UART- oder SPI-Schnittstellen sind meist sehr teuer und beinhalten zusätzliche Funktionen, wie zum Beispiel das Erkennen unterschiedlicher Akkus mittels zusätzlichem Chip, die für diesen Anwendungsfall jedoch nicht von Bedeutung sind.
Eine Ausnahme bildet der DS2745 „Low-Cost I2C Battery Monitor“ der Firma Maxim, welcher mittels I²C-Bus anzusteuern ist und alle geforderten Eigenschaften erfüllt.
Des Weiteren bietet er eine Messung der Temperatur und der aktuellen Versorgungsspannung des ICs an, welche neben der aktuellen Stromstärke und der Gesamtkapazität über das einfache Auslesen von Registern über eine Zweidrahtleitung zugänglich sind. Zusätzlich ist es möglich, für einzelne Werte Offsets zu definieren und die gesamte Messung zurückzusetzen.
Mit seiner hohen Eingangsimpedanz von 15MΩ und einer hohen Versorgungsspannungsvarianz von 2,5V bis 4,5V lässt sich der IC mittels einfachem Spannungsteiler, auch bei der stark schwankenden Versorgungsspannung der Batterie, sicher betreiben.
Der IC besitzt einen Spannungsmessbereich von -51,2mV bis +51,2mV und lässt sich somit durch die Wahl eines geeigneten Shunt-Widerstandes im Bereich um die 15mΩ auf einen Messbereich von ca. ±3,4A einstellen.

Aufbau und Schaltung der Messplatine

Für einen sicheren und stabilen Betrieb des DS2745 sind neben dem obligatorischen Präzisionswiderstand zur Erzeugung des zu messenden Spannungsabfalles auch einige weitere Bauteile nötig, welche die Versorgungsspannung des ICs auf den richtigen Wert bringen und Störungen abblocken.
Für die grundlegende Verschaltung des ICs wurde die, aus dem Datenblatt im Literaturverzeichnis ersichtliche, Beispielschaltung leicht verändert und um einen Spannungsteiler erweitert, welcher die Variable Versorgungsspannung von 11,2V bis 7,2V in einen Bereich von ca. 4,5 bis 2,5V herunterregelt. Der genaue Aufbau der Schaltung ist in Abbildung 7 zu sehen.

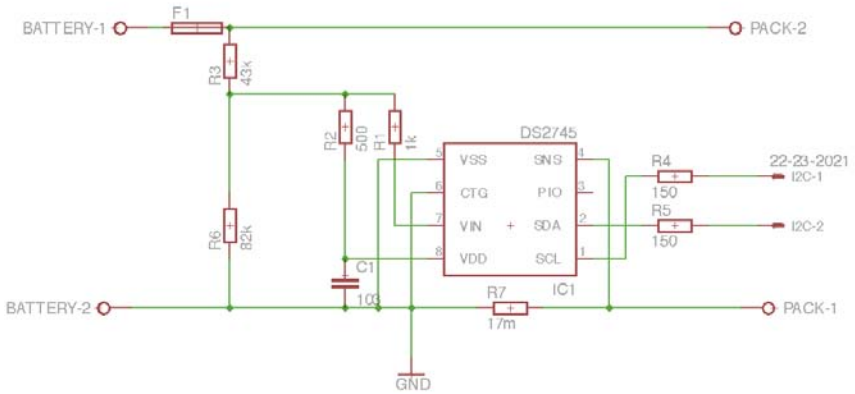


Abbildung 7 Schaltung des Batterie-Managements

Wie dem Schaltplan zu entnehmen ist, wird die Schaltung über die beiden Printklemmen BATTERY mit Spannung versorgt. Das Arduino-Board samt Motorboard wird über die Klemmen PACK angeschlossen, so dass der gesamte Strom im Rückleiter über den

Präzisionswiderstand R7 geleitet wird. Über diesen wird mittels der beiden Eingangssignale SNS und VSS des DS2745 der Spannungsabfall und dessen Richtung gemessen.
Der Signaleingang SNS kann eine Spannung von ±51,2 mV messen. Um einen maximalen Strom von 3A messen zu können, welcher dem Maximalstrom des Modellfahrzeug-motors entspricht, ist nach dem Ohm'schen Gesetz ein Widerstand mit einem Wert von 51,2mV/3A = 17mΩ nötig. Da es diesen Wert nicht zu kaufen gibt, wurde ein 15mΩ Widerstand eingesetzt und somit der messbare Strom auf ca. 3,4A erhöht.
Der Kondensator C1 dient zur Abblockung hochfrequenter Störungen der Versorgungsspannung und die beiden Widerstände R4 und R5 zur Strombegrenzung der I²C-Schnittstelle.
Zusätzlich wurde in die Versorgungsleitung eine Feinsicherung integriert, um die nachfolgende Schaltung samt Mikrocontroller, welche ihre Spannung über die Printklemme PACK erhält, vor Überstrom zu schützen.
Die beiden Pins der I²C-Schnittstelle wurden auf einen Steckverbinder geschalten, so dass eine einfache Verbindung zum Motorboard möglich wird.
Aufgrund von Platzgründen und der einfacheren Austauschbarkeit bei Defekten wurde die gesamte Schaltung auf einer eigenen Platine aufgebaut, deren Layout in Abbildung 8 dargestellt ist.

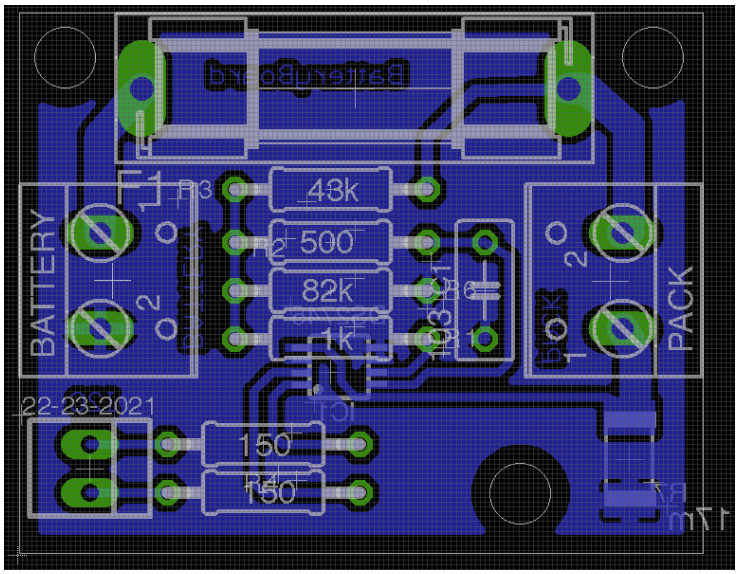


Abbildung 8 Layout der Battery-Management-Platine

Auslesen der Messwerte mittels I²C-Bus

Wie im Kapitel „Auswahl des passenden ICs" beschrieben, misst der DS2745 nur die am Messwiderstand abfallende Spannung und integriert diese über die Zeit auf. Neben diesen aktuellen und integrierten Werten werden zusätzlich noch die Temperatur und die Versorgungsspannung des ICs gemessen. Alle diese vier Messwerte schreibt der IC in eigene Register (2x8 Bit), welche über den I²C-Bus ausgelesen werden können.
Zusätzlich besitz der DS2745 noch weitere Steuerregister, über welche allgemeine Einstellungen sowie Offsets für die beiden Spannungswerte eingestellt werden können und der akkumulierte Spannungswert mit einem Anfangswert initialisiert und notfalls zurückgesetzt werden kann.
Eine Übersicht über die Register lässt sich im Datenblatt (siehe Literaturverzeichnis) des ICs nachlesen.

Beispielhaft soll hier das Auslesen, Beschreiben und Umrechnen des Accumulated Current Registers (ACR) gezeigt werden.
Die beiden Speicherregister des ACR besitzen die Adressen 0x10 und 0x11 und können voll, also mit jeweils 8 Bit, belegt werden. Es bestehen somit 2^{16} = 65536 diskrete Werte, in welche die akkumulierten Spannungswerte eingeteilt werden können.
Der maximale Speicherwert des Registers bei 0xFFFF liegt laut Datenblatt bei 409,6mVh, so dass ein Bit einem Wert von 409,6mVh/65536 = 6,25µVh entspricht. Bei einem Messwiderstand von 15mΩ entspricht ein Bit einer Kapazität von ca. 417µAh und der gesamte Messbereich einer Kapazität von 27,3Ah.
Geht man davon aus, dass der Akku im geladen Zustand in das Fahrzeug eingebaut wird, so muss man die aktuelle Ladung des Akkus im Chip hinterlegen, um die Restkapazität korrekt angezeigt zu bekommen. Aus diesem Grund ist das ACR das einzige Messregister des DS2745, welches auch Schreibzugriffe zulässt. Um die 1300mAh des in diesem Modellfahrzeug verwendeten Carson Racing Pack in den Chip zu schreiben, muss dieser Wert erst in den entsprechenden Spannungswert über dem Widerstand umgerechnet werden und anschließend als Binärzahl in das Register mittels I²C geschrieben werden.
Bei 1,3Ah und 15mΩ erhält man einen korrespondierenden Spannungswert von 1,3Ah*15mΩ = 19,5mVh, was einem Wert von 19,5mVh/6,25µVh/Bit = $(3120)_{10}$ = $(C30)_{16}$ = $(110000110000)_2$ entspricht. Dieser muss nun in 2x8 Bit gesplittet werden und kann daraufhin in die beiden Register 0x10 und 0x11 des DS2745 geschrieben werden.

Für die Kommunikation über I²C stellt das Arduino-Framework die Klasse „Wire“ zur Verfügung. Diese Klasse kapselt die Eigenschaften des Protokolls und enthält hierfür spezielle Operationen. Eine beispielhafte Funktion zum Setzen eines Registers ist in Abbildung 9 gezeigt.
Für den Start der Kommunikation muss in der setup-Routine des Arduino-Programms einmalig „Wire.begin()“ aufgerufen werden. Diese Funktion setzt interne Variablen und belegt die beiden Analogen Pins A4(SDA) und A5(SCL) des Arduino zur Kommunikation.
Um den soeben berechneten Wert in das ACR zu schreiben, wird zuerst mittels „Wire.beginTransmission(72)“ eine Verbindung mit dem Gerät der Identifikationsnummer 72 (Standardeinstellung DS2745) aufgebaut und dann mittels „Wire.write(Register)“ das passende Register adressiert. Daraufhin kann das soeben adressierte Register ebenfalls mittels „Wire.write(Daten)“ mit Daten beschrieben werden. Durch den Aufruf von „Wire.endTransmission()“ wird, anders als es der Name vermuten lässt, nicht die Verbindung beendet, sondern nur die in einen Buffer gelegten Werte an den DS2745 geschickt.

```
void setRegister(int registerbyte, int sendbyte){
  Wire.beginTransmission(72);    // begin transmission with slave device #72
  Wire.write(registerbyte); //Set register pointer to registerbyte
  Wire.write(sendbyte); // Write value of sendbyte to register
  int error = Wire.endTransmission(); // end trandmission
  if(DEBUG>0){
    Serial.print("Write To Register ");
    Serial.print(registerbyte,HEX);
    Serial.print(" Value: ");
    Serial.println(sendbyte,HEX);
    if (error!=0){
      Serial.print("I2C Error: ");
      Serial.println(error);
    }
  }
}
```

Abbildung 9 Funktion zum Beschreiben eines Registers

Von dem nun in das Register geschriebenen Kapazitätswert wird vom IC nun automatisch, je nach Stromrichtung, ein dem fließenden Strom entsprechender Betrag subtrahiert oder addiert.
Zum Auslesen des ACR kann nun ähnlich dem Schreiben auf das Register vorgegangen werden. Wieder wird eine Verbindung mittels „Wire.beginTransmission(72)“ zum DS2745 aufgebaut und anschließend mittels „Wire.write(Register)“ das auszulesende Register adressiert. Mittels „Wire.requestFrom(72,n)“ kann nun eine beliebige Anzahl n an Bytes vom Battery-Management angefordert und mittels „Wire.read()“ nacheinander ausgelesen werden. Hierbei wird nach jedem „Wire.read()“-Befehl das adressierte Register inkrementiert, so dass der Inhalt der n nachfolgenden Register nacheinander ausgegeben wird.
Die vom DS2745 erhaltenen Werte für die beiden Register 0x10 und 0x11 müssen nun gemäß Datenblatt miteinander zu einer Binärzahl mittels Bitshifting verknüpft werden und anschließend analog zum Beschreiben wieder in Kapazitätswerte umgerechnet werden.
Dieses Vorgehen ist nicht nur beim ACR zu benutzen, sondern verhält sich zum Auslesen der anderen Messwerte analog. Hierbei sind nur die jeweils passenden Registeradressen sowie die Umrechnungswerte zu beachten.

Verwendung und Auswertung des Ultraschallsensors HC-SR04

Für die Messung von Abständen mittels Ultraschall gibt es eine Unzahl von verschiedenen Sensoren unterschiedlicher Hersteller. Wichtig bei der Auswahl des Sensors war eine Messreichweite von mindestens 3 Metern, ein möglichst niedriger Preis und ein einfaches Kommunikationsinterface. Das für dieses Projekt ausgewählte Modell HC-SR04 gibt es bei deutschen Händlern für unter 10€ zu kaufen und ist somit das mit Abstand günstigste Modell auf dem Markt. Es besitz eine maximale Reichweite von 4 Metern und einen Öffnungswinkel von ca. 15°, was eine recht gute Richtungsauflösung ermöglicht.
Ein Foto des Sensors, aus welchem die Bauweise und die Pinbelegung des Sensors hervorgehen, ist in Abbildung 10 gezeigt.

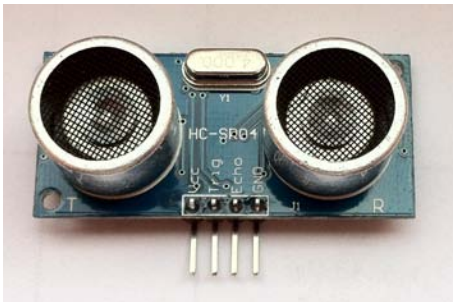


Abbildung 10 Fotographie des HC-SR04 Ultraschall-Sensors

Im Gegensatz zu teilweise deutlich teureren Sensoren fehlen dem HC-SR04 ein I²C oder UART Interface, welches für diesen Einsatzzweck jedoch nicht zwingend erforderlich ist. Anstelle von einem dieser beiden Interfaces besitzt der hier eingesetzte Sensor neben den beiden Spannungsversorgungs-Pins einen Trigger- und einen Echo-Pin. Diese beiden Pins werden für das Einleiten einer Messung (Trigger) und das Empfangen des zurückgestreuten Signals (Echo) benutzt.
Von diesem Sensor wurde je einer am vorderen Ende des Modellfahrzeuges in Fahrtrichtung und am hinteren Ende entgegen der Fahrtrichtung platziert, um sowohl

bei einer normalen Vorwärtsfahrt als auch beim Rückwärtsfahren eine Überwachung des Abstandes zu erreichen.
Zum Einleiten einer Messung muss der Trigger-Pin des Sensors auf einem Low-Signal liegen. Um einen stabilen Low-Pegel zu erlangen, wird zwei Mikrosekunden gewartet bevor der Trigger-Pin für 10µs auf einen High-Pegel geschalten und anschließend wieder auf Low gesetzt wird.
Hierdurch wird dem Sensor der Beginn einer Messung signalisiert und der Sensor beginnt mit dem Aussenden von Ultraschallwellen. Der Sensor setzt den Echo-Pin nun von Low auf High bis er das Echo des ausgesendeten Ultraschallsignals erhält. In diesem Moment setzt er den Echo-Pin wieder auf Low.
Über die von der Arduino-IDE mitgelieferte Funktion „pulseIn(Pin,Value,Timeout)“ wird auf ebendiesen High-Impuls gewartet und die Zeitdauer des Impulses gemessen. Diese Zeitdauer ist proportional zum Abstand des Hindernisses vom Sensor.
Der Abstand ergibt sich aus der halben Zeitdauer durch die Schallgeschwindigkeit in Luft (29,1 µs/cm). Der genaue Ablauf der Messfunktion ist in Abbildung 11 nachzulesen.

```
long measureDistance(int TriggerPin, int EchoPin)
{
    //Send Signal
    digitalWrite(TriggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(TriggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(TriggerPin, LOW);

    //wait for receiving the signal (max. waittime = Sensor-Timeout)
    long duration = pulseIn(EchoPin, HIGH,SensorTimeout);
    long distance = (duration/2) / 29.1; // Duration until Signal / 2 / Speed of Sound im microseconds/cm
    if (distance == 0 || distance > maxSensorRange){//Nothing in given max. Range
      distance = -1;
    }
    if(DEBUG>0){
      Serial.print("Distance: ");
      Serial.println(distance);
    }
    return distance;
}
```

Abbildung 11 Funktion zum Ansteuern des HC-SR04

Die zu übergebende Timeout-Variable wurde so dimensioniert, dass die maximal messbare Reichweite bei 3m liegt. Daraus folgt, dass die Funktion ein maximales Delay von 300cm*2*29,1µs/cm = 17,5ms erzeugt. Wird in dieser Zeit kein Echo gemessen, wird eine -1 ausgegeben.
Mittels der berechneten Abstandswerte kann nun in das vom Steuerrechner gesendete Geschwindigkeitssignal eingegriffen werden.
Hierzu wird der gemessene Abstandswert einem Abstandsbereich zugeordnet und der entsprechend definierte maximale Geschwindigkeitswert des Fahrzeuges mit sinkendem Abstand zum Hindernis auf einen kleineren Wert eingestellt. Somit soll erreicht werden, dass ein Zusammenstoß mit einem Hindernis keine zu großen Schäden am Modellfahrzeug anrichten kann.

Schaltung und Steuerung der High-Power-LEDs und der Status-LEDs

Zur Steuerung des Fahrzeuges auch außerhalb der Sichtweite des Fahrzeugführers wird an den Steuerrechner dauerhaft das Videosignal der Handkamera übertragen. Diese Kamera ist jedoch nur bei guten Lichtverhältnissen sinnvoll einzusetzen. Um eine Steuerung des Fahrzeuges auch bei schlechten Lichtverhältnissen oder sogar in der Dunkelheit zu ermöglichen, wurden an der Vorderseite des Fahrzeuges zwei 1W High-

Power-LEDs der Firma Avago angebaut. Diese beiden LEDs besitzen bei 3,5V Versorgungsspannung einen Nennstrom von 350mA und sind somit nicht direkt vom Arduino steuerbar.
Zum Ansteuern der LEDs wurde zwischen diesen und dem Arduino ein ILD 4035 LED-Treiber der Firma Infinion geschaltet. Dieser 350mA Treiber ermöglicht neben dem Bereitstellen einer Konstantstromquelle auch das Dimmen der angeschlossenen LEDs. Hierfür kann ein PWM-Signal an den Enable-Eingang des Chips angelegt werden und über dieses der Strom durch die LEDs proportional gesteuert werden.
Da der Chip ebenso wie der DS2745 zur Strommessung einen Messwiederstand (hier 330mΩ) sowie einige weitere Bauteile zur Stabilisierung benötigt, wurde auch für diesen IC eine kleine Platine entwickelt. Die Verschaltung der Platine ist in Abbildung 12 gezeigt.

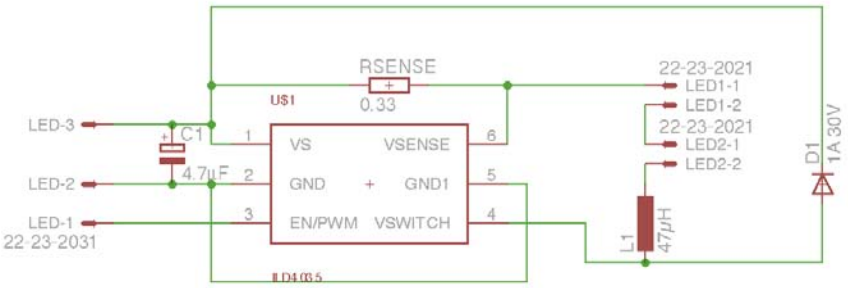


Abbildung 12 Schaltplan der LED-Treiber-Platine

Um einen konstanten Strom von 350mA unabhängig von der angehängten Last zur Verfügung zu stellen, misst der ILD 4035, nach gleichem Prinzip wie der schon beschriebene DS2745, den aktuell fließenden Strom anhand des Spannungsabfalls zwischen VS und VSense über einem 330mΩ Widerstand und regelt seinen Ausgang somit selbst auf das jeweils benötigte Spannungspotential an VSwitch.
Zur Abblockung von hochfrequenten Störungen wurde ein Kondensator zwischen Versorgungsspannung und Ground geschaltet. Dieser bildet mit der Ohm'schen Last im IC einen Tiefpass.
Um den Einschalt- und Ausschaltstrom der LEDs zu Verringern, wurde in Reihe eine Induktivität mit 47µH gesetzt. Dies bewirkt einen langsameren Anstieg der Stromstärke durch die LEDs und somit eine geringere Belastung des Treiberbausteines.
Zum Schutz vor Spannungsspitzen in Sperrrichtung der LEDs wurde zusätzlich eine Schottky-Diode als Freilaufdiode parallel zur Induktivität und den LEDs verbaut. Der Metall-Halbleiter-Übergang der Schottky-Diode schaltet schneller durch als der Halbleiter-Halbleiter-Übergang der LEDs und schließt eine möglicherweise schädliche Spannung somit rechtzeitig kurz.
Das Layout der fertigen Platine ist in Abbildung 13 dargestellt.

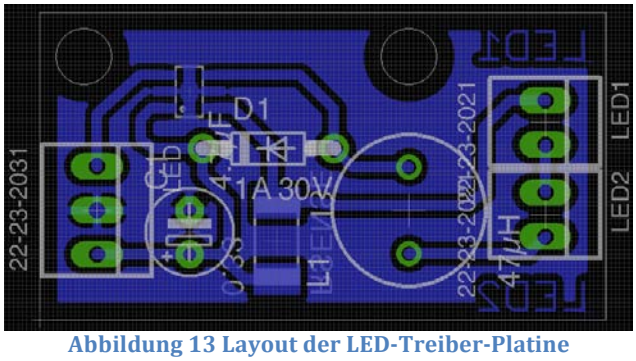


Abbildung 13 Layout der LED-Treiber-Platine

Eine Fotografie der fertig gebauten und auf dem Modellfahrzeug befestigten Platine samt der zu treibenden High-Power-LEDs ist in Abbildung 14 gezeigt.

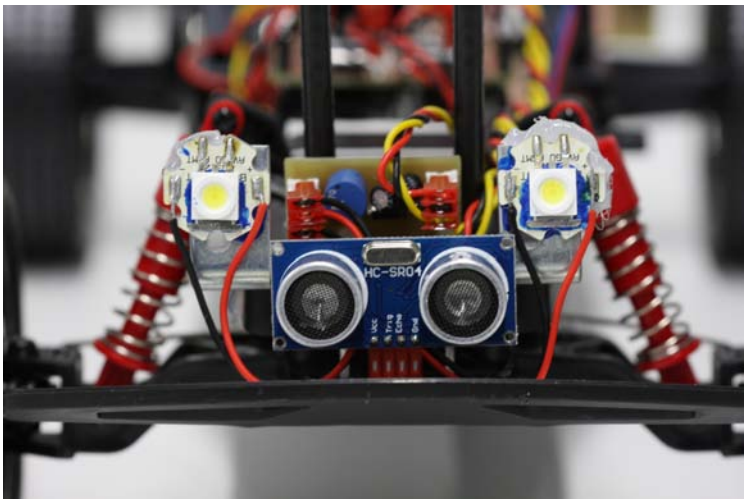


Abbildung 14 LED-Treiber-Platine (Hintergrund) samt High-Power-LEDs und Ultraschallsensor

Zum Ablesen des aktuellen Zustandes des Arduino in Bezug auf Spannungsversorgung und der Verbindung zum Smartphone wurde an die linke Seite des Modellfahrzeuges eine kleine Platine mit 3 LEDs in den Farben Rot, Grün und Gelb angebracht. Letztere kann über die Software am Steuerrechner beliebig ein- und ausgeschaltet werden, um eine bedarfsgerechte Anzeige beliebiger Ereignisse zu ermöglichen.
Die rote LED zeigt den Verbindungsstatus mit dem Smartphone an. Bei jedem Steuerungsbefehl leuchtet die LED kurz auf und erlischt nach Ausführung des Befehls wieder.
Für die Anzeige der korrekten Spannungsversorgung ist die grüne LED vorgesehen. Besitzt das Arduino die nötige 5V-Spannung und ist fertig gebootet, so leuchtet diese LED dauerhaft.
Ein Bild der am Rahmen des Modellfahrzeuges befestigten Platine und deren Layout ist in Abbildung 15 zu sehen.

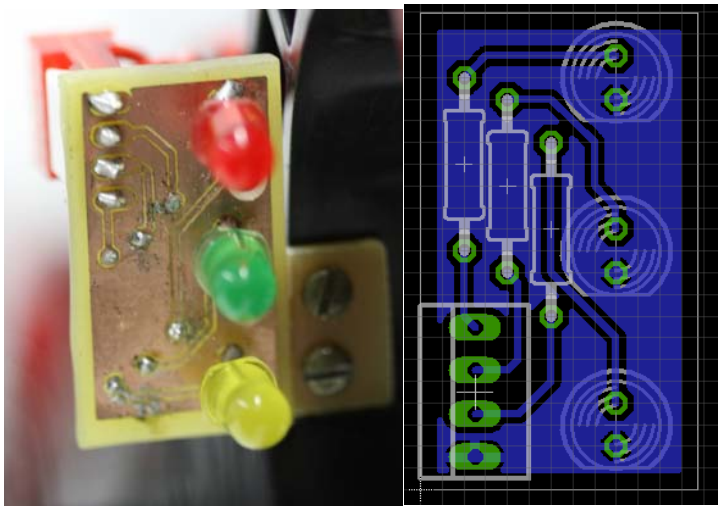


Abbildung 15 Foto und Layout der Status-LED-Platine

Konstruktion und Einbau der Handyhalterung

Für die Umsetzung der Steuerbefehle des Steuerrechners in Befehle für den Arduino, die drahtlose Kommunikation zwischen diesen beiden und der Aufnahme eines Live-Bildes ist ein Samsung Galaxy S2 Smartphone über USB an den Arduino angeschlossen. Dieses Smartphone wurde bisher mittels eines Klettbandes an die Rückseite des Modellfahrzeuges befestigt. Da diese Lösung auf Dauer keine sichere Halterung darstellt, sollte für das Smartphone eine stabile Halterung konzipiert werden, welche mittels eines 3D-Druckers hergestellt und auf das Grundgerüst des Modellfahrzeuges befestigt werden sollte.
Für den Entwurf eines druckbaren 3D-Modells wurde die Software Autodesk Inventor in einer Studentenlizenz benutzt. In diesem Programm wurde zuerst ein grobes Modell des Smartphones erstellt und anhand diesem eine Halterung entwickelt.
Die Halterung soll die Möglichkeit besitzen, einen Mikro-USB-Stecker zu befestigen, so dass man das Smartphone durch einfaches Einstecken in die Handyhalterung gleichzeitig mit dem Arduino verbinden kann. Hierzu wurde an die reine Halterung an der Unterseite ein Bogen angebracht, in welchen der Mikro-USB Stecker eingesteckt und justiert werden kann.
Das Smartphone selbst wird seitlich von zwei U-Profilen gehalten, in welche das Handy von oben eingeschoben werden kann.
Die Halterung soll zwischen den beiden hinteren „Backen“ des Modellfahrzeuggestells befestigt werden, so dass die Abdeckung des Modellfahrzeuges noch darüber gesetzt werden kann. Um das Einschieben in die Halterung zu ermöglichen muss eine Öffnung in die Abdeckung des Modellfahrzeuges geschnitten werden.
Ein 3D-Modell der Halterung ist in Abbildung 16 zu sehen. Eine genauere technische Zeichnung ist in Anhang 3 angehängt.
Zum Zeitpunkt der Fertigstellung dieser Arbeit war die Halterung für das Smartphone leider noch nicht fertig gedruckt und montiert, so dass eine abschließende Bewertung der Konstruktion noch nicht möglich ist.

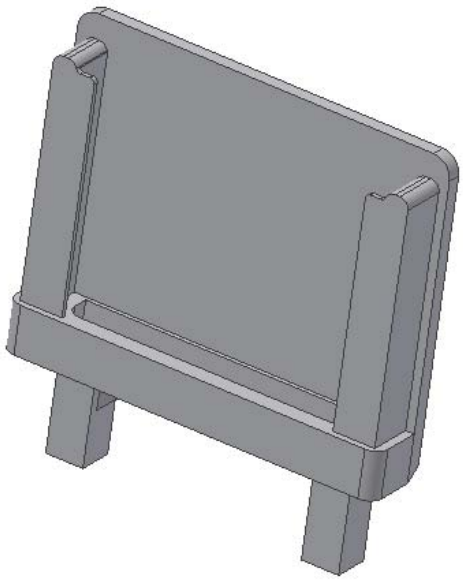


Abbildung 16 3D-Modell der entwickelten Halterung

Ansteuerung des entstandenen Gesamtsystems

Die grundlegende Ansteuerung des fertigen Gesamtsystems erfolgt wie in Abbildung 17 dargestellt.
Das auf dem Modellfahrzeug befestigte Smartphone kommuniziert auf der einen Seite mittels einer über USB emulierten seriellen Schnittstelle mit dem Arduino-Board und auf der anderen Seite mittels WLAN oder UMTS mit einem Steuerrechner, auf welchem eine spezielle Java-Applikation läuft.

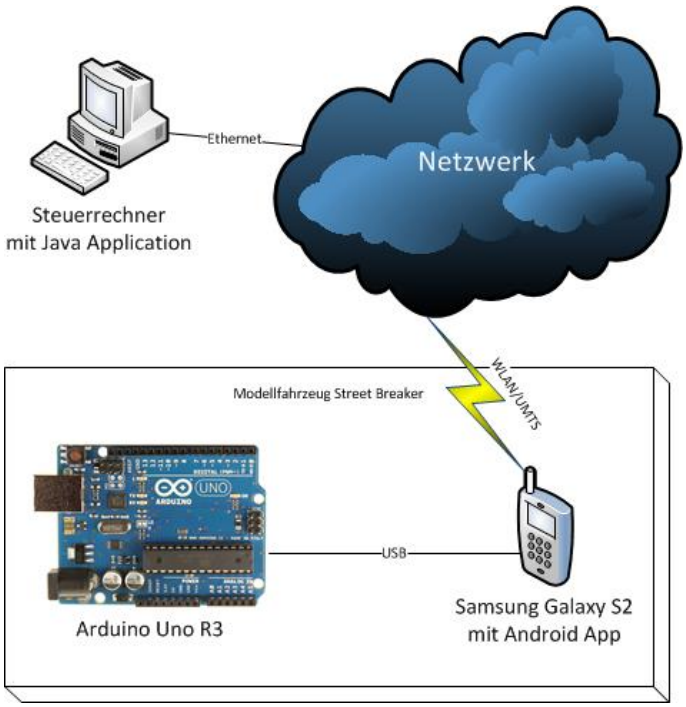


Abbildung 17 Kommunikationsverbindungen zwischen Steuerrechner und Arduino

Das Smartphone dient hierbei als Bridge zwischen Steuerrechner und Arduino. Die vom Steuerrechner empfangenen Steuerbefehle (IP-Pakete) werden vom Handy auf die für den Arduino verständliche serielle Kommunikation umgewandelt und die vom Arduino ausgesendeten Antworten über die Funkverbindung an den Steuerrechner zurückgesendet.
Die auf dem Android-Smartphone und dem Steuerrechner laufenden Anwendungen sind nicht Teil dieser Arbeit sondern wurden zuvor entwickelt und werden aktuell auf die in diesem Projekt entstandenen Veränderungen an der Beschaltung des Arduino angepasst.
Das für die Kommunikation zwischen Smartphone und Arduino nötige Protokoll zur Steuerung des Fahrzeuges sowie zur Auslesung der Sensordaten wurde jedoch aufgrund der vielfältigen Erweiterungen an der Hardware des Arduino in Rahmen des Projekts komplett neu entwickelt und in die Software des Arduino implementiert. Das komplette Protokoll ist in Anhang 4 nachzulesen.
Das bisherige Protokoll verwendete zur Ansteuerung des Arduino 4 Byte. Jeweils ein Byte für die Geschwindigkeit und der Beschleunigungsrichtung (vorwärts/rückwärts) sowie für die Steuerrichtung sowie den Steuerwinkel. Da die Steuer- und Beschleunigungswerte jeweils nur den Wert 1 oder 0 annehmen können ist eine Übertragung mittels einem Bit statt einem Byte ausreichend.
Auch die beiden anderen Steuerbefehle besitzen nur Werte zwischen 0 und 9 und können somit mit 4 Bit codiert werden.

Es werden daher für die Beschleunigung und die Geschwindigkeit sowie für die Steuerrichtung und den Steuerwinkel jeweils nur noch ein statt zwei Byte benötigt.
Des Weiteren wurde ein Byte mit Header-Informationen wie Protokolltyp und Länge einer Protokolleinheit in Byte eingefügt, um auch nur Teilinformationen versenden zu können und spätere Erweiterungen des Protokolls zu ermöglichen.
Außerdem wurde ein Byte zum Steuern der High-Power-LEDs, der Status-LED und dem softwareseitigem Zurücksetzen des Battery-Managements hinzugefügt. Diese Funktionen sind auf dem Arduino schon implementiert, jedoch aktuell von der Smartphone-App sowie der Desktop-Applikation noch nicht unterstützt, genauso wie die Anzeige der aktuellen Statusinformationen wie Akkurestkapazität, aktuellem Stromverbrauch, Versorgungsspannung und Temperatur.
Hierzu wird vom Arduino auf jeden Steuerbefehl mit sechs Byte geantwortet. Die Aufteilung der Bytes ist ebenfalls in Anhang 4 nachzulesen. Da eine Antwort des Arduino auf Steuerbefehle bisher in der Kommunikation mit dem Steuerrechner nicht vorgesehen war, ist auch diese Funktion noch nicht vollständig funktionsfähig, aber schon jetzt in der Firmware des Arduino implementiert.

Zukünftige Erweiterungsmöglichkeiten

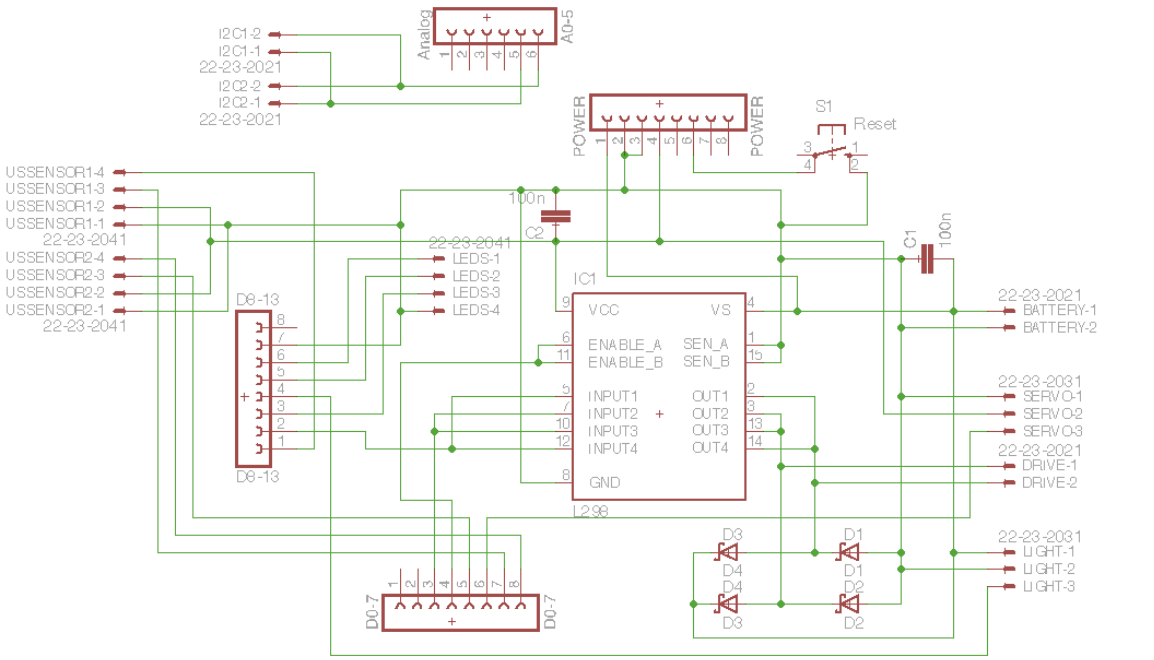
Durch die in diesem Projekt entstandenen Erweiterungen mit der beschriebenen Aktuatorik und Sensorik ist eine wesentlich einfachere und sicherere Bedienung des Modellfahrzeuges möglich geworden.
Jedoch ist ein Großteil der gewonnenen Informationen aktuell durch die fehlenden Schnittstellen zwischen Arduino und Android-Smartphone für den Endbenutzer nicht nutzbar. Hier wurde schon mit der Weiterentwicklung begonnen, welche in Zukunft eine einfache Handhabung mittels Desktop-Anwendung erlauben wird.
Für Hardware-Erweiterungen direkt am Mikrocontroller-Board wurde schon bei der Entwicklung des Motorboards ein weiterer Steckverbinder für eine I^2C Schnittstelle integriert, welcher parallel zur Schnittstelle für das Battery-Management geschaltet ist. Dieser zusätzliche Steckverbinder kann zum Beispiel für die Kommunikation mit einem GPS-Empfänger oder anderen I^2C kompatiblen Sensoren oder Aktoren benutzt werden.
Für die Auswertung von weiteren Sensoren und das Steuern weiterer Aktuatoren wurde im Übertragungsprotokoll ebenfalls die Möglichkeit einer Erweiterung berücksichtigt. Zum einen sind noch freie Bits in den bestehenden übertragenen Bytes für die spätere Erweiterung reserviert. Zum anderen können weitere Bytes am Ende des Protokolls angehängt werden. Hierzu besteht die Möglichkeit die Protokollversion im Header zu erhöhen und somit eine Unterscheidung zwischen neuem und altem Protokoll zu ermöglichen.
Der Erweiterbarkeit des Modellfahrzeuges sollte somit ohne größere Probleme und mit nur wenigen softwareseitigen Veränderungen möglich sein.

Literaturverzeichnis

- Schaltungstechnik; Johann Siegl; Springer Verlag; 3. Auflage
- Eagle Handbuch; Cadsoft Computer GmbH; Version 6 http://www.cadsoft.de/wp-content/uploads/2011/05/V6_manual_de.pdf
- Mikrocontroller.net; Leiterbahnbreite; Aufgerufen am 17.10.2013 http://www.mikrocontroller.net/articles/Leiterbahnbreite
- Akkukonfigurator.de; Firma Akkuline OHG; Aufgerufen am 30.09.2013; http://www.akkukonfigurator.de/NiCD-NiMH_Akku-Spannung.aspx
- Nickel-Metallhydrid-Akku; Elektronik-Kompendium; Aufgerufen am 30.09.2013; http://www.elektronik-kompendium.de/sites/bau/1101251.htm
- Datenblatt DS2745; Maxim Integrated; Version 2; Aufgerufen am 26.05.2013; http://datasheets.maximintegrated.com/en/ds/DS2745.pdf
- Arduino Homepage; arduino.cc; Aufgerufen am 19.07.2013;
- Arduino Playground; playground.arduino.cc; Aufgerufen am 19.07.2013
- Datenblattt HC - SR04; ElecFreaks.com; Aufgerufen am 04.10.2013; http://elecfreaks.com/store/download/HC-SR04.pdf
- Datenblatt ILD4035; Avago Technologies; Aufgerufen am 04.10.2013; http://www.avagotech.com/docs/AV02-1673EN
- Datenblatt L298N; STMicroelectronics ; Aufgerufen am 10.06.2013; http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000240.pdf

Anhang

Anhang 1 Schaltplan der neugestalteten Motorsteuerungsplatine

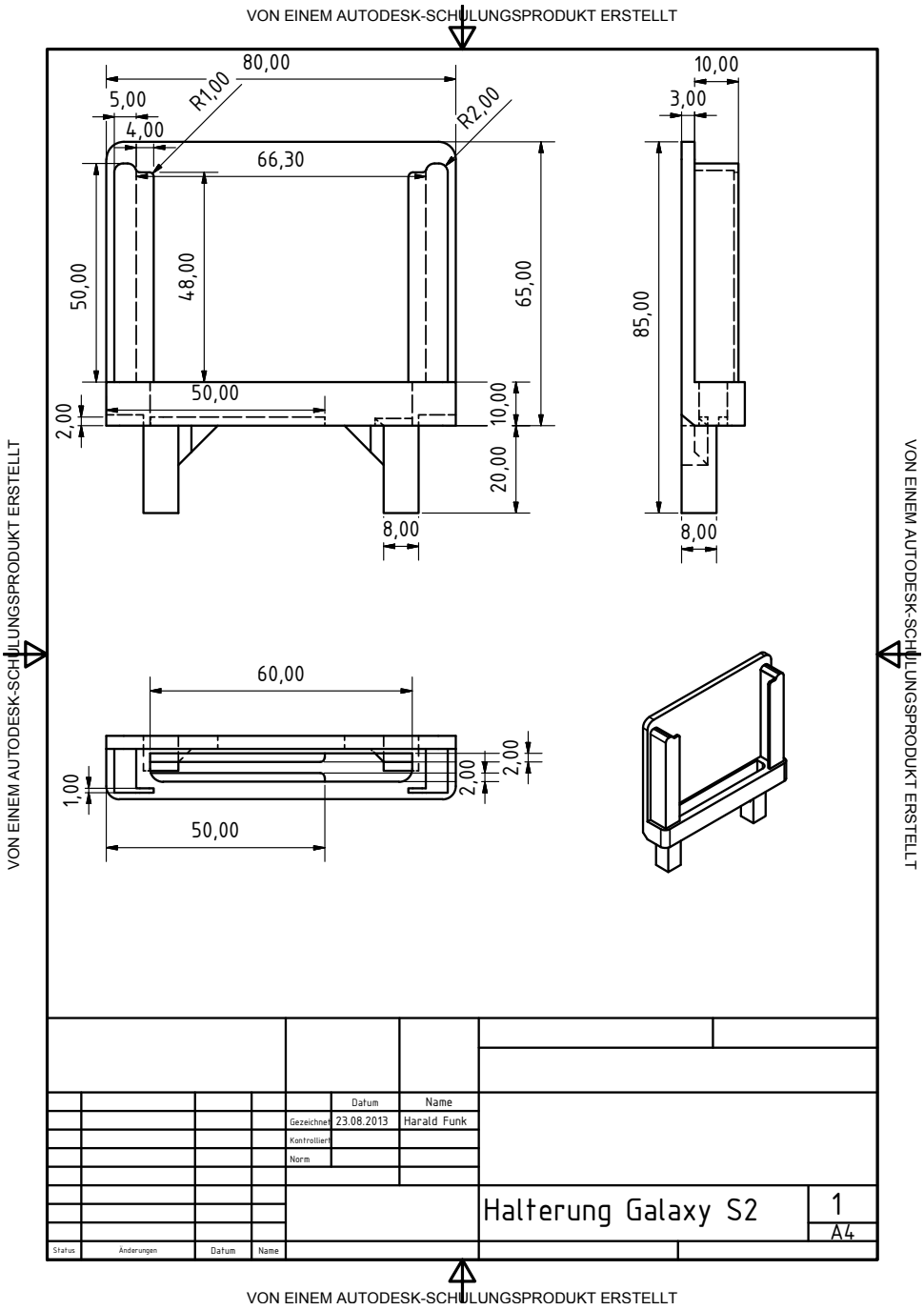


Anhang 2 Verschaltung Arduino mit Motorboard

Eagle PIN	Arduino PIN	Funktion	Verbunden mit
D0-7 1	D0	RX	-
D0-7 2	D1	TX	-
D0-7 3	D2	Richtungs-Eingang A2 und B2	IC1 Input2; IC1 Input3
D0-7 4	D3~	Enable Motor	IC1 EnableA; IC1 Enable B
D0-7 5	D4	Trigger US-Sensor1	USSENSOR1 3
D0-7 6	D5~	Servo Signal	SERVO 3
D0-7 7	D6~	Trigger US-Sensor2	USSENSOR2 3
D0-7 8	D7	Echo US-Sensor 1	USSENSOR1 4
D8-13 1	D8	Echo US-Sensor 2	USSENSOR2 4
D8-13 2	D9(~)	Richtungs-Eingang A1 und B1	IC1 Input1; IC1 Input4
D8-13 3	D10(~)	LED Rot	LED 1
D8-13 4	D11~	Enable Licht	LIGHT3
D8-13 5	D12	LED Grün	LED 2
D8-13 6	D13	LED Gelb	LED 3
D8-13 7	GND	GND	siehe Schaltplan
D8-13 8	AREF	-	-
Analog 1	A0	-	-
Analog 2	A1	-	-
Analog 3	A2	-	-
Analog 4	A3	-	-
Analog 5	A4 SDA	I2C SDA	I2C1 1;I2C2 1
Analog 6	A5 SCL	I2C SCL	I2C1 2;I2C2 2
Power 1	Vin	Versorgungsspannung 9,6V	LIGHT1; IC1 VS;

Power 2	GND	GND	siehe Schaltplan
Power 3	GND	GND	siehe Schaltplan
Power 4	5V	Versorgungsspannung 5V	USSENSOR1 2; USSENSOR2 2; IC1 VCC; SERVO 2
Power 5	3.3V	-	-
Power 6	RESET	Arduino Reset	RESET 4
Power 7	IOREF	-	-
Power 8	-	-	-

Anhang 3 Technische Zeichnung der Halterung für das Samsung Galaxy S2



Anhang 4 Protokoll zur Datenübermittlung

Android → Arduino

Byte	Bedeutung	Wertebelegung
1.7-1.6	Protocol Version	Protokoll-Version (aktuell 1)
1.3-1.0	Length	Anzahl der **folgenden** Bytes
2.7-2.4	Speed – Value	0 – 9: Geschwindigkeit in 10% von v_{max}
2.3	Speed – Direction	0: rückwärts 1: vorwärts
3.7-3.4	Steering – Steps	0 – 8: 90 + steps*90/8
3.3	Steering – Direction	1: rechts 0: links
4.7	Enable LED	0: aus 1: an
4.6	Enable Light	0: aus 1: an
4.5	Reset-Accumulated Current	0: - 1: einmaliger Reset

Byte 1: Protocol Header

MSB LSB

PV1	PV2			L1	L2	L3	L4
Protocol Version		Reserved		Length			

Byte 2: Speed

SpV1	SpV2	SpV3	SpV4	SpD			
Speed-Vale				Speed Direction	Reserved		

Byte 3: Steering

StS1	StS2	StS3	StS4	StD			
Steering Steps				Steering Direction	Reserved		

Byte 4: Lights and Battery

LED	L1			RAC			
LED Status	Light Status	Reserved	Reserved	Reset Acc Current	Reserved		

Arduino → Android: Jeweils als Antwort auf einen beliebigen Befehl vom Handy

Byte	Bedeutung	Wertebelegung
1.7-1.6	Protocol Version	Protokoll-Version (aktuell 1)
1.3-1.0	Length	Anzahl der **folgenden** Bytes
2	Momentary Current	Strom in mA/10
3	Absolute Residual Battery Capacity	Kapazität in mAh
4.7-4.1	Procental Residual Battery Capacity	Kapazität in %
5	Voltage	in 0.1 V
6	Sensor Temperature	in 0.5 °C

Byte 1: Protocol Header

MSB LSB

PV1	PV2			L1	L2	L3	L4
Protocol Version		Reserved		Length			

Byte 2: Momentary Current

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8
Momentary Current							

Byte 3: Absolute Residual Battery Capacity

ARC1	ARC2	ARC3	ARC4	ARC5	ARC6	ARC7	ARC8
Absolute Residual Battery Capacity							

Byte 4: Procental Residual Battery Capacity

PRC1	PRC2	PRC3	PRC4	PRC5	PRC6	PRC7	
Procental Residual Battery Capacity							Reserved

Byte 5: Voltage

V1	V2	V3	V4	V5	V6	V7	V8
Voltage							

Byte 6: Temperature

T1	T2	T3	T4	T5	T6	T7	T8
Temperature							