04.01.2015

Dokumentation

Arduino Car

Robin Köhnlein

CT - Halbjahresprojekt

2014 / 2015

Gliederung

I. Projektplanung

1.1 Zielsetzung

Mein Ziel für das Halbjahresprojekt im Fach CT ist es, ein ferngesteuertes Auto zu entwickeln und zu bauen. Ein Funduino Mikrocontroller soll alle Funktionen und Prozesse des Autos steuern. Als Signaleingabeeinheit möchte ich eine Smartphone App verwenden, welche über Bluetooth mit dem Funduino kommuniziert. Dadurch soll das Auto über ein Smartphone gesteuert werden können.

Um dies zu erreichen muss ich mich zuerst mit dem Funduino und der Arduino Software beschäftigen, um zu verstehen, wie man den Funduino programmiert. Mein Ziel ist es, den Funduino Mikrocontroller und die dazugehörigen Programmierbefehle soweit zu verstehen, dass sich Lösungswege für die geplanten Funktionen (siehe Seite 4 ff.) ergeben.

Meine Zielsetzung umfasst auch die Elektronik nicht nur theoretisch zu entwerfen, sondern sie auch praktisch umzusetzen und diese in das Auto zu installieren. Nur wenn die Elektronik, der Aufbau des Autos und das Steuerprogramm zusammen passen, kann das Auto am Ende funktionieren.

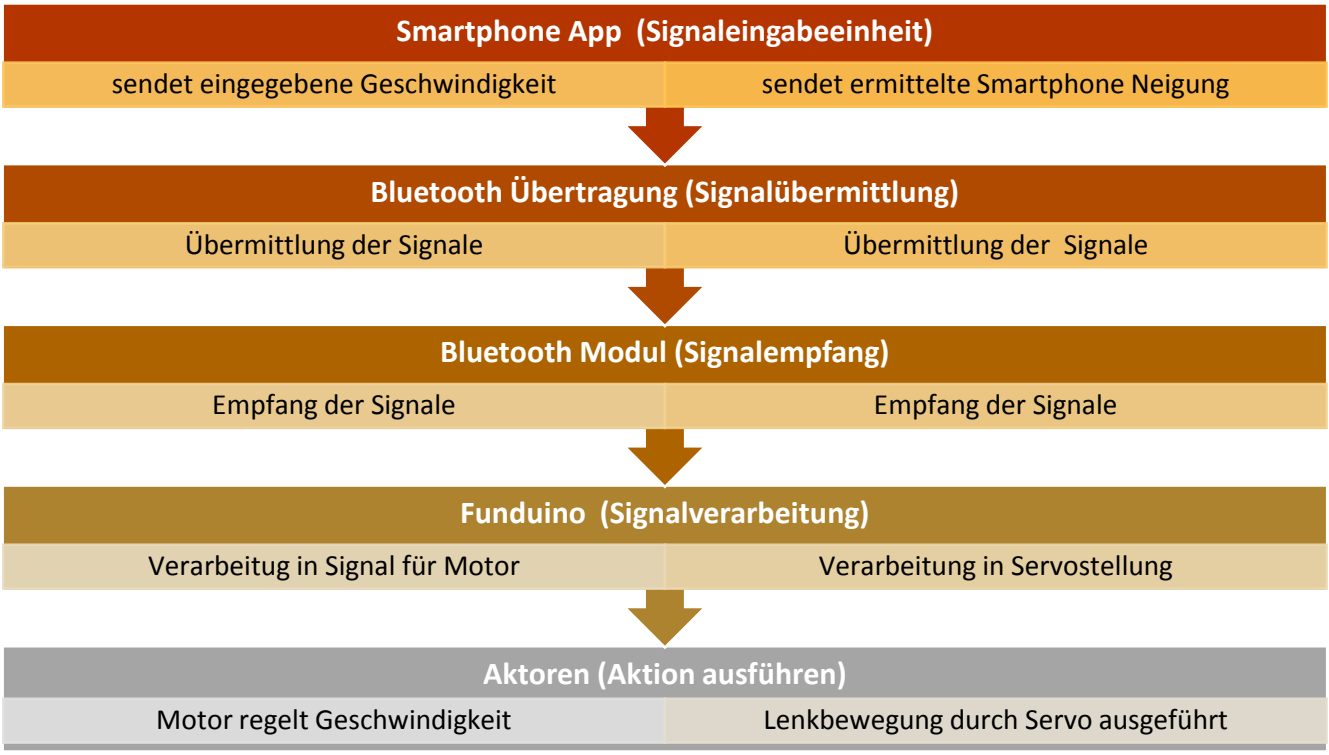
1.2 Geplante Funktionen

Bewegungssteuerung

Für die Bewegungssteuerung des Autos sollen die Hinterräder als Antrieb dienen. An den Hinterrädern befindet sich deswegen der Motor, der vom Funduino gesteuert wird. Die Geschwindigkeit des Autos bzw. des Motors soll einstellbar sein. Über die Smartphone App soll man verschiedene Geschwindigkeitsstufen des Arduino Car einstellen, sowie diese stufenlos senken oder erhöhen können.

Die Lenkung befindet sich an den Vorderrädern des Autos. Sie wird ebenfalls über die App gesteuert. Das Kippen bzw. Neigen des Smartphones soll die App erkennen und über Bluetooth dem Funduino übermitteln. Dieser erkennt die Neigungswerte, verarbeitet sie und gibt ein entsprechendes Signal an den Servo aus, welcher sich an der Vorderachse befindet. Dieser Servo ist mit der Lenkachse verbunden und stellt die gewünschte Richtung der Vorderräder ein.

Da die Geschwindigkeit und die Lenkung des Arduino Car durch die Smartphone App über Bluetooth gesteuert werden sollen, benötigt man ein Bluetooth Modul. Dieses muss als „Slave“ agieren und die Daten der App empfangen und an den Funduino übermitteln.



Signal LED

Eine LED in der Autokarosserie soll durch Blinken zeigen, dass die Bluetooth Verbindung zwischen Funduino und Smartphone aufgebaut ist.

Scheinwerfer

Das Auto besitzt an der Vorderseite einen Scheinwerfer. Dieser besteht aus mehreren LEDs und soll sich in seiner Leuchtstärke automatisch an die Umgebung anpassen. Wenn es dunkel ist, soll er hell leuchten und wenn es hell ist, schwach scheinen.

Hierfür ermittelt ein Fotowiderstand die Umgebungshelligkeit und der Funduino soll die entsprechende Leuchtintensität für den Scheinwerfer ermitteln.

Rücklicht

Am Arduino Car befinden sich an der Rückseite zwei Rücklichter. Diese sollen durch unterschiedlich schnelles Blinken gewisse Aktionen kennzeichnen, z.B. soll das rechte Licht bei einer gefahrenen Rechtskurve blinken.

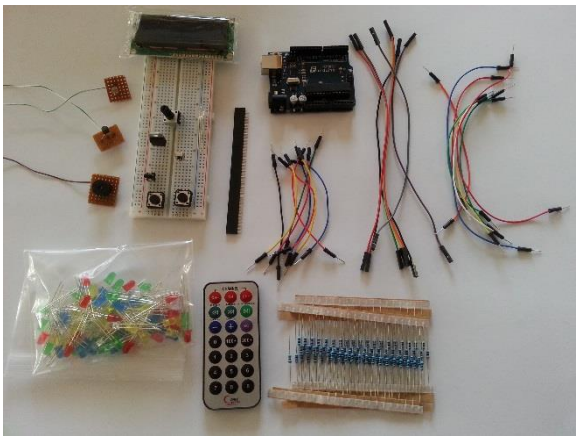
Kühlung

Im Arduino Car ist ein CPU-Kühler eingebaut. Über den Temperatursensor TMP 36 soll der Funduino die Temperatur im Autogehäuse ermitteln und den CPU-Kühler bei Überschreiten einer Temperaturgrenze zur Kühlung anschalten.

Einparkhilfe

In der Karosserie des Autos ist ein Ultraschallsensor am Heck eingebaut. Dieser übermittelt die Zeit des Schalls vom Auto zum Hindernis und zurück. Im Funduino werden diese Ergebnisse in Abstände umgerechnet. Je kleiner der Abstand ist, desto schneller soll der Piezo Speaker piepsen.

1.3 Materialliste für das Gesamtprojekt

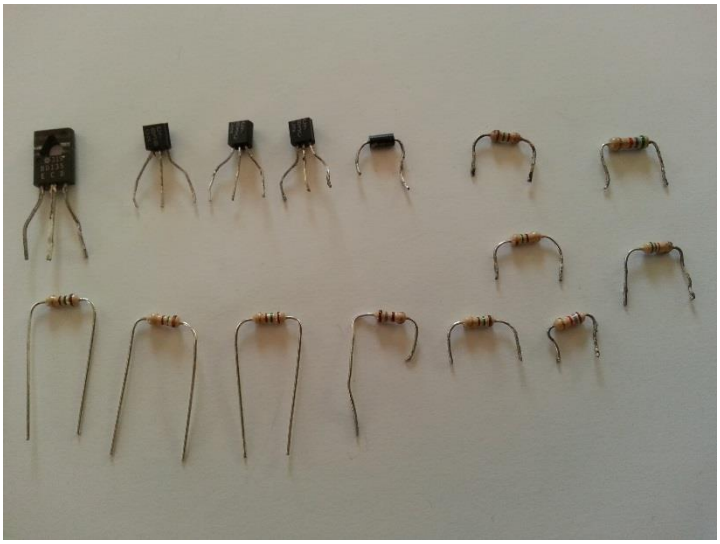


- 1 x Funduino Uno
- 1 x Funduino Set 3:
  - Kabel
  - LEDs
  - Widerstände
  - Sensoren
  - Steckbrett
  - uvm. *[1]
- Verwendete Programme/Software:
  - Arduino 1.0.6
  - Fritzing 0.9.0b
  - MIT App Inventor 2
  - SolidWorks 2013 Edition

- 1 x Servo MT955
- 1 x DC-Gleichstrommotor
- 1 x CPU-Kühler
- 1 x Android Smartphone
- Module:
  - 1 x Bluetooth Modul, HC-05
  - 1 x H-Brücke, Motortreiber Modul, L298N
- Sensoren:
  - 1 x Ultraschallsensor, HC–SR04
  - 1 x Temperatursensor, TMP36
  - 1 x Fotowiderstand 1kΩ
- USB Verbindungskabel:
  - 1 x USB Kabel kurz: USB A Stecker auf USB B Stecker
  - 1 x USB Kabel: USB A Buchse auf USB A Stecker
- Stromversorgung:
  - 2 x 9V NiMH-Akku
  - 8 x 1,5V Varta Energy AA Mignon Alkaline Batterie
  - 1 x Universal Profi-Schnell-Ladegerät (für NiMH-Akkus)
  - 1 x Batteriehalterung
  - 1 x 4,8V NiCD Battery Pack, incl. Ladestation
  - 2 x Batterieklemme
  - Multimeter KMM-940

*[1] genaue Angabe im Funduino-Shop, unter :
http://www.funduinoshop.com/epages/78096195.sf/de_DE/?ObjectPath=/Shops/78096195/Products/01-U3

- Elektronikbauteile:
  - Kabel (versch. Farben)
  - 6 x rote LED
  - 8 x weiße LED
  - 1 x blaue LED
  - 6 x 150Ω Widerstand
  - 1 x 100Ω Widerstand
  - 1 x 10kΩ Widerstand
  - 1 x 180Ω Widerstand
  - 1 x 1,8kΩ Widerstand
  - 2 x 5,6kΩ Widerstand
  - 1 x Diode, 1N4001
  - 3 x BC237c NPN Transistor
  - 1 x BD135 NPN Transistor
  - 1 x Schiebschalter 6 Pins 2xUM S106



- Werkzeug und Material zur Elektronikinstallation:
  - elektronischer Lötkolben
  - Gaslötkolben mit feiner Lötspitze
  - Lötfett
  - feiner und grober Lötzinn
  - Lochrasterplatte
  - Zangen/ Cuttermesser
  - Fädelstift mit Draht
  - Silberdraht
  - Isolierband und Kabelverbinder EVK2
  - Heißklebepistole
  - Multimeter, Oszilloskop

- Material und Werkzeug für den Chassis-/Karosseriebau:
  - 4 mm starke Sperrholzplatte
  - 8 mm starke Sperrholzplatte
  - Edelstahlgitter
  - Schrauben/Unterlagscheiben/Muttern
  - Holzleim/Silikon/Montageklebstoff
  - Dekupiersäge, Kreissäge
  - Bohrmaschine
  - Schraubzwingen
  - verschiedene Schleifwerkzeuge/ Feilen
  - Zeichenutensilien (Meterstab, Winkel, …)
  - Schraubenzieher/Kombizangen

II. Software- und Hardwareentwicklung

2.1 Die Smartphone App

2.1.1 MIT App Inventor 2 – Der Aufbau

Überlegung:

Zur Steuerung des Arduino Car möchte ich eine App auf dem Smartphone verwenden, um über Bluetooth mit dem Auto zu kommunizieren.

Da die App an das Arduino Car angepasst sein soll, muss ich selbst eine App erstellen.
Um eine „richtige“ App mit Programmiersprache wie z.B. Java zu erstellen, fehlen mir die Grundkenntnisse. Diese zu erlernen, würde den zeitlichen Rahmen des Halbjahresprojektes sprengen. Deswegen verwende ich für meine App einen Online-Baukasten, welche im Internet meist kostenlos angeboten werden.

Entschieden habe ich mich für die Software „MIT App Inventor 2“ von Google. Bei dieser Software benötigt man nur einen Google Account und kann dann von jedem PC auf seinen Arbeitsbereich zugreifen.

Im App Inventor kann man unkompliziert selbst eine Android App erstellen, welche speziell für selbstgestaltete Projekte ausgelegt sind.
Man entwirft in dem Programm seine App auf zwei Ebenen:
Dem Design Editor und dem Blocks Editor:

Design Editor:

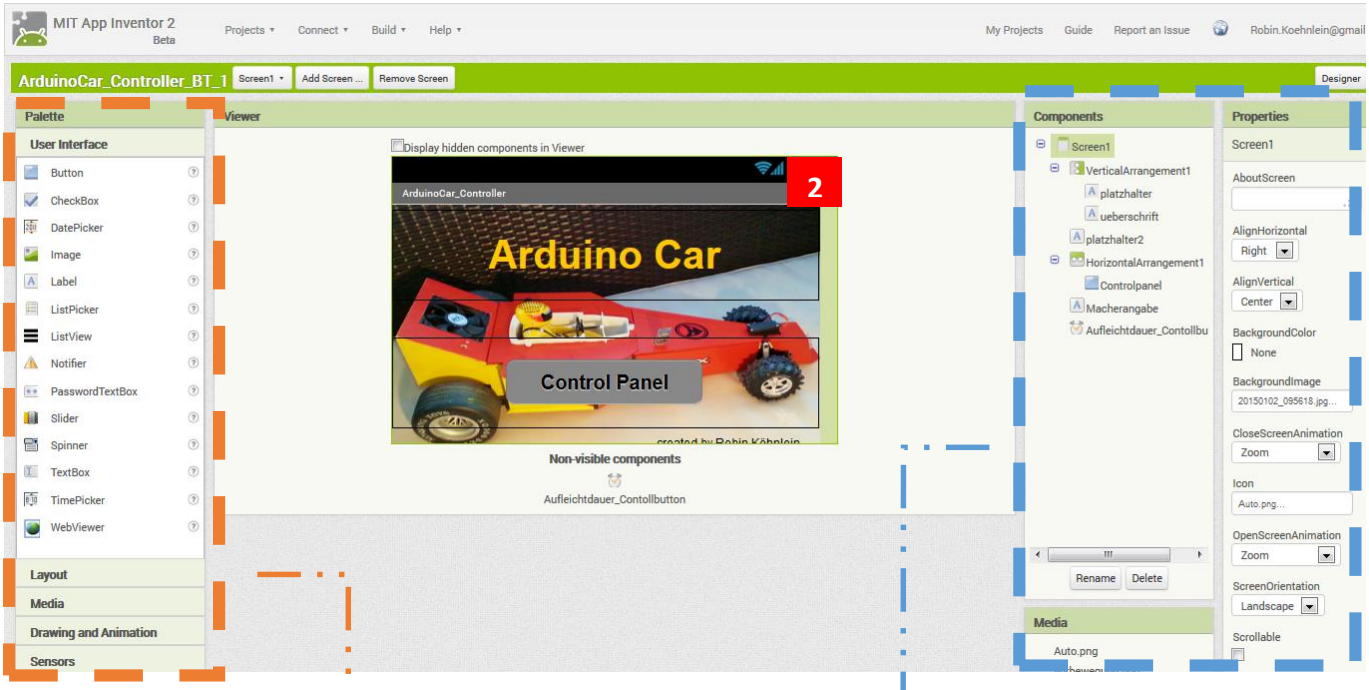
In Design Editor kann man das Layout der eigenen App gestalten. Der App Inventor gibt hierfür mehrere Vorlagen und Möglichkeiten vor, die per „Drag and Drop“ auf die Bildschirmdarstellung (Viewer) gezogen werden können. So kann man beispielsweise Bilder, Buttons, Schiebeschalter (Slider), etc. leicht in die Oberfläche der eigenen App einbauen.

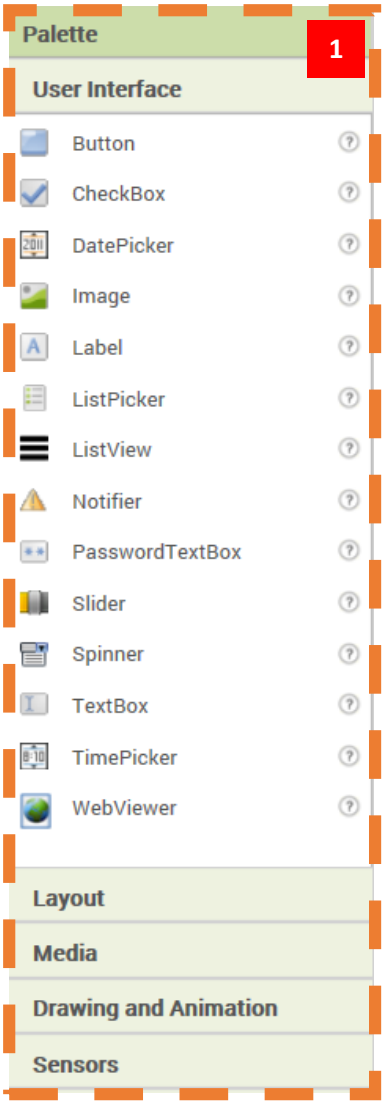
Aufbau des Design Editors:

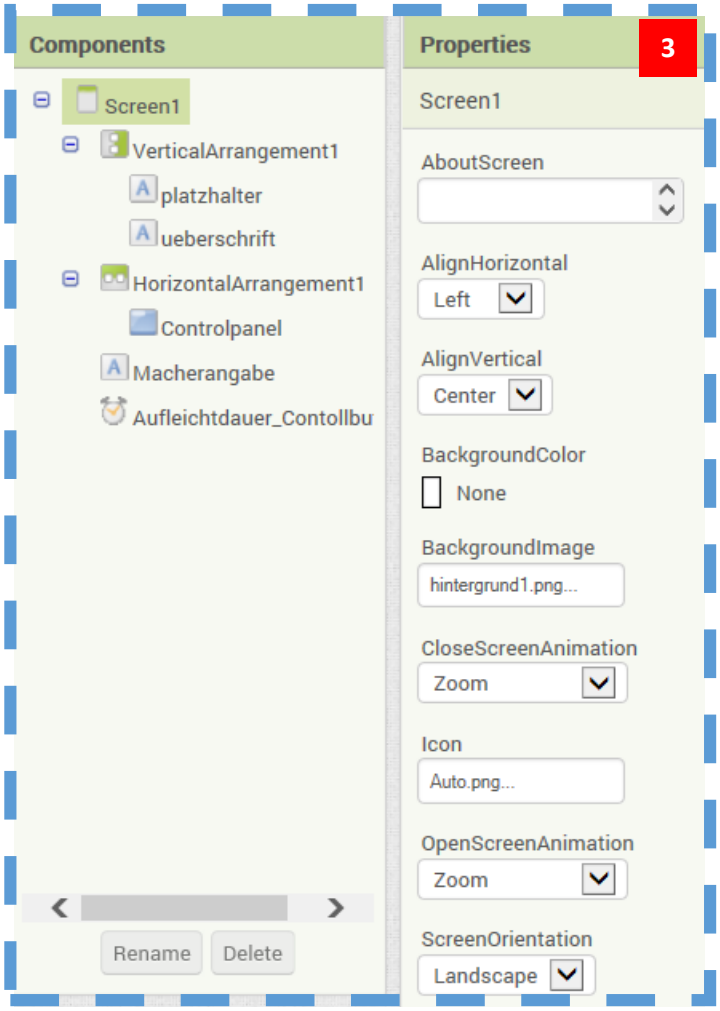
1 Palette: Hier befinden sich die Vorlagen für die Layout Gestaltung

2 Viewer: Ansicht des Smartphone Monitors, Bearbeitungsbereich für das Layout

3 Components/Properties: Hier sind die verwendeten Layout Komponenten aufgelistet und man kann sie in Ausrichtung, Größe und Darstellung bearbeiten


MIT App Inventor 2
Beta
Projects
Connect
Build
Help
My Projects
Guide
Report an Issue
Robin.Koehnlein@gmail
ArduinoCar_Controller_BT_1
Screen1
Add Screen ...
Remove Screen
Designer
Palette
User Interface
Button
CheckBox
DatePicker
Image
Label
ListPicker
ListView
Notifier
PasswordTextBox
Slider
Spinner
TextBox
TimePicker
WebViewer
Layout
Media
Drawing and Animation
Sensors
Viewer
Display hidden components in Viewer
ArduinoCar_Controller
Arduino Car
Control Panel
2
Non-visible components
Aufleichtdauer_Contollbutton
Components
Screen1
VerticalArrangement1
platzhalter
ueberschrift
platzhalter2
HorizontalArrangement1
Controlpanel
Macherangabe
Aufleichtdauer_Contollbu
Rename
Delete
Media
Auto.png
Properties
Screen1
AboutScreen
AlignHorizontal
Right
AlignVertical
Center
BackgroundColor
None
BackgroundImage
CloseScreenAnimation
Zoom
Icon
Auto.png...
OpenScreenAnimation
Zoom
ScreenOrientation
Landscape
Scrollable


Palette
1
User Interface
Button
CheckBox
DatePicker
Image
Label
ListPicker
ListView
Notifier
PasswordTextBox
Slider
Spinner
TextBox
TimePicker
WebViewer
Layout
Media
Drawing and Animation
Sensors


Components
Screen1
VerticalArrangement1
platzhalter
ueberschrift
HorizontalArrangement1
Controlpanel
Macherangabe
Aufleichtdauer_Contollbu
Rename
Delete
Properties
3
Screen1
AboutScreen
AlignHorizontal
Left
AlignVertical
Center
BackgroundColor
None
BackgroundImage
hintergrund1.png...
CloseScreenAnimation
Zoom
Icon
Auto.png...
OpenScreenAnimation
Zoom
ScreenOrientation
Landscape

Blocks Editor:

Im Blocks Editor wird das Programm bzw. die Hintergrundabläufe der App erstellt. Man kann in ihm die verwendeten Komponenten des Design Editors aufrufen und diesen in Verbindung mit vorgegebenen Logikfunktionen, mathematischen Funktionen und weiteren Verknüpfungen Aktionen zuordnen. Hier kann man z.B. festlegen, was bei dem Betätigen eines Buttons geschehen soll.
Der Blocks Editor ist sehr logisch aufgebaut und erinnert an ein Puzzle, da man das Programm der App durch vorgegebene Komponenten und Funktionen, in Form von einzelnen Blöcken, zusammensetzt.

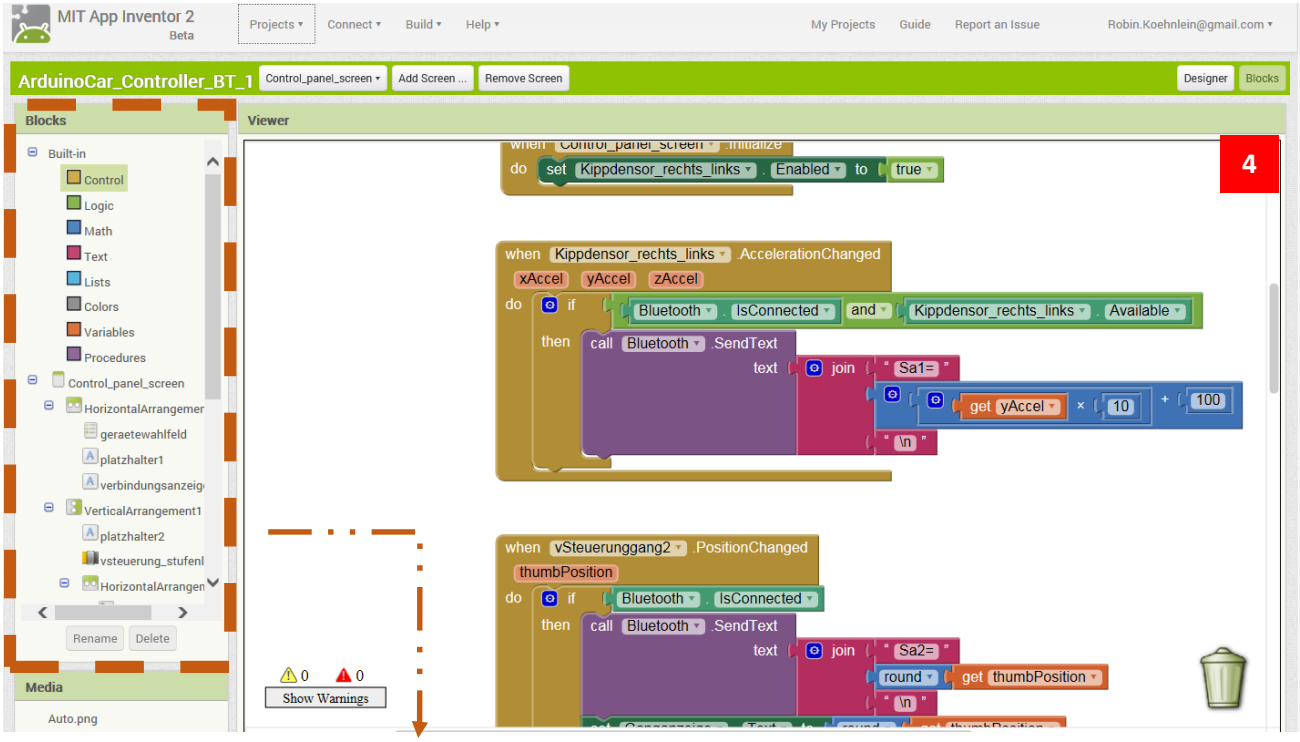
Aufbau des Blocks Editors:

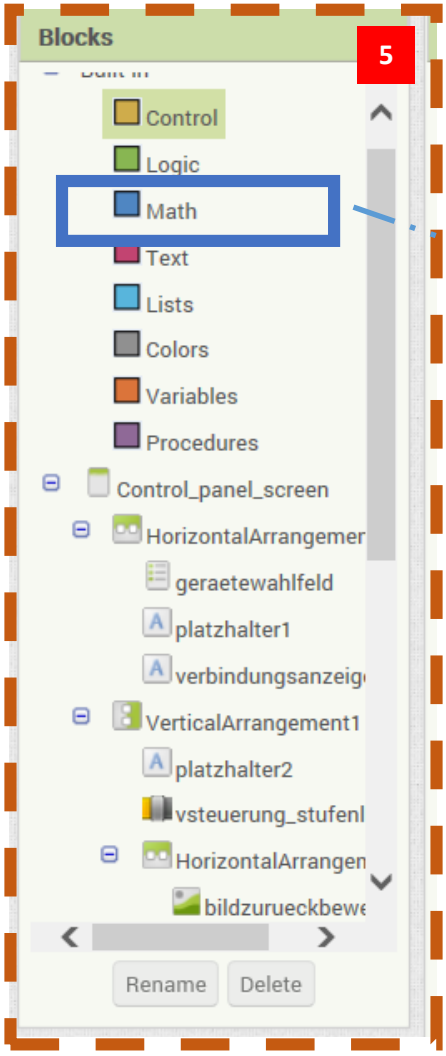
4 Viewer: Arbeitsbereich in dem das Programm aus einzelnen Blöcken zusammengesetzt wird

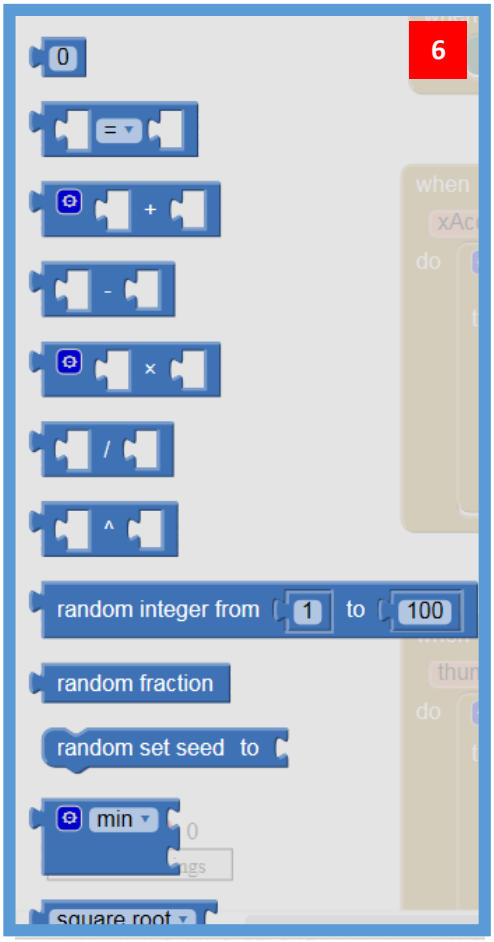
5 Blocks: hier befinden sich die vorgegebenen Blöcke für Logikfunktionen, mathematischen Funktionen …

6 Wird im „Blocks-Abschnitt“ eine Kategorie der vorgeschlagenen Funktion angeklickt öffnet sich ein weiteres Feld. Dieses enthält eine Auflistung aller möglichen Blöcke zu der dazugehörigen Kategorie.

Hier: Auflistung aller Blöcke für mathematische Funktionen

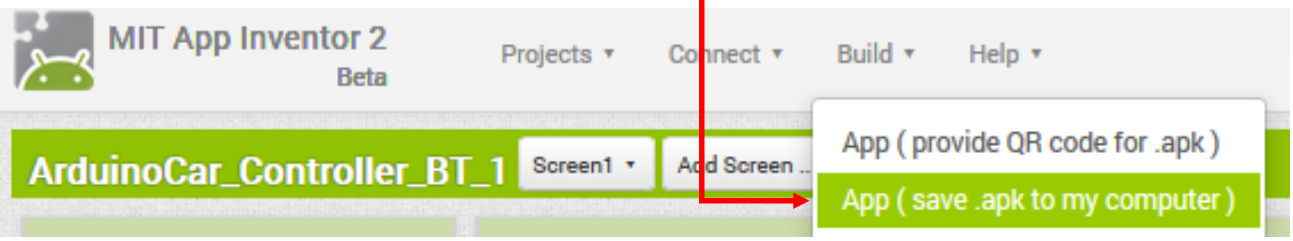

MIT App Inventor 2
Beta
Projects
Connect
Build
Help
My Projects
Guide
Report an Issue
Robin.Koehnlein@gmail.com
ArduinoCar_Controller_BT_1
Control_panel_screen
Add Screen ...
Remove Screen
Designer
Blocks
Viewer
4
Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Control_panel_screen
HorizontalArrangemer
geraetewahlfeld
platzhalter1
verbindungsanzeig
VerticalArrangement1
platzhalter2
vsteuerung_stufenl
HorizontalArrangen
Rename
Delete
Media
Auto.png
Show Warnings
when Kippdensor_rechts_links .AccelerationChanged
xAccel yAccel zAccel
do if Bluetooth . IsConnected and Kippdensor_rechts_links . Available
then call Bluetooth .SendText
text join " Sa1= "
get yAccel × 10 + 100
" \n "
when vSteuerunggang2 .PositionChanged
thumbPosition
do if Bluetooth . IsConnected
then call Bluetooth .SendText
text join " Sa2= "
round get thumbPosition
" \n "


Blocks
5
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Control_panel_screen
HorizontalArrangemer
geraetewahlfeld
platzhalter1
verbindungsanzeig
VerticalArrangement1
platzhalter2
vsteuerung_stufenl
HorizontalArrangen
bildzurueckbewe
Rename
Delete


6
0
=
+
-
×
/
^
random integer from 1 to 100
random fraction
random set seed to
min
square root

2.1.2 Testmodus der erstellten App auf dem Smartphone:

Um zu sehen, ob die App auf dem Smartphone funktioniert, muss man normalerweise zuerst die App als „.apk-Datei“ herunterladen und diese dann per USB Verbindung auf das Smartphone übertragen.



Da dies aufwendig ist und viel Zeit in Anspruch nimmt, kommt es nicht in Frage, dies nach jeder kleinen Veränderung der App zu machen. Hinzu kommt, dass die App dann auch jedes Mal neu installiert werden müsste.

Als Lösung hierfür gibt es im MIT App Inventor eine Funktion, die während der Bearbeitung eine Verbindung zum Smartphone per USB Kabel aufbaut. Dadurch kann man seinen Arbeitsbereich bzw. den Viewer jederzeit auf dem Smartphone Monitor sehen und das Layout individuell anpassen. Dafür ist die „MIT AI2 Companion“ App auf dem Smartphone notwendig. Diese kann kostenlos vom Playstore heruntergeladen werden.
Auf dem PC muss der „ai Starter“ geöffnet sein. Dies ist ein Programm, welches bei der Installation des MIT App Inventors enthalten ist und in Verbindung mit der Companion App auf dem Smartphone den Testmodus ermöglicht.

Problem: Aktivierung des USB-Debugging Modus

Mein Problem war, dass man für den Testmodus den „USB-Debugging Modus“ seines Smartphone aktivieren muss. Dieser ist auf einem Android Smartphone nicht leicht zu erreichen und die Suche ist kompliziert:

Schritte zum Aktivieren des USB-Debugging Modus:

Einstellungen – Optionen – Info zum Gerät – 7 x auf die „Build Nummer“ tippen – Entwickleroptionen (jetzt öffnen) – USB Debugging aktivieren

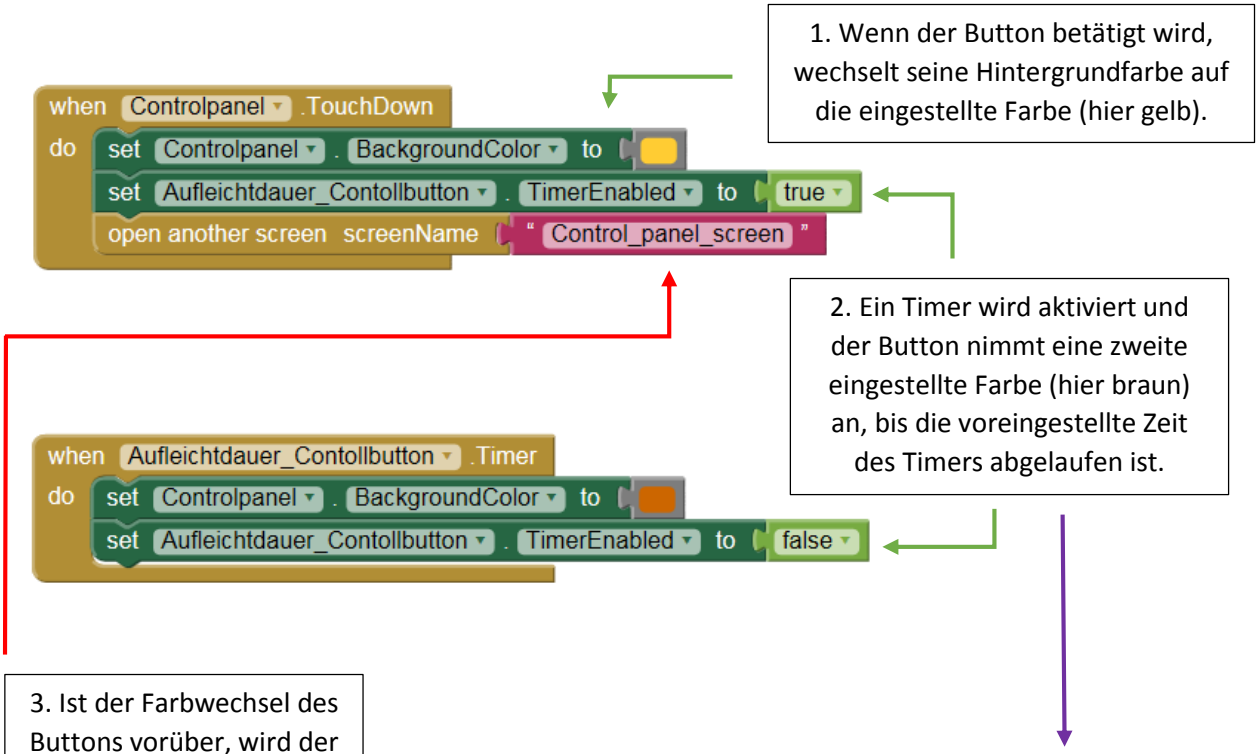
Kann man nach diesem Schritt den Test Modus immer noch nicht anwenden, könnte es am „Android USB driver“ liegen. Bei mir war dieser nicht aktuell. Um den „Android USB driver“ zu aktualisieren bzw. neu zu installieren muss dieser über den Geräte-Manager des PC installiert oder ein Update für den Driver durchgeführt werden.

Spätestens jetzt sollte der Testmodus auf dem Smartphone über die „Connect Funktion“ im MIT App Inventor 2 funktionieren.

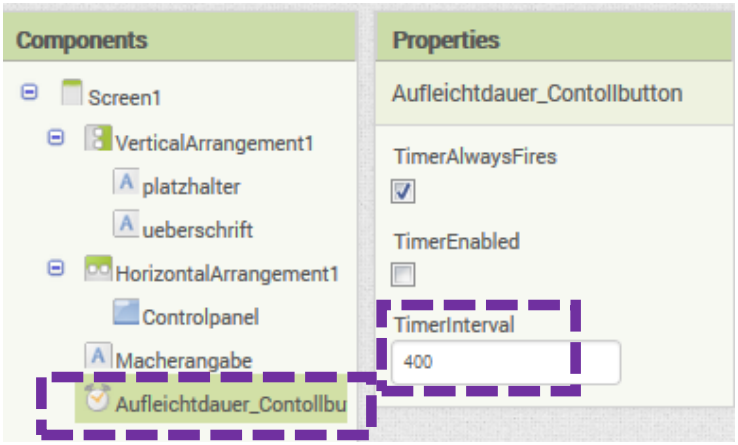
2.1.3 ArduinoCar_Controller_BT_1

Die App, die ich für mein Projekt erstellt habe, heißt „ArduinoCar_Controller_BT_1“.
Sie besteht aus zwei Screens. Im zweiten Screen, dem „Control_panel_screen“, befinden sich die Steuerelemente für das Arduino Car. Durch diese Steuerelemente soll man das Arduino Car stufenlos lenken und die Geschwindigkeit der Vor- bzw. Rückwärtsbewegung einstellen können.
Die Kommunikation zwischen App und Arduino Car erfolgt über Bluetooth. Dadurch muss eine Bluetooth Verbindung zwischen der Smartphone App und dem HC-05 Bluetooth Modul am Funduino aufgebaut werden. Über diese Bluetooth Verbindung sollen Signale an den Funduino übermittelt werden.

Um zum „Control Panel“ zu gelangen, also dem Screen, von dem die Bluetooth Verbindung erstellt wird, muss dieser erst geöffnet werden. Dies wird über Betätigung eines Buttons im ersten Screen erreicht.
Die Buttons in meiner App werden wie folgt im Blocks Editor programmiert:



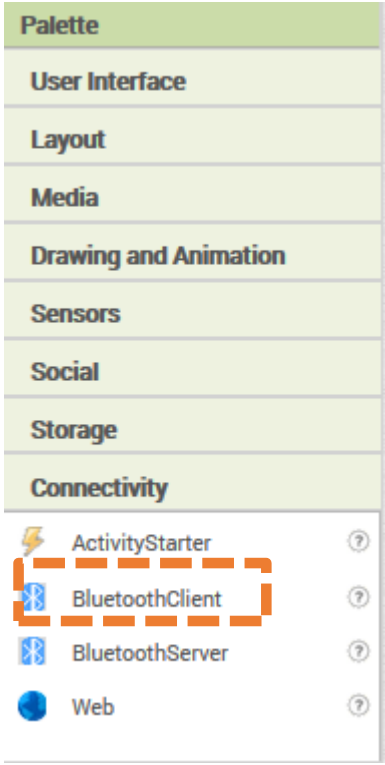
Darstellung des Timers und Einstellung der Timerzeit :



2.1.3.1 Die Bluetooth Verbindung

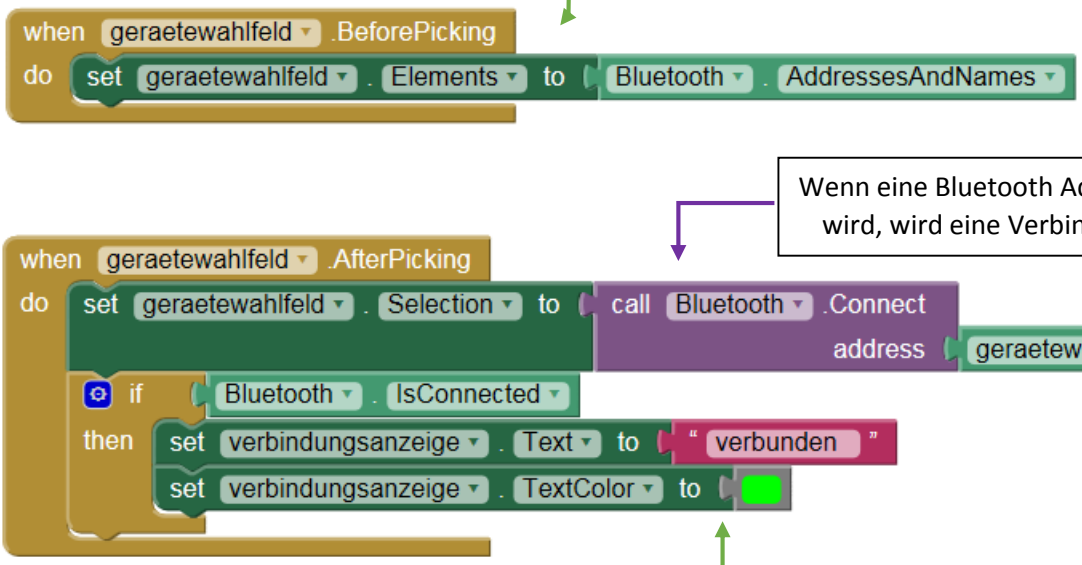
Hierfür gibt es im App Inventor eine Funktion, die das Smartphone mit Bluetooth fähigen Geräten verbindet. Voraussetzung hierfür ist, dass das Bluetooth fähige Gerät im Vorfeld gekoppelt wurde.

Diese Funktion ist einfach in die App einzubauen. Man findet sie als „non-visible component“ im Kommunikationsbereich der Palette im Design Editor. Diese kann von dort auf die Arbeitsfläche gezogen werden. In Verbindung mit einem „ListPicker“, also eine aufgelistete Auswahl der empfangenden Bluetooth Adressen, kann eine Adresse ausgewählt werden und die Bluetooth Funktion verbindet sich automatisch mit diesem Gerät, wenn dies möglich ist.



Programmierung im Blocks Editor:

Gerätewahlfeld (= ListPicker) listet alle Bluetooth Adressen auf

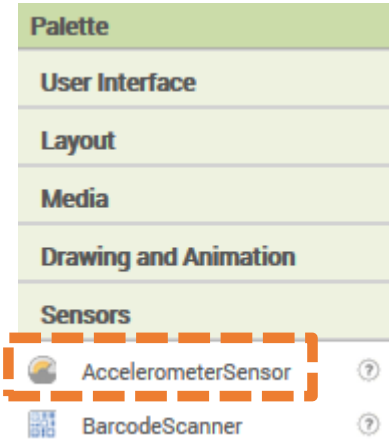


Wenn eine Bluetooth Adresse ausgewählt wird, wird eine Verbindung aufgebaut

Wenn eine Bluetooth Verbindung besteht wird der Text eines Textfeldes auf „verbunden“ geändert und erhält die Farbe grün

2.1.3.2 Steuerelemente:

Accelerometer Sensor:

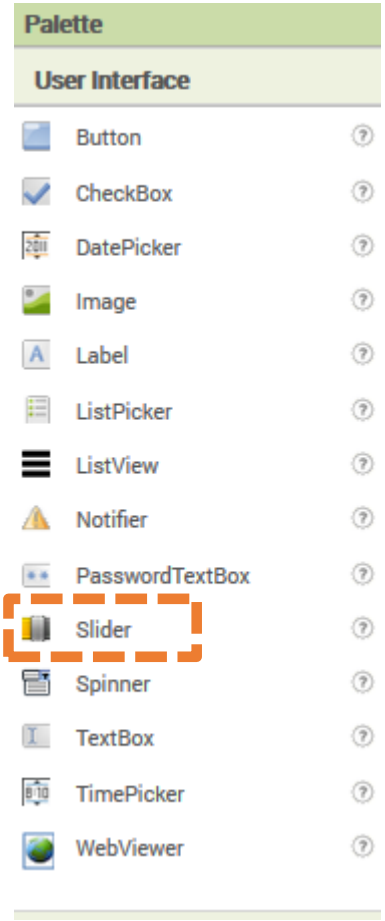


Dieser Sensor misst die Beschleunigung g in $\frac{m}{s^2}$. Dadurch kann durch die Messwerte des Accelerometer Sensors bestimmt werden, ob das Handy gekippt bzw. geneigt wird und wie stark diese Neigung ist. Da für meine Steuerung nur die vertikale Neigung des Smartphones von Bedeutung ist, müssen nur die „y-Accel – Werte“ des Sensors verarbeitet werden.

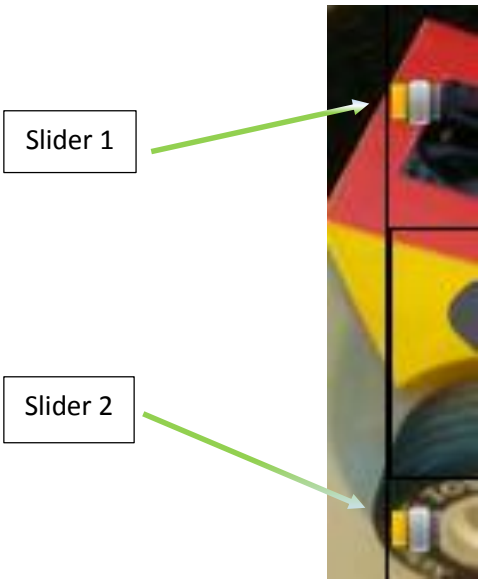
Der Accelerometer Sensor ist, wie die Bluetooth Funktion, eine unsichtbare Komponente der App und man erkennt ihn nicht auf dem Smartphone Monitor.

Symbol des Accelerometer Sensors:

Geschwindigkeitssteuerung:



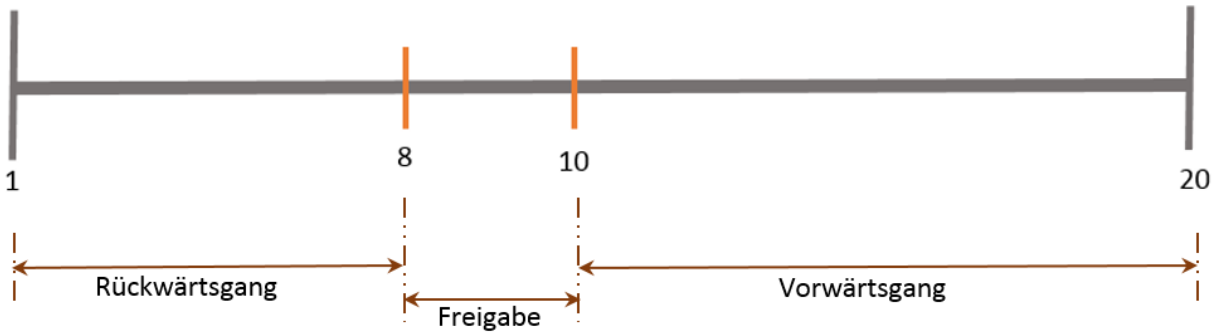
Die Geschwindigkeitssteuerung wird mit zwei Slidern geregelt. Diese Slider sind „Schiebeschalter“, welche je nach Position einen bestimmten Wert ausgeben. Der Wertebereich der Slider ist frei wählbar. Dadurch kann die Geschwindigkeitszu- und Abnahme so gestaltet werden, dass sie proportional zu den Sliderwerten ab- oder zunimmt.



1. Slider 1:

Dieser Slider ermöglicht die stufenlose Geschwindigkeitsregelung des Arduino Car. Er hat einen Wertebereich von 1 – 20 und gliedert sich in drei Bereiche auf: den Rückwärtsfahrbereich, den Freigabebereich und den Vorwärtsfahrbereich.
Im Vor- und Rückwärtsfahrbereich ist die Geschwindigkeit stufenlos regelbar und sie nimmt zu, je weiter der Slider von der Mitte entfernt steht.

Bereiche des Slider 1:



2. Slider 2:

Der Slider 2 ermöglicht eine nicht stufenlose Geschwindigkeitsregelung des Arduino Car. Er erkennt Werte von 1 – 6 und rundet dazwischenliegende Werte auf ganze Zahlen. Dadurch entstehen sechs Gänge, die jeweils eine bestimmte Bewegungsfunktion im Auto auslösen. In den Gängen 1 und 2 soll das Auto rückwärtsfahren, im dritten Gang soll sich das Auto nicht bewegen und im vierten, fünften und sechsten Ganz soll eine Vorwärtsbewegung stattfinden.

Bereiche/Gänge des Slider 2:

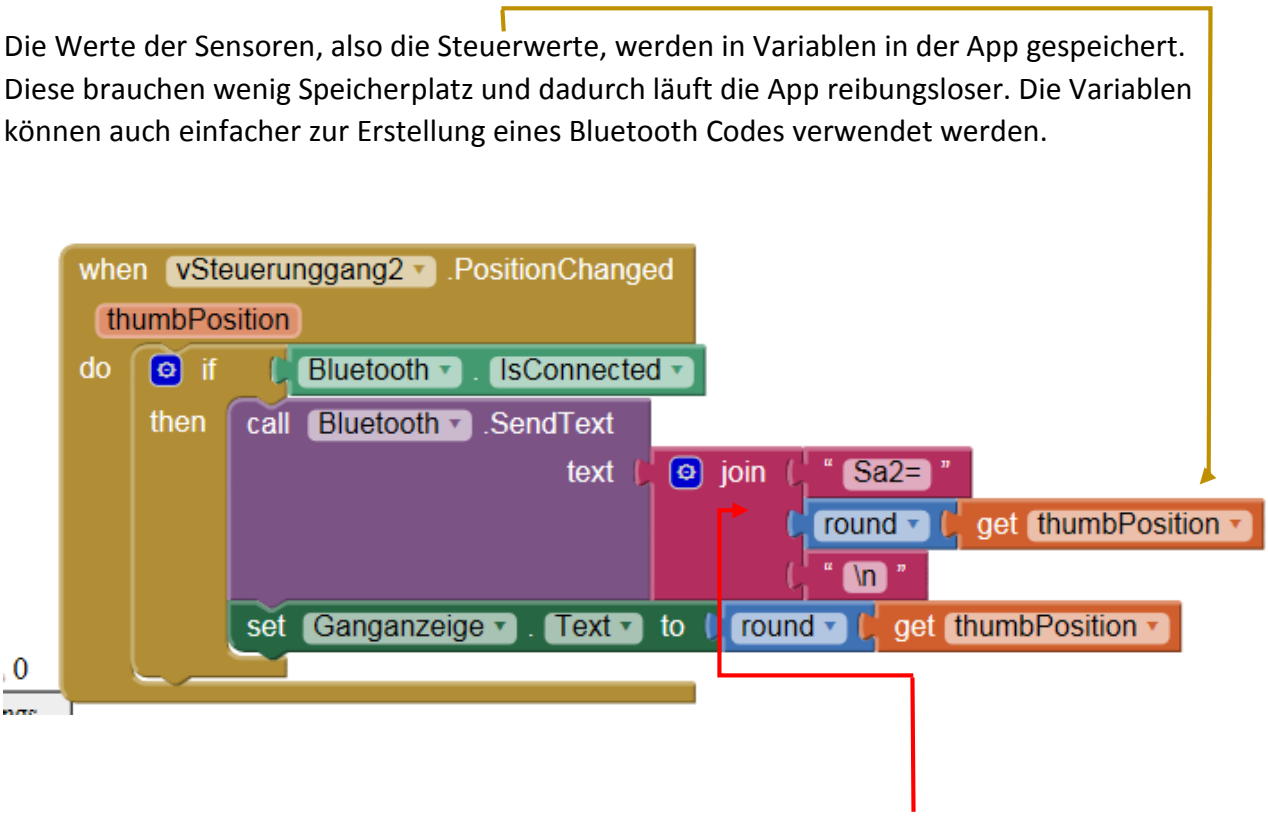
2.1.3.3 Erstellen eines Bluetooth Codes

Da die verschiedenen Aktionen und somit auch die Ergebnisse der Steuerelemente unabhängig voneinander sein müssen, muss ein Bluetooth Signal, bzw. ein Bluetooth Code an den Funduino gesendet werden. Dort soll dieser in die einzelnen Bestandteile zerlegt werden und eine klar zugeordnete Funktion hervorrufen.

Dies hat bei mir mehrere Probleme verursacht, da die einzelnen Werte der drei Steuerelemente jeweils Zahlen sind und somit der Funduino oft die Werte zu der richtigen Aktion nicht zuordnen konnte. Um diese Fehler bei der Auswertung der Bluetooth Signale zu umgehen, habe ich unter anderem versucht, für die Sliderwerte Symbole bzw. Buchstaben zu verwenden. Da über die Bluetooth Übertragung jedoch immer nur ein Wert auf einmal gesendet werden kann, war die Übertragung schon stark durch die Neigungswerte beansprucht, da der Accelerometer Sensor permanent die Smartphone Neigung ermittelt. Das Senden der Symbole von den Slidern war beeinträchtigt und unterdrückt. Es kam oft zu zeitlichen Verzögerungen in der App und zu Zusammenbrüchen der Bluetooth Übertragung.

Letztendlich habe ich eine Lösung gefunden um eine stetige, eindeutige Bluetooth Übertragung zu erhalten:

Die Werte der Sensoren, also die Steuerwerte, werden in Variablen in der App gespeichert. Diese brauchen wenig Speicherplatz und dadurch läuft die App reibungsloser. Die Variablen können auch einfacher zur Erstellung eines Bluetooth Codes verwendet werden.



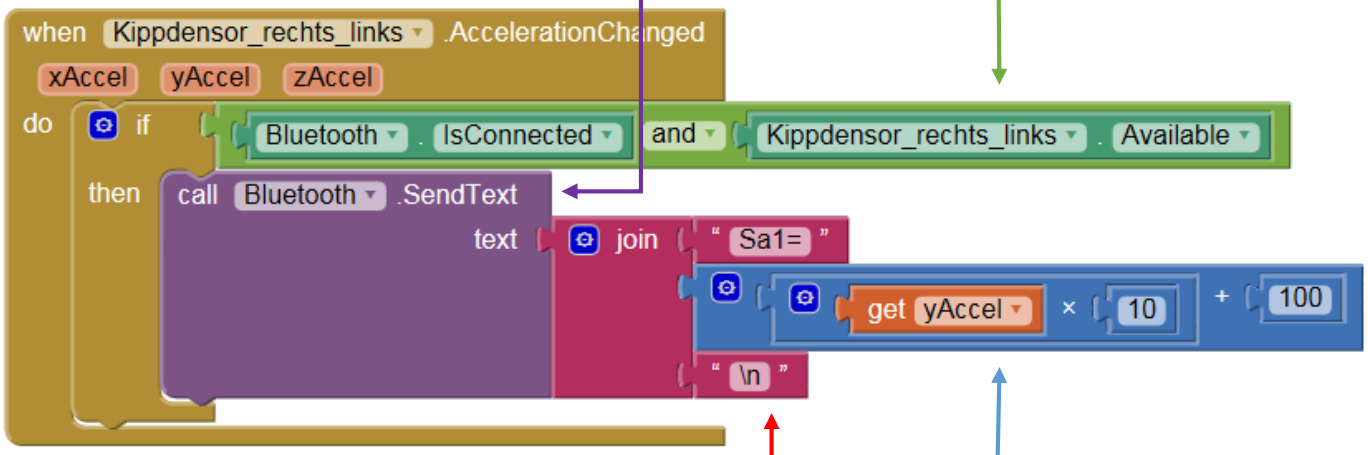
Die ermittelten Sensorwerte werden nun im Blocks Editor erkannt, ggf. bearbeitet und durch weitere Symbole zu einem Code zusammengesetzt (Bedeutung der einzelnen Bluetooth Codes siehe Seite 27).
Der entstehende Code wird über die aufgebaute Bluetooth Verbindung gesendet und im Funduino zerlegt und ausgewertet.

Programmteil für Erstellen und Übermittlung der Codes für die einzelnen Funktionen:

Accelerometer Sensor:

Der Accelerometer Sensor startet die Beschleunigungswerte des Smartphones bzw. die Neigung zu ermitteln, wenn eine Bluetooth Verbindung besteht und der Accelerometer Sensor des Smartphone verfügbar/bereit ist.

Die App sendet einen Text per Bluetooth an das verbundene, Bluetooth fähige Gerät. In meinem Projekt ist dies der Funduino.



Der gesendete Text bzw. der Bluetooth Code wird aus mehreren Teilen zusammengesetzt:

S a 1 = [Wert zwischen 5 und 195] \ n

Der Accelerometer Sensor ermittelt Werte zwischen minus 9,5 und plus 9,5. Um keine negativen Werte per Bluetooth an den Funduino senden zu müssen, werden die Ergebnisse des Kippsensors in einer mathematischen Funktion mit 10 multipliziert und mit 100 addiert. Dadurch erhält man Ergebnisse im Bereich zwischen plus 5 und plus 195.

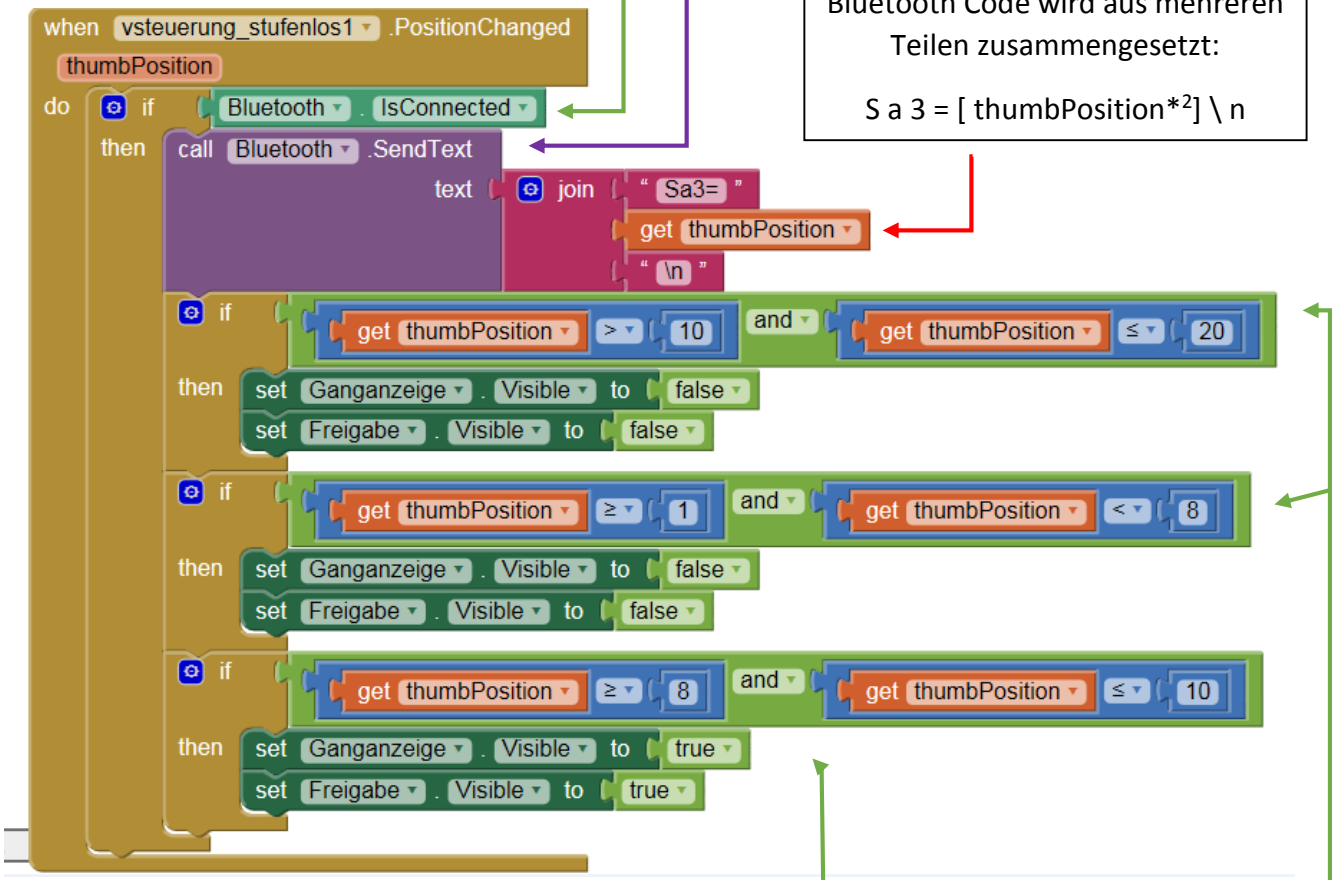
Slider 1:

Der Slider 1 startet seine Positionswerte zu ermitteln, wenn eine Bluetooth Verbindung besteht.

Die App sendet einen Text per Bluetooth an das verbundene, Bluetooth fähige Gerät.

Der gesendete Text bzw. der Bluetooth Code wird aus mehreren Teilen zusammengesetzt:

S a 3 = [thumbPosition*[2]] \ n



Steht der Slider 1 im Freigabebereich, werden die Benachrichtigungstextfelder für Freigabe und die Ganganzeige sichtbar dargestellt.

Wenn sich der Slider 1 im Rückwärtsfahrbereich oder im Vorwärtsfahrbereich (siehe Seite 16) befindet, wird ein Textfeld unterdrückt bzw. nicht sichtbar dargestellt. Dieses würde normalerweise den Gang bei nicht stufenloser Geschwindigkeitsregelung anzeigen. Dasselbe wird mit einem Benachrichtigungstextfeld für „Freigabe von Slider 2“ gemacht.

(→siehe Dominanz der stufenlosen Geschwindigkeitsregelung Seite 36)

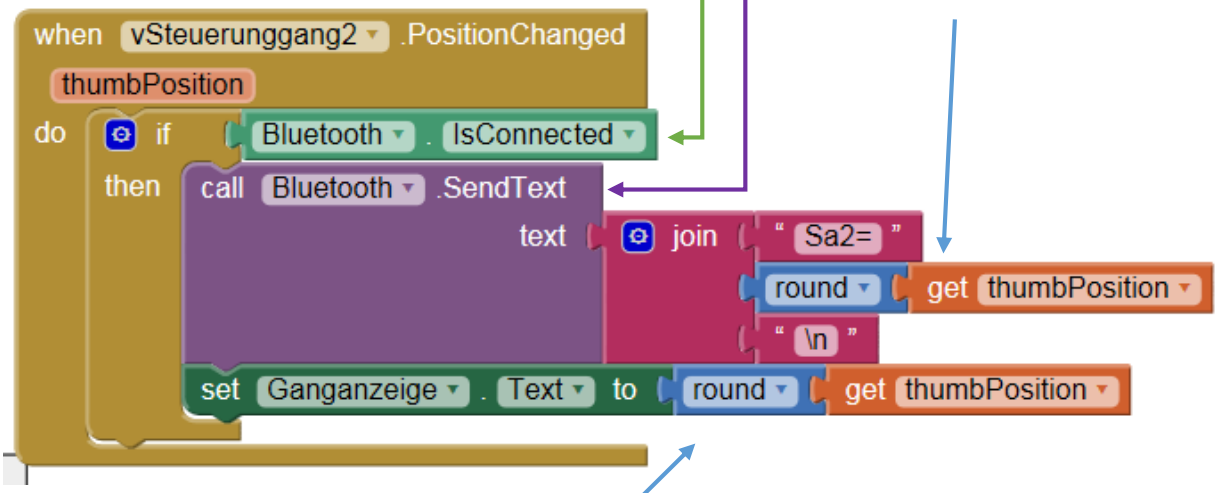
*[2] thumbPosition = momentan ermittelter Wert des Sliders 1

Slider 2:

Die App sendet einen Text per Bluetooth an das verbundene, Bluetooth fähige Gerät.

Der Slider 2 startet seine Positionswerte zu ermitteln, wenn eine Bluetooth Verbindung besteht.

Mittels einer mathematischen Funktion werden die ermittelten Werte des Slider 2 auf ganze Zahlen gerundet.



Die gerundeten Werte des Sliders 2 werden im Benachrichtigungstextfeld dargestellt. Sie zeigen an, in welchem Gang man sich befindet.

Der gesendete Text bzw. der Bluetooth Code wird aus mehreren Teilen zusammengesetzt:

S a 2 = [thumbPosition*[3]] \ n

*[3] thumbPosition = momentan ermittelter Wert des Sliders 2

<u>Gesamtes Layout der ArduinoCar_Controller_BT_1</u> (im Bearbeitungsmodus)

<u>Screen1:</u>

<u>Control_panel_screen:</u>

→Da die Screens im Bearbeitungsmodus dargestellt sind, werden sie verzerrt und falsch abgebildet. Diese Verzerrung kommt später auf dem Smartphone Monitor nicht vor.

2.2 Die verwendeten Module

2.2.1 HC-05 Bluetooth Modul

Die funktionsfähige App für das Smartphone kann über Bluetooth Signale bzw. Befehle zu einem Bluetooth fähigen Empfänger senden. Um mit dem Funduino drahtlos zu kommunizieren, benötige ich ein Bluetooth Modul. Dieses Modul kann sich mit der App über Bluetooth „pairen“ und die Signale empfangen, welche dann über die serielle Schnittstelle an Pin 0 und 1 in den Funduino eingelesen werden. Das Bluetooth Modul ist also ein „Übermittler“ für die Bluetooth Codes.

Für mein Projekt habe ich mich für das HC-05 Bluetooth Modul entschieden. Dieses Modul ist sehr preiswert, es kann für unter 10,00 € gekauft werden. Es hat auch den Vorteil, dass es als „Slave“ und „Master“ konfiguriert werden kann. Da die Datenübertragung in meinem Projekt immer einseitig von der App zum Bluetooth Modul stattfindet, muss das HC-05 Modul nur Signale empfangen. Es wird also als „Slave“ betrieben.

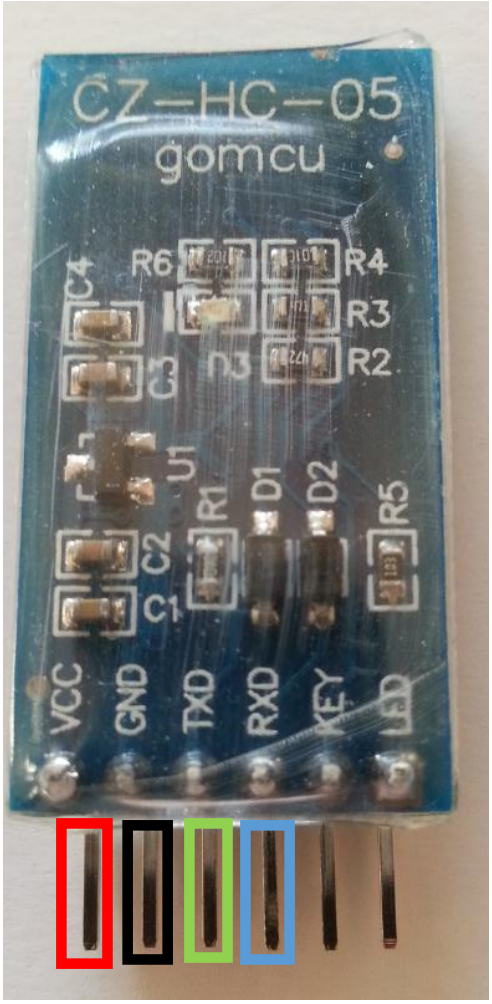
Das HC-05 Modul hat 6 Pins. Davon werden in meinem Projekt vier genutzt:

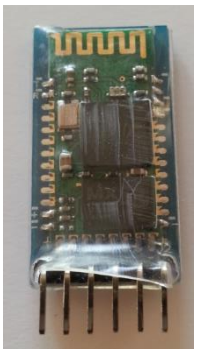
Der VCC-Pin wird an den 3,3 V Pin des Funduino angeschlossen. Dies ist die Betriebsspannung für das Modul.

Der Ground Pin wird ebenfalls mit dem Ground Pin vom Funduino verbunden.

Der TXD-Pin wird mit dem RX-Pin (= Pin 0) des Funduino verbunden.

Der RXD-Pin wird mit dem TX-Pin (= Pin 1) des Funduino verbunden.





Dadurch, dass ich den TXD-Pin des Moduls mit dem Pin 0 und den RXD-Pin mit dem Pin 1 des Funduino verbinde, nutze ich die Standardanschlüsse für eine serielle Schnittstelle des Funduino um die Bluetooth Signale einzulesen.
Hieraus ergibt sich ein Problem, denn diese serielle Schnittstelle des Funduino wird standartmäßig für den USB Datenverkehr mit dem PC beim Uploaden eines Programms oder bei Benutzung des Serial Ports verwendet.
Es besteht zwar die Möglichkeit, über eine Softserial-Library, die serielle Verbindung für das HC-05 Modul über zwei andere Ports des Funduino zu erstellen, da ich allerdings alle Pins für mein Projekt benötige, ist dies nicht möglich. Bei neuem Uploaden des Programms per USB muss aufgrund dessen der VCC-Pin des HC-05 Moduls von der Betriebsspannung 3,3 V getrennt werden. Sonst würde dies zu Fehlern beim Uploaden führen.

Datenübertragung an das HC-05 Modul und Einlesen der Daten in den Funduino:

Wenn das HC-05 Modul, wie auf Seite 22 ff. beschrieben, angeschlossen ist, ist es bereit eine Bluetooth Verbindung einzugehen. Dies erkennt man an einem Blinken der LED am Bluetooth Modul.
Das HC-05 Modul erscheint in der App als mögliches Gerät für eine Bluetooth Verbindung. Wird es in der App ausgewählt, wird das Smartphone mit dem HC-05 Modul verbunden, dies erkennt man an einer Änderung des Blinkverhaltens der Signalleuchte. Der Datenaustausch über Bluetooth findet statt und die Werte können in den Funduino über die serielle Schnittstelle übertragen werden.

Programmabschnitt der Arduinosoftware für die Ermittlung der Bluetooth Daten vom HC-05 Modul:

Um die Werte des HC-05 Moduls einzulesen muss die serielle Datenübertragung mit der Baudrate 9600 begonnen werden. Danach können die empfangenen Bluetooth Codes über „Serial-Befehle“, wie Serial.read(), Serial.find() etc., im Programm eingelesen werden:

```
void loop()
{
  zeit = millis();

  if(testlauf == false){

    if(Serial.find(STARTBEFEHL)){

      signalLED();

      while(Serial.available() < 1);
      {
        char action = Serial.read();
        int aktionsnummer = Serial.parseInt();
        int steuerwerte = Serial.parseInt();
```

Bluetooth Signal wird über serielle Schnittstelle eingelesen und verarbeitet

2.2.2 Das L298N Motortreiber Modul

Problem:

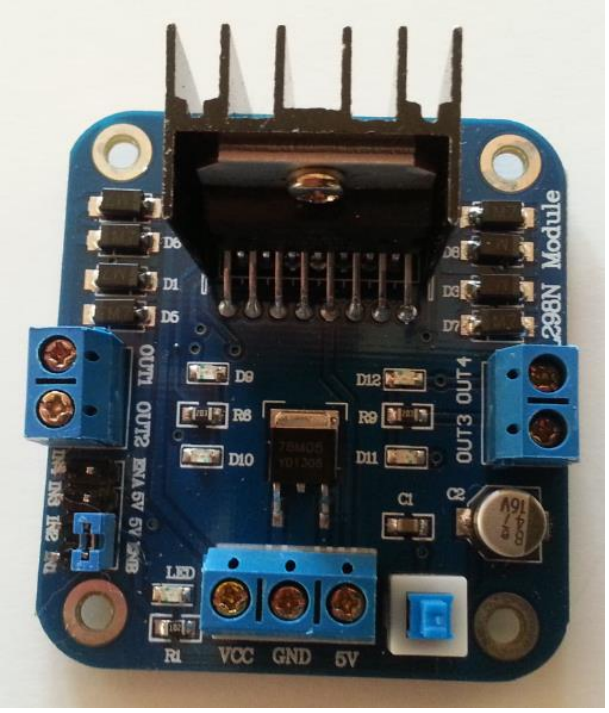
Der Motor soll im Links- und Rechtslauf gesteuert werden können. Da bei meinem DC-Gleichstrommotor dies durch Umkehrung der plus und minus Anschlüsse der Spannungsversorgung erreicht wird, ist dies mit einem Transistor als Schalter für größere Stromstärken nicht möglich.

Lösung:

Der Motor muss in der Geschwindigkeit und in der Drehrichtung gezielt gesteuert werden können. Hierfür verwende ich das Motortreiber Modul L298N.
Dieses Modul ist für die benötigte Stromstärke meines Motors geeignet:

ermittelte Daten des Motors	Daten des L298N Moduls	
Stromstärke …	Stromversorgung: 5V – 35 V	OK
	Steuerstrom : ca. 36 mA	OK
… beim Anfahren ≈ 1,7 A … bei reibungslosem Fahren ≈ 0,5 A … bei Fahren mit Widerstand ≈ 1,2 A – 1,3 A	Spitzenstrom ≈ 2 A	OK

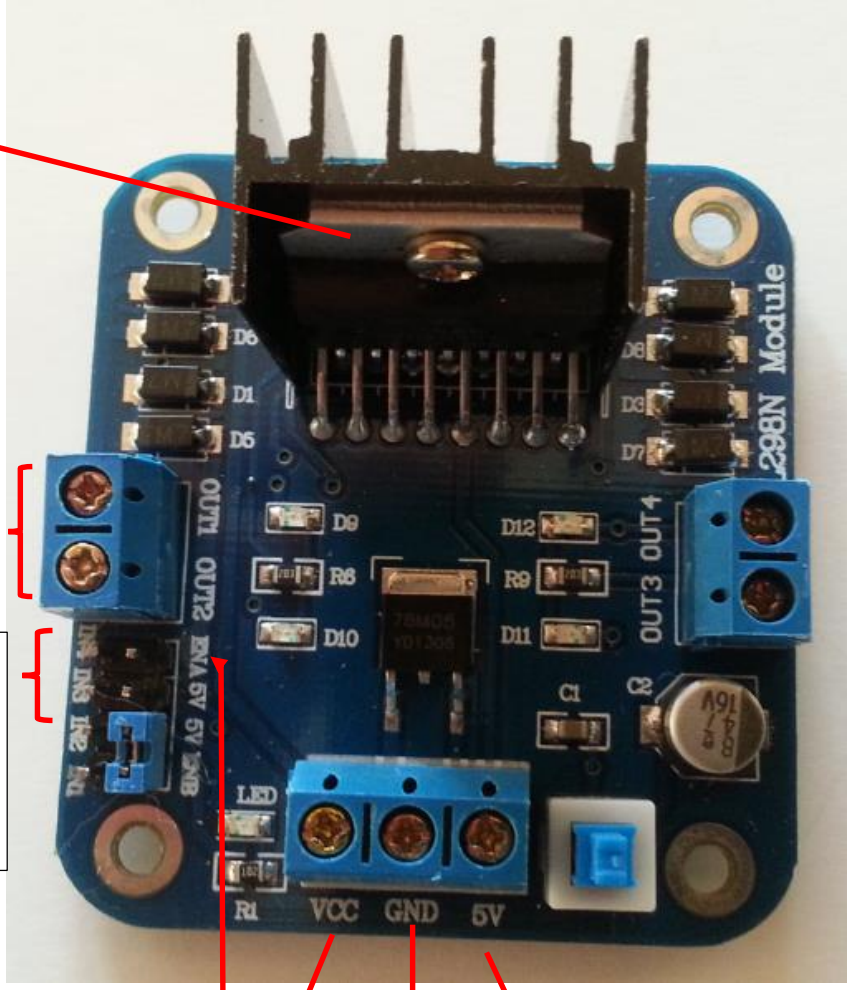
Mit dem L298N Modul kann die Drehrichtung des Motors beeinflusst werden und es hat einen Anschluss für PWM-Signale*[4]. Durch sie kann die Geschwindigkeit geregelt werden. Da die Freilaufdioden in diesem Modul enthalten sind, benötige ich diese zum Schutz des Motors nicht mehr zusätzlich. Somit ist dieses Modul optimal für mein Projekt und meine Motoransteuerung geeignet. Es ist für ca. 10,00€ zu erhalten.



*[4] PWM = Pulsweitenmodulation → bei PWM-Signalen wird das Signal im Wechsel auf „HIGH“ und auf „LOW“ gesetzt. Die Dauer der „HIGH – Impulse“ ist variabel.

Anschluss des L298N Moduls und des Motors:

Da das Motortreiber Modul für zwei unabhängige Motoren ausgelegt ist, verwende ich nicht alle Anschlüsse des Moduls.

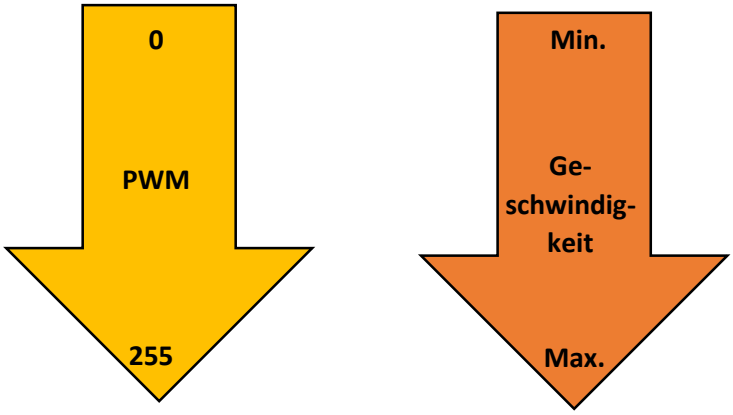


Funktionsweise des L298N Moduls:

Die Drehrichtung des Motors wird über die Anschlüsse IN1 und IN2 gesteuert:

Zustand IN1 / Pin4	Zustand IN2 / Pin2	Motorfunktion
LOW	LOW	Motor stoppt
LOW	HIGH	Motor dreht rückwärts
HIGH	LOW	Motor dreht vorwärts
HIGH	HIGH	Motor stoppt

Die Geschwindigkeit des Motors wird hierbei durch den ENA-Anschluss bzw. den Pin3 am Funduino gesteuert. Man kann über diesen Anschluss ein PWM-Signal*[4] an das Modul senden und dies beeinflusst die Geschwindigkeit des Motors. Je höher das PWM-Signal ist, desto höher ist auch die Umdrehungsgeschwindigkeit des Motors.



Programmierung zur Steuerung des L298N Moduls:

Im Programm wird der Motor über das Motortreiber Modul, wie in der Funktionsweise beschrieben, gesteuert:

```
void stufenlos_back()
{
  v_stufenlos= map (sliderl,1,7.9,200,0);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  analogWrite(turbo,v_stufenlos);
  freigabe=false;
}
```

Zur Drehrichtungsbestimmung des Motors wird IN1 und IN2 auf HIGH oder LOW gesetzt.

Die Geschwindigkeit des Motors wird über den ENA Anschluss (hier: turbo) über ein PWM-Signal gesteuert.

*[4] PWM = Pulsweitenmodulation

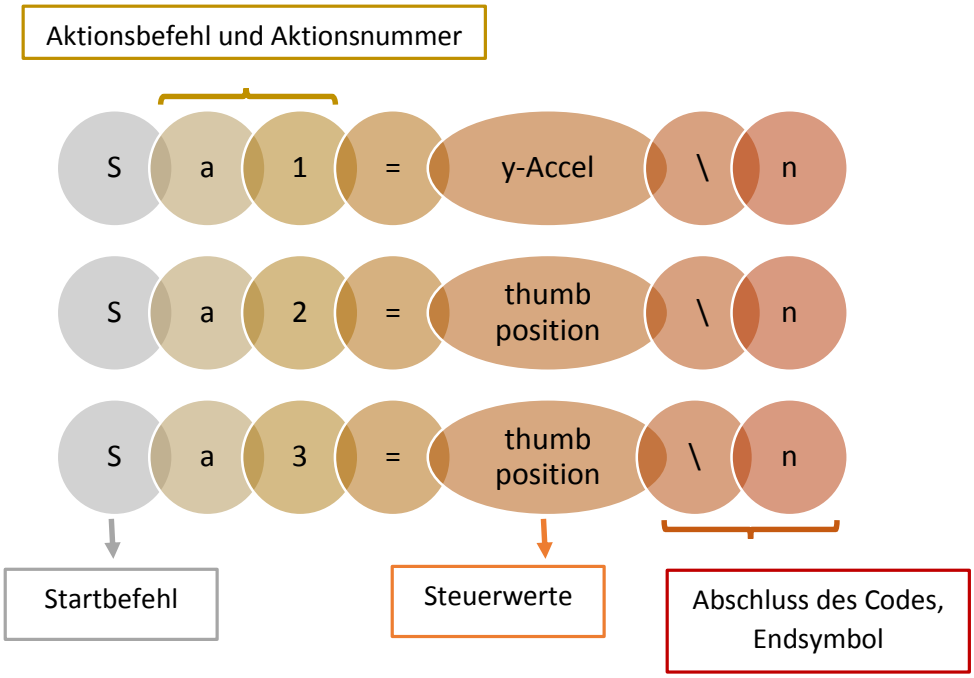
2.3 Die Projekteinzelteile

2.3.1 Die Lenkung / Steuerung

Die Lenkung und Steuerung erfolgt über die Smartphone App. Diese sendet bei Betätigung der Steuerelemente verschiedene Bluetooth Codes, die in den Funduino über das HC-05 Bluetooth Modul eingelesen werden. Der Verbindungsaufbau und die Einstellung des HC-05 Moduls sind auf Seite 22 ff. beschrieben. Im Funduino wird der Code in die einzelnen Bestandteile „zerlegt“, Schritt für Schritt untersucht und eine zugeordnete Aktion ausgeführt.

2.3.1.1 Aufbau und Bedeutung der Bluetooth-Codes:

Im Folgenden sind die 3 Möglichkeiten dargestellt, die bei der Bluetooth Signalübertragung auftreten können:



S = Startbefehl

→ Symbol um Bluetooth Daten im Funduino zu empfangen und einzulesen
→ erst jetzt wird der Code eingelesen und bearbeitet

a = Aktionsbefehl

→ eine Aktion wird gestartet
→ darauffolgende Zahl, die Aktionsnummer, definiert welche Steuerelemente die Steuerwerte senden

y-Accel / thumb position / thumb position = Steuerwerte

→ Nachdem durch den Aktionsbefehl klar wird, zu welcher Aktion der Bluetooth Code gehört, wird nun der Sensorwert des passenden Steuerelements der App eingelesen und später im Funduino Programm bearbeitet.

Programmteil für Einlesung / Bearbeitung der Bluetooth Codes:

```
void loop()
{
  zeit = millis();                                  // millis()-Zeit gespeichert, läuft ab Einschalten in ms durch

  if(testlauf == false){                            // Ausschalten der BT Code Bearbeitung bei Sensorentest

    if(Serial.find(STARTBEFEHL)){                   // für BT-Steuerung "S" des BT Codes notwendig

      signalLED();

      while(Serial.available() < 1);                // Warten bis BT Code ankommt
      {
        char action = Serial.read();                // Einlesen BT Code
        int aktionsnummer = Serial.parseInt();      // Aktionsnummer des BT Codes ermitteln
        int steuerwerte = Serial.parseInt();        // Steuerwert des BT Codes ermitteln

        if(action == AKTIONSBEFEHL)                 // wenn "a" in BT Code gesendet wird, Aktion ausgeführt
        {
```

Nachdem die Bluetooth Codes im Funduino angelangt sind, werden diese verarbeitet:

I. Der Funduino startet erst die Verarbeitung der Codes, wenn er das „S“ des Signals erkennt.

II. Der Funduino „wartet“ auf das „a“ bzw. den Aktionsbefehl und die dazugehörige Aktionsnummer. Je nach Aktionsnummer springt er in einen anderen Programmteil:

Aktionsbefehl	a	a	a
Aktionsnummer	1	2	3
sendendes Steuerelement	Accelerometer Sensor	Slider 2	Slider 1
dazugehöriger Steuerwert	y-Accel	thumb position (von Slider 2)	thumb position (von Slider 1)
möglicher Wertebereich der Steuerwerte	5 – 195	1 – 6	1 - 20
ausgeführte/bear-beitete Aktion	Lenkung	nicht stufenlose Geschwindigkeits-regelung (Gänge)	stufenlose Geschwindigkeits-regelung

2.3.1.2 Lenkung:

Ist die Aktionsnummer des Bluetooth Codes „1“, so werden die Werte des Accelerometer Sensors eingelesen und in der Variable „neigung“ gespeichert. Der Funduino transformiert/ändert nun die „neigung- Variable“ in einen Lenkwert/ einen Drehwinkel für den Servo. Diese transformierte „neigung - Variable“ wird später direkt in den Servo an der Vorderachse geschrieben, da sie nun den Anfahrbefehl für diesen beinhaltet.
Die Transformation/ Änderung geschieht durch eine „map-Funktion“. Hierdurch wird verhältnismäßig ermittelt, wo der eingelesene Steuerwert in dem Wertebereich von +5 bis +195 steht. Genau dieser Standort in dem Intervall [+5 ; +195] wird auf das Intervall der möglichen Lenkwerte [+ 75 ;+105] übersetzt. Der Lenkwert bzw. die veränderte „neigung - Variable“ wird also durch Transformation des eingelesenen Steuerwertes in einen neuen Wertebereich, bzw. ein neues Intervall, bestimmt. Der errechnete Lenkwert wird nun an den Servo ausgegeben, dieser bringt die Lenkachse des Autos in die errechnete Position.

Ein Beispiel um dies zu verstehen:

	Eingelesener Steuerwert	Lenkwert, bzw. transformierte „neigung-Variable“
möglicher Wertebereich/ Intervall	[+5 ; +195]	[+75 ; +105]
Wert	100 →	90
Erklärung	Der Wert 100 steht genau in der *Mitte* des Intervalls [+5 ; +195] →	Es wird nun der Wert gesucht, der ebenfalls in der *Mitte* des Intervalls [+75 ; +105] steht . Dies ist der Wert 90.
Verhältnismäßigkeit	Stellung des Smartphones bei diesem gesendeten Steuerwert : keine Neigung ↔	Stellung des Servos bei dem errechneten Lenkwert: Keine Neigung, keine gefahrene Kurve

Sieht man sich die Tabelle noch einmal an, erkennt man, dass die Transformation der „neigung-Variablen“ in ein anderes Intervall sinnvoll ist. Der „map-Befehl“ wirkt wie eine Übersetzung des eingelesenen Steuertwertes in einen Lenkwert, der an den Servo ausgegeben werden kann.
Da die Steuerwerte hier die Smartphone Neigung angeben, verändert sich die Neigung des Servos proportional zu der Neigung des Smartphones. Dies kann man in der letzten Zeile der Tabelle erkennen, denn wenn das Smartphone nicht geneigt wird, wird an den Servo der Winkel 90° ausgegeben. Da der Servo so eingebaut ist, dass das Auto bei 90° geradeaus fährt, tritt kein Lenkeffekt des Arduino Car bei einem ungeneigten Smartphone auf.
Lenkung und Smartphone Neigung verhalten sich gleich.

Beschränkung des Lenkeinschlags:

Das Intervall der Lenkwerte reicht nur von +75 bis +105, da ich eine Lenkachse aus einem anderen RC-Car in meinem Arduino Car verbaut habe. Diese Lenkachse lässt leider nur eine geringe Neigung der Vorderräder zu. Deswegen habe ich durch mehrerer Probeläufe versucht herauszufinden, wie der maximale Einschlagbereich der Vorderachse bzw. der Lenkachse ist, um diesen komplett auszunutzen. Herausgefunden habe ich, dass mit meinem Servoeinbau und meiner Übersetzung des Servos auf die Lenkachse der maximal mögliche Drehwinkel für den Servo zur einen Seite +75 und zur anderen Seite +195 ist.

Für die Ansteuerung des Servos verwende ich die Servo Library, welche schon in der Arduino 1.0.6 Programmiersoftware enthalten ist. Durch sie kann unkompliziert der Winkel angegeben werden, der die Position des Servos definiert.

Programmteil für Bearbeitung der Lenkfunktion:

```
if(action == AKTIONSBEFEHL)                    // wenn "a" in BT Code gesendet wird, Aktion ausgeführt
{
  if(aktionsnummer == 1)                       // Befehl im BT Code für Lenkung
  {
    neigung = steuerwerte;                     // einlesen Smartphone Neigung
    if (neigung >=5 && neigung <= 195)
    {
      neigung = map(neigung, 5,195, 75, 105);   // Umrechnung Smartphone Neigung in Lenkwert für Servo
      lenkung.write(neigung);                   // Lenkwert an Servo ausgeben --> Auto lenkt
    }
  }
```

2.1.3.3 Stufenlose Geschwindigkeitsregelung:

Ist die Aktionsnummer des eingelesenen Bluetooth Codes „3“, so werden die Werte des Slider 1 eingelesen und in der Variable „slider1“ gespeichert.
Der Funduino vergleicht in meinem Programm, in welchem Wertebereich der eingelesene Wert liegt und führt danach ein bestimmtes Unterprogramm aus, welches eine Funktion im Arduino Car steuert:

```
if(aktionsnummer == 3 )              // Befehl im BT Code für stufenlose Geschwindigkeitssteuerung
{
  slider1= steuerwerte;              // Einlesen Wert von Slider 1

  if(slider1> 10 && slider1 <=20){   // Verarbeitung des Wertes von Slider 1 --> in Begewungsaktion des Autos
    stufenlos_vor();
  }

  else if( slider1>=1 && slider1<8){
    stufenlos_back();
  }

  else if (slider1>=8 && slider1<=10){
    stufenlos_freigabe();
  }
}
```

In der darauf folgenden Tabelle erkennt man, welche Funktionen bzw. welches Unterprogramm ausgeführt wird, wenn der eingelesene Steuerwert in einem der drei Bereiche liegt.
Das Aufrufen der Unterprogramme, je nach Größe des Steuerwertes, wird durch „if-Befehle“ im Steuerprogramm erreicht.

Wertebereich w	Eingelesener Steuerwert	ausgeführte Funktion	Name des Unterprogrammes
$\mathbf{1 \leq w < 8}$	[1 ; 8 [	Arduino Car fährt zurück	stufenlos_back()
$\mathbf{8 \leq w \leq 10}$	[8 ; 10]	Freigebe für Slider 2 / Gangsteuerung	stufenlos_freigabe()
$\mathbf{10 < w \leq 20}$	] 10 ; 20]	Arduino Car fährt vorwärts	stufenlos_vor()

stufenlos_back():

Hier wird der in der Variable „slider1“ gespeicherte Steuerwert des Slider 1 in ein PWM-Signal umgerechnet. Dieses PWM-Signal wird zur Geschwindigkeitsregelung des Motors an den ENA Anschluss des L298N Motortreiber Moduls ausgegeben. Da das Auto rückwärtsfahren soll, wird IN1 „LOW“ und IN2 „HIGH“ gesetzt.

```
void stufenlos_back()
{
  v_stufenlos= map (slider1,1,7.9,255,130);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  analogWrite(turbo,v_stufenlos);
  freigabe=false;
}
```

stufenlos_freigabe():

Die Gangsteuerung durch Slider 2 wird freigegeben (siehe Seite 36)

```
void stufenlos_freigabe()
{
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
  analogWrite(turbo,LOW);
  freigabe=true;
}
```

stufenlos_vor():

Hier wird der in der Variable „slider1“ gespeicherte Steuerwert des Slider 1 ebenfalls in ein PWM-Signal umgerechnet. Dieses PWM-Signal wird zur Geschwindigkeitsregelung des Motors an den ENA Anschluss des L298N Motortreiber Moduls ausgegeben. Da das Auto vorwärtsfahren soll, wird IN1 „HIGH“ und IN2 „LOW“ gesetzt.

```
void stufenlos_vor()
{
  v_stufenlos= map (slider1,9,20,130,255);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  analogWrite(turbo,v_stufenlos);
  freigabe=false;
}
```

→für Ansteuerung des Motors über das L298N Modul siehe Seite 24 ff.

Da die empfangenen Steuerwerte nicht gerundet sind und jeder Steuerwert des Slider 1 eine neue Geschwindigkeit hervorruft, ist die Geschwindigkeit beim Vorwärts- und Rückwärtsfahren stufenlos regelbar. Die Geschwindigkeit verhält sich hier also proportional zu der Position des Slider 1.

2.1.3.4 Nicht stufenlose Geschwindigkeitsregelung / Gangsteuerung

Ist die Aktionsnummer des eingelesenen Bluetooth Codes „2“, so werden die Werte des Slider 2 eingelesen und in der Variable „slider2“ gespeichert. Da diese von meiner Smartphone App auf ganze Zahlen von 1-6 gerundet wurden, können sechs verschiedene Steuerwerte des Slider 2 vorkommen. Diese sechs Werte von 1-6 entsprechen sechs Gängen, vom ersten bis zum sechsten Gang. Durch einen „switch-Befehl“ wird je nach eintreffendem Steuerwert ein dazugehöriges Unterprogramm aufgerufen. Dies ist übersichtlicher als viele „if-Befehle“ einzusetzen.

```
if(aktionsnummer == 2 && freigabe == true)
{
  slider2= steuerwerte ;

  switch(slider2){
  case 1:
    gang_s_back();
    break;
  case 2:
    gang_l_back();
    break;
  case 3:
    gang_stopp();
    break;
  case 4:
    gang_l_vor();
    break;
  case 5:
    gang_m_vor();
    break;
  case 6:
    gang_s_vor();
    break;
  }
}
```

Im beigelegten Gesamtprogramm werden durch Kommentare die einzelnen Programmzeilen genauer erklärt. Diese Kommentare sind in der Dokumentation aus Platzgründen zum Teil weggelassen worden.

In der darauf folgenden Tabelle erkennt man, welche Funktion bzw. welches Unterprogramm bei welchem Gang bzw. Steuerwert, ausgeführt wird

Sliderwert / Steuerwert	Gang	ausgeführte Funktion	Name des Unterprogramms
1	1	schnell zurückfahren	gang_s_back()
2	2	langsam zurück fahren	gang_l_back()
3	3	keine Bewegung	gang_stopp()
4	4	langsam vorwärts fahren	gang_l_vor()
5	5	mit mittlerer Geschwindigkeit vorwärts fahren	gang_m_vor()
6	6	schnell vorwärts fahren	gang_s_vor()

Die einzelnen Unterprogramme der Gangsteuerung, welche durch die Steuerwerte aufgerufen werden, sind immer gleich aufgebaut. In ihnen werden die Ausgänge IN1 und IN2 des Motortreiber Moduls „HIGH“ oder „LOW“ gesetzt um die gewünschte Drehrichtung des Motors zu erhalten.
Des Weiteren wird in jedem Unterprogramm ein PWM-Signal per „analogWrite“ in den „turbo“ Ausgang des Funduino geschrieben. Da dieser Ausgang mit dem ENA Anschluss des L298N Moduls verbunden ist, steuert dieses PWM-Signal in Form der Variable „geschwindigkeit“ die Geschwindigkeit des Arduino Car.

Die hervorgerufene Funktion durch das Ausgeben von Signalen an „IN1“, „IN2“ und „turbo“ ist in der obigen Tabelle in der „ausgeführte Funktion –Spalte“ erkennbar.

In jedem Unterprogramm, für jeden Gang ist auch immer eine „for-Schleife“ enthalten. Diese dient dazu, die „geschwindigkeit-Variable“ bei Wechseln des Ganges in kleine Schritte in die Sollgeschwindigkeit des neuen Ganges zu ändern. Dadurch soll eine abrupte Geschwindigkeitsänderung verhindert werden, wenn der Gang geändert wird.
Der Funduino wiederholt so lange die „for-Schleife“ und senkt oder erhöht die „geschwindigkeit-Variable“ bis sie der Sollgeschwindigkeit des Ganges entspricht. Da die „for-Schleife“ vom Funduino sehr schnell wiederholt wird, ergibt sich nur eine sehr geringe, fast unerkennbare Zeitverzögerung im Programm. Dafür wird jedoch die Geschwindigkeitszu- bzw. Abnahme des Motors fließender durchgeführt.

Ein Beispiel für ein Unterprogramm der Gangsteuerung:

gang_l_back():

```
void gang_l_back()
{
  for ( int geschwindigkeit=0 ; geschwindigkeit <= 130; geschwindigkeit +=10)
  {
    digitalWrite(IN1,LOW);
    digitalWrite(IN2,HIGH);
    analogWrite(turbo, geschwindigkeit);
  }
  for ( int geschwindigkeit=255 ; geschwindigkeit >=130 ; geschwindigkeit -=10)
  {
    digitalWrite(IN1,LOW);
    digitalWrite(IN2,HIGH);
    analogWrite(turbo, geschwindigkeit);
  }
}
```

2.3.1.5 Dominanz der stufenlosen Geschwindigkeitsregelung / des Slider 1

In meiner Smartphone App sind zwei verschiedene Slider eingebaut, die jeweils die Geschwindigkeits- und Richtungssteuerung des Motors regeln, es muss ein Slider bzw. ein Bluetooth Code zur Motorsteuerung dominieren. Sonst könnten von beiden Slidern gleichzeitig verschiedene Bluetooth Codes bearbeitet werden, die unterschiedliche Funktionen hervorrufen. Dies wäre eine große Fehlerquelle im Programm.
Durch die Dominanz einer der beiden Slider wird festgelegt, welcher Slider die Motorsteuerung bestimmt, wenn beide Slider einen Bluetooth Code senden.
In meiner Steuerung ist der dominante Slider immer Slider 1 und dadurch hat die stufenlose Geschwindigkeitsregelung Vorrang.

Diese Dominanz des Slider 1 wird durch eine Variable „freigabe“ erreicht. Diese „boolean Variable“ kann nur „true“ oder „false“ sein. Immer wenn der Slider 1 einen Steuerwert sendet, durch welchen das Unterprogramm zur Vor- oder Rückwärtsbewegung aufgerufen wird, wird diese Variable auf „false“ gesetzt.
Dadurch, dass die Freigabevariable in meinem Programm jedoch auch in der „if-Bedingung“ für das Bearbeiten der Steuerwerte des Slider 2 enthalten ist, wird der Slider 2 vom Funduino nur wahrgenommen, wenn die Aktionsnummer des Blutooth Codes „2“ ist und zusätzlich die Variable „freigabe“ auf „true“ steht. Somit bearbeitet der Funduino im Falle von Signalen von beiden Slidern immer die Werte des Slider 1.

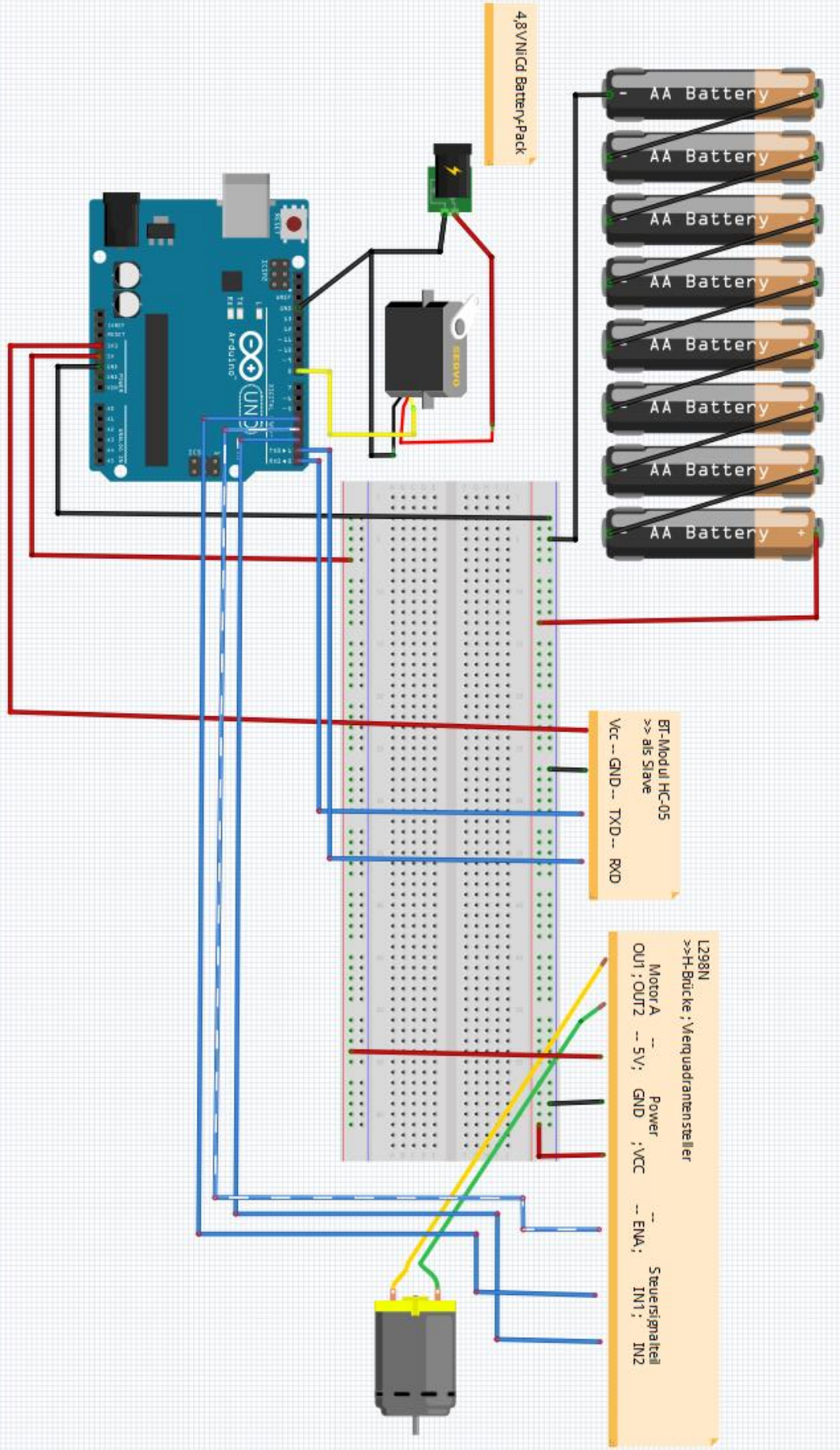
```
if(aktionsnummer == 2 && freigabe == true)
{
  slider2= steuerwerte ;
```

Eine Ausnahme gibt es nur, wenn Slider 1 im Freigabebereich steht (siehe Seite 16). Dies wird in der App durch einen Hinweistext verdeutlicht. Jetzt wird die Variable „freigabe“ auf „true“ gesetzt. Wenn nun Bluetooth Codes von Slider 2 eingehen, werden diese bearbeitet und der Slider 2 bzw. die Gangsteuerung bestimmt die Motorsteuerung.
Dies hält jedoch nur so lange an, wie der Slider 1 im Freigabebereich steht und dadurch Slider 2 freigegeben wird.

```
void stufenlos_freigabe()                    // Unterprogramm Freigabe für Slider 2
{
  digitalWrite(IN1,LOW);                     //     |
  digitalWrite(IN2,LOW);                     //     | Motoransteuerung durch Slider 1 aus
  analogWrite(turbo,LOW);                    //     |
  freigabe=true;                             // Slider 2 "einschalten"; Freigabe für Gangsteuerung
}
```

Durch diese Dominanz des Slider 1 und dadurch der stufenlosen Geschwindigkeitssteuerung gegenüber der Gangsteuerung, wird die Motorsteuerung immer eindeutig von einem Steuerelement der App beeinflusst und Fehlfunktionen werden vermieden.

Anschlussplan für Lenkung und Steuerung:



2.3.2 Die Signal LED

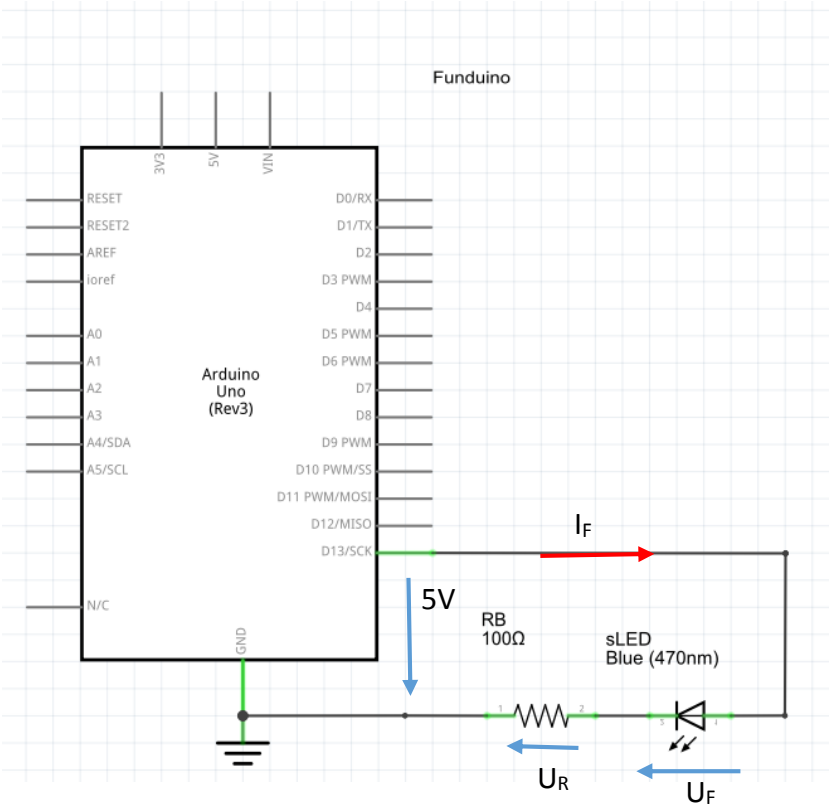
Verwendete Bauteile:

- 1 x blaue LED
- 1 x 100 Ω Vorwiderstand

Funktion:

Die Signal LED ist in der Karosserie des Arduino Car eingebaut und soll blinken, wenn eine Bluetooth Verbindung besteht.

Schaltplan



Berechnung des Vorwiderstandes R_V:

Für LEDs müssen immer Vorwiderstände verwendet werden, welche dafür sorgen, dass an der LED nur die zulässige Spannung anliegt. Durch diese Vorwiderstände wird auch der Strom durch die LEDs begrenzt, sodass die Stromstärke nicht über die maximal zulässige Stromstärke $I_{F\,max}$ der LED ansteigt. Für $I_{F\,max}$ nehme ich in meinen Berechnungen immer den Wert 20 mA an.

Kennwerte der blauen LED:

$U_{F\,blau}$= 3,1 V

$I_{F\,max}$ = 20 mA

$U_{b\,Fun}$ = 5 V *5

*5 $U_{b\,Fun}$ = 5V Betriebsspannung von Funduino Pins

$$R_V = \frac{U_{b\,Fun} - U_{F\,blau}}{I_{F\,max}} = \frac{5V - 3{,}1V}{20mA} = 95\Omega$$

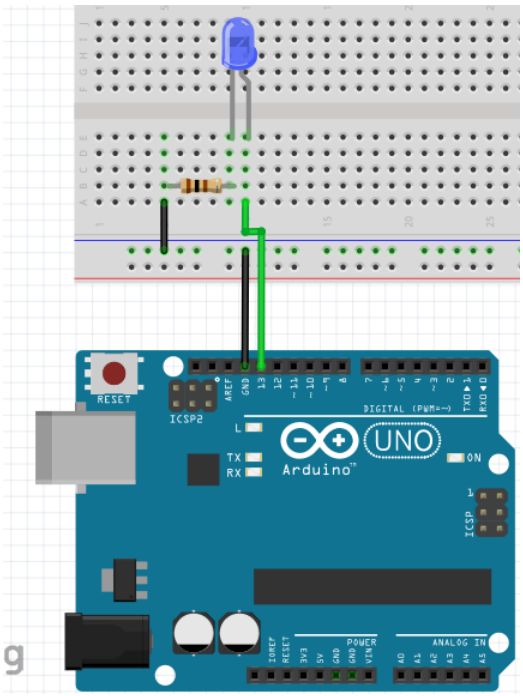
Damit die maximal zulässige Stromstärke nicht überschritten wird, muss in der Praxis für den Vorwiderstand der nächst größere Wert aus der E12-Widerstandsreihe gewählt werden:

$$\boldsymbol{R_V = 100\Omega}$$

Mit dem gewählten Vorwiderstand R_V ergibt sich ein Strom durch die Signal LED wie folgt:

$$I_F = \frac{U_{b\,Fun} - U_{F\,blau}}{R_V} = \frac{5V - 3{,}1V}{100\Omega} = 19mA$$

Anschlussplan:



Programmierung:

Allgemeines Problem im Programm:

Die Signal LED soll immer in einem bestimmten Intervall auf und ab leuchten. Dies muss ohne delay ()-Befehle programmiert werden, damit es zu keinem Anhalten des Programmdurchlaufs kommt. Die delay()-Funktionen müssen nicht nur hier, sondern im ganzen Programm vermieden werden, da durch sie das Programm gestoppt wird. Dies hätte zur Folge, dass das Arduino Car kurzzeitig nicht steuerbar ist.

Um dies zu vermeiden, wird in meinem Programm die Zeit in Millisekunden ab Einschalten des Funduino in der Variable „zeit“ gespeichert. Diese Zeit wird durch die millis()-Funktion der Arduino Software ermittelt.

Diese „zeit-Variable“ ermöglicht es, festgelegte zeitliche Abläufe im Programm zu regeln ohne delay()-Befehle zu benutzen. Dies wird durch „Wecker“ geregelt, welche als Variablen angegeben sind und mit der durchlaufenden Variable „zeit“ verglichen werden. Entspricht die „wecker-Variable“ der Variable „zeit“, wird eine Aktion ausgeführt und der „Wecker“ wird durch Addition eines Zahlenwertes neu eingestellt.

signalLED():

Das Unterprogramm signalLED() wird aufgerufen, wenn der Startbefehl „S“ des Bluetooth Codes erkannt wird. Dadurch lässt sich kontrollieren, ob eine Kommunikation zwischen Arduino Car und Smartphone App besteht.

```
if(Serial.find(STARTBEFEHL)){     // für BT-Steuerung "S" des BT Codes notwendig

  signalLED();
```

Die Signal LED wird ebenso durch einen Vergleich der „zeit“ Variable mit der „wechsel_sLED - Variable“ zum Blinken gebracht. In der zweiten Variablen wird die Zeit festgehalten, als sich der Status (HIGH, LOW) der LED das letzte Mal geändert hat. Ist eine festgelegte Zeit (hier: 500ms) seit dem letzten Statuswechsel verstrichen, wird der Zustand bzw. der Status der Signal LED umgekehrt und die „wechsel_sLED -Variable“ erneuert. Dadurch ist die LED immer 500ms an und 500ms aus: sie blinkt!!

```
void signalLED()
{
  if (( zeit - wechsel_sLED) > 500){
    wechsel_sLED = zeit ;

    if (sLED_stat == LOW){
      sLED_stat= HIGH;
    }
    else{
      sLED_stat = LOW;
    }
    digitalWrite(sLED, sLED_stat);
  }
}
```

2.3.3 Das Rücklicht

Verwendete Bauteile:

- 1 x NiMH 9V Block Akku
- 2 x BC 237c NPN Transistor
- 6 x rote LED
- 2 x 5,6 kΩ Basisvorwiderstand
- 2 x 150Ω Vorwiderstand

Funktion:

Im Heck der Autokarosserie befinden sich zwei Reihenschaltungen mit je drei LEDs. Sie bilden ein rechtes und ein linkes Rücklicht. Diese Rücklichter sollen bei bestimmten Bewegungen, wie z.B. Rechtslenkung, Linkslenkung und Rückwärtsfahrt eine bestimmte Blinkfunktion ausführen.

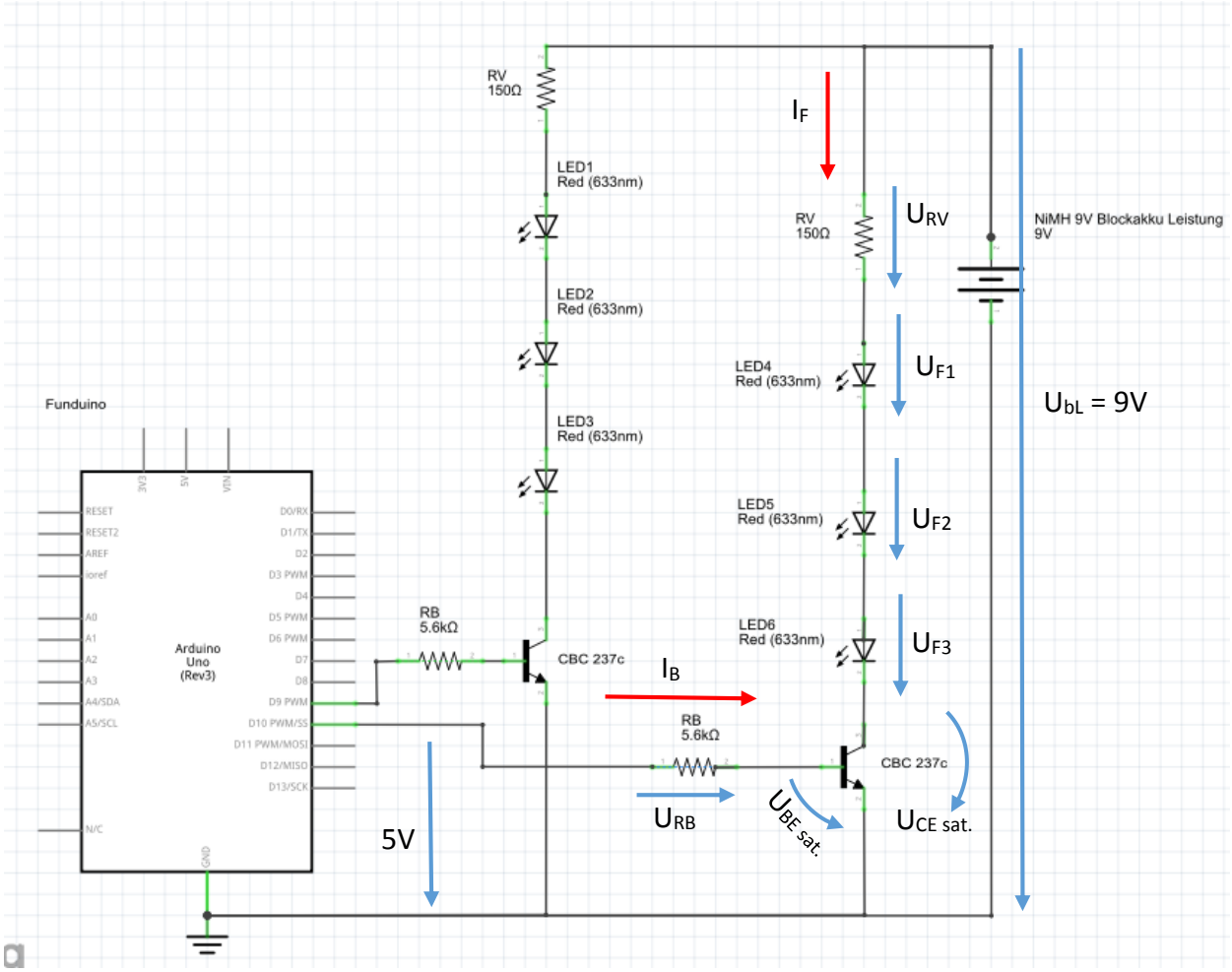
Ein Transistor als Schalter für große Leistungen:

In allen Transistorschaltungen meines Projekts, werden die Transistoren zum Schalten einer größeren Leistung verwendet:
Die Basis der Transistoren ist immer mit einem Funduino Pin verbunden. Wird dieser Pin „HIGH“ gesetzt, fließt der Strom I_B, welcher durch den Basisvorwiderstand R_B begrenzt wird. Dadurch wird der Transistor übersteuert und durch die CE-Strecke kann der Strom I_C durch den Transistor fließen. Grund dafür ist, dass ein Transistor bei Übersteuerung nur einen kleinen Widerstand zwischen Kollektor C und Emitter E aufweist.
Durch die gewählte Beschaltung können zwei Zustände auftreten:

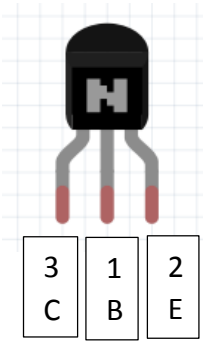
Zustand	Basisstrom I_B / Steuersignal	Transistor	I_C
1	$I_B = 0$; Pin == LOW	nicht übersteuert, großer Widerstand zwischen C-E	$I_C = 0$; es kann kein Strom im Leistungsteil fließen
2	I_B = gering, aber ausreichend ; Pin == HIGH	wird übersteuert; ist gesättigt; kleiner Widerstand zwischen C-E	$I_C > 0$; es kann ein Strom im Leistungsteil fließen

Der Transistor verhält sich wie ein Schalter.
Durch ihn kann man den Leistungsteil mit der Betriebsspannung U_b= 9V durch kleine Ströme an der Basis, mit dem Funduino steuern.
Dies ist notwendig, da die LED-Reihenschaltungen oder Motoren eine höhere Betriebsspannung benötigen, als sie ein Funduino Pin ausgeben kann. Man kann die Verbraucher, welche eine Spannung über 5V und eine Stromstärke über 36 mA benötigen, nicht direkt durch einen Funduino Pin steuern.

Schaltplan:



Berechnung der Rücklichtschaltung



Kennwerte des BC 237c NPN Transistor

Verstärkungsfaktor B = 150

Übersteuerungsfaktor ü = 5

$U_{CE\ sat}$ = 0,2 V

$U_{BE\ sat}$ = 0,6 V

1: Basis B
2: Emitter E
3: Kollektor C

Kennwerte einer roten LED

U_F = 2,1 V

$I_{F\ max}$ = 20 mA

Berechnung des Vorwiderstandes der LED-Reichenschaltung:

$$R_V = \frac{U_{bL} - U_{F1} - U_{F2} - U_{F3} - U_{CE\,sat}}{I_{F\,max}} = \frac{9V - 2{,}1V - 2{,}1V - 2{,}1V - 0{,}2V}{20mA} = 125\Omega$$

Hier wähle ich aus der E12-Reihe den Widerstand 150Ω, da man bei der Berechnung des Vorwiderstandes für LEDs immer einen größeren Widerstand wählt. Dadurch wird die angenommene maximale Stromstärke $I_{F\,max}$ nicht überschritten.

$$\boldsymbol{R_V = 150\Omega}$$

Durch den gewählten Vorwiderstand ergibt sich ein Strom durch die Reihenschaltung wie folgt:

$$I_F = \frac{U_{bL} - U_{F1} - U_{F2} - U_{F3} - U_{CE\,sat}}{R_V} = \frac{9V - 2{,}1V - 2{,}1V - 2{,}1V - 0{,}2V}{150\Omega} = 16{,}67mA$$

Berechnung des Basisvorwiderstandes des Transistors:

$$I_B = \frac{I_F}{B} \cdot \text{ü} = \frac{16{,}67mA}{150} \cdot 5 = 0{,}56mA$$

$$R_B = \frac{U_{b\,Fun} - U_{BE\,sat}}{I_B} = \frac{5V - 0{,}6V}{0{,}56mA} = 7{,}86k\Omega$$

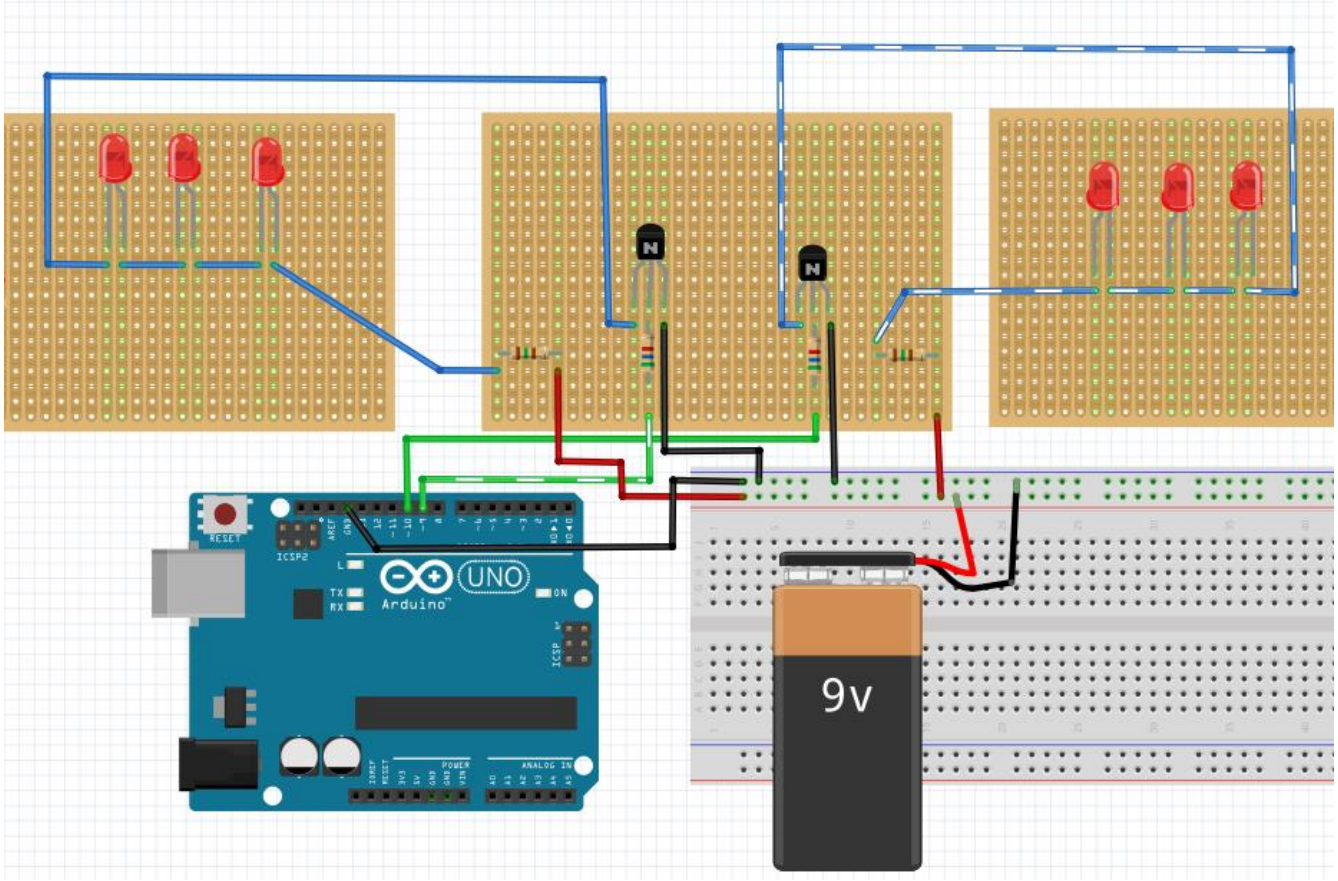
Damit der Basisstrom I_B immer groß genug ist um den Transistor zu übersteuern, wird ein kleinerer Widerstandswert aus der E12-Widerstandsreihe gewählt, welcher mir zur Verfügung steht:

$$\boldsymbol{R_B = 5{,}6k\Omega}$$

Mit dem gewählten Vorwiderstand R_B ergibt sich der Basisstrom I_B wie folgt:

$$I_B = \frac{U_{b\,Fun} - U_{BE\,sat}}{R_B} = \frac{5V - 0{,}6V}{5{,}6k\Omega} = 0{,}79mA$$

Anschlussplan:



Programmierung:

Die Rücklichter sollen sich je nach Aktion mit folgender Vorgabe blinken:

Aktion		Blinkmuster	Unterprogramm
Vorwärts-bewegung	Linkslenkung	rechtes Blinklicht schnell blinken	ruecklicht_links_blinken()
	Rechtslenkung	linkes Blinklicht schnell blinken	ruecklicht_rechts_blinken()
keine Bewegung		beide Rücklichter an	----------
Rückwärtsfahrt		beide Rücklichter blinken langsam und gleichzeitig	rueckfahrblinken()

Das Problem ist, dass jede oben genannte Aktion durch die Lenkungswerte und die Steuerwerte des Slider 1 ODER Slider 2 hervorgerufen wird. Man muss also in den „if-Befehlen“ jeweils mehrere mögliche Zustände der Steuerelemente für eine Aktion eingeben.

Dadurch sind die Blinkmuster bei der stufenlosen Geschwindigkeitssteuerung und der Gangsteuerung gleich.
Ein Beispiel hierfür:

```
else if (slider1>=8 && slider1<= 10 && slider2 == 3)     // Bedingung: Auto steht / fährt nicht
{
  digitalWrite(rlicht_l, HIGH);
  digitalWrite(rlicht_r,HIGH);
}

else if (slider1 >=1 && slider1 <8 || slider1>=8 && slider1<= 10 && slider2< 3  ) // Bedingung: Auto fährt zurück
{
  rueckfahrblinken();
}
```

Der jeweilige Wechsel von „An“ und „Aus“ der Rücklichter wird nach der gleichen Vorgehensweise wie bei der Signal LED (Seite 39), ohne delay()-Funktionen erreicht.

Als Beispiel ist der Programmteil ruecklicht_links_blinken() abgebildet:

```
void ruecklicht_rechts_blinken()      // Unterprogramm für Blinken des rechten Rücklichts bei Rechtslenkung
{
  if ((zeit - rlr_wechsel)  > 250)    // prüfen ob die Blinkdauer (250) vom rechten Rücklicht schon verstrichen ist
  {
    rlr_wechsel = zeit ;              // aktuelle millis()-Zeit gespeichert=Zeitpunkt des Statuswechsel von rechtem Rücklicht

      if (rlr_stat == LOW){           // Statuswert des rechten Rücklichts umkehren
      rlr_stat = HIGH;
    }
    else{
      rlr_stat = LOW;
    }
    digitalWrite(rlicht_r, rlr_stat);   // den neuen Statuswert in rechtes Rücklicht schreiben --> Blinken
    digitalWrite(rlicht_l, LOW);        // Verriegelung linkes Rücklicht
  }
}
```

2.3.4 Der Scheinwerfer

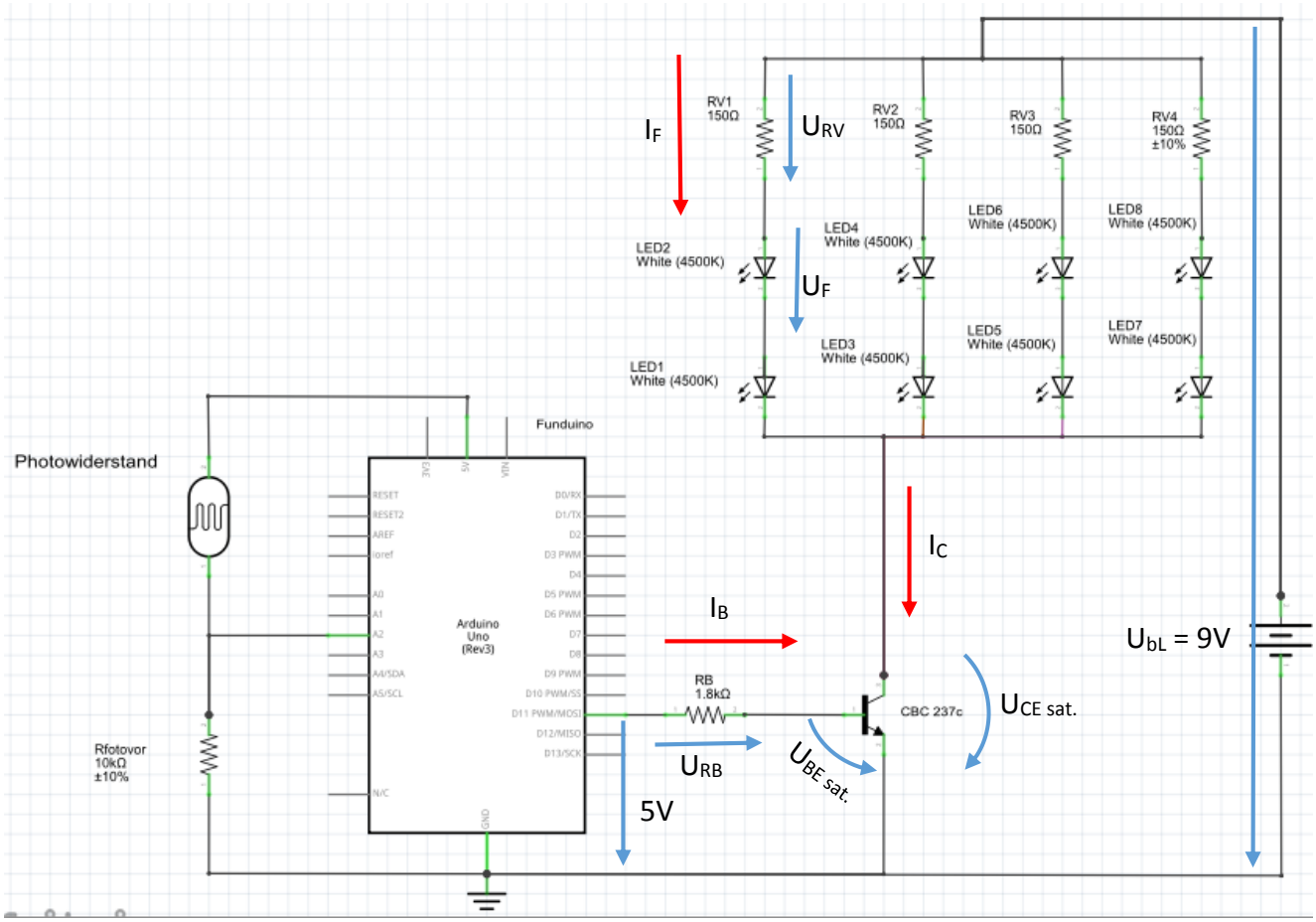
Verwendete Bauteile:

- 1 x NiMH 9V Block Akku
- 1 x BC 237c NPN Transistor
- 1 x 1,8$k\Omega$ Basisvorwiderstand
- 4 x 150Ω Vorwiderstand
- 6 x weiße LED
- 1 x Fotowiderstand
- 1x 10$k\Omega$ Fotovorwiderstand

Funktion:

Die LEDs werden in eine gemischte Schaltung zu einem großen Scheinwerfer zusammengesetzt. Der Scheinwerfer soll leuchten und seine Leuchtstärke an die Umgebungshelligkeit anpassen. Wenn es hell ist, soll dies der Funduino durch den Fotowiderstand erkennen und darauf den Scheinwerfer schwächer leuchten lassen und umgekehrt.

Schaltplan:



Berechnung der Scheinwerferschaltung:

Kennwerte BC 237c NPN Transistor

Siehe Berechnung der Rücklichtschaltung Seite 41.

Kennwerte einer weißen LED:

U_F = 3,3 V

$I_{F\,max}$ = 20 mA

Berechnung des Vorwiderstandes der LED Reihenschaltungen:

$$R_V = \frac{U_{bL} - U_{F1} - U_{F2} - U_{CE\,sat}}{I_{F\,max}} = \frac{9V - 3{,}3V - 3{,}3V - 0{,}2V}{20mA} = 110\Omega$$

Aus der E-12-Widerstansreiehe wird ein größerer Wert für den Vorwiderstand R_V gewählt:

$$\boldsymbol{R_V = 150\Omega}$$

Mit dem gewählten Vorwiderstand R_V ergibt sich der Strom I_F wie folgt:

$$I_F = \frac{U_{bL} - U_{F1} - U_{F2} - U_{CE\,sat}}{R_V} = \frac{9V - 3{,}3V - 3{,}3V - 0{,}2V}{150\Omega} = 14{,}67mA$$

Berechnung des Basisvorwiderstandes R_B des Transistors:

$$I_C = 4 \cdot I_F = 4 \cdot 14{,}67mA = 58{,}68mA$$

$$I_B = \frac{I_C}{B} \cdot \ddot{u} = \frac{58{,}68mA}{150} \cdot 5 = 1{,}96mA$$

$$R_B = \frac{U_{b\,Fun} - U_{BE\,sat}}{I_B} = \frac{5V - 0{,}6V}{1{,}96mA} = 2{,}24k\Omega$$

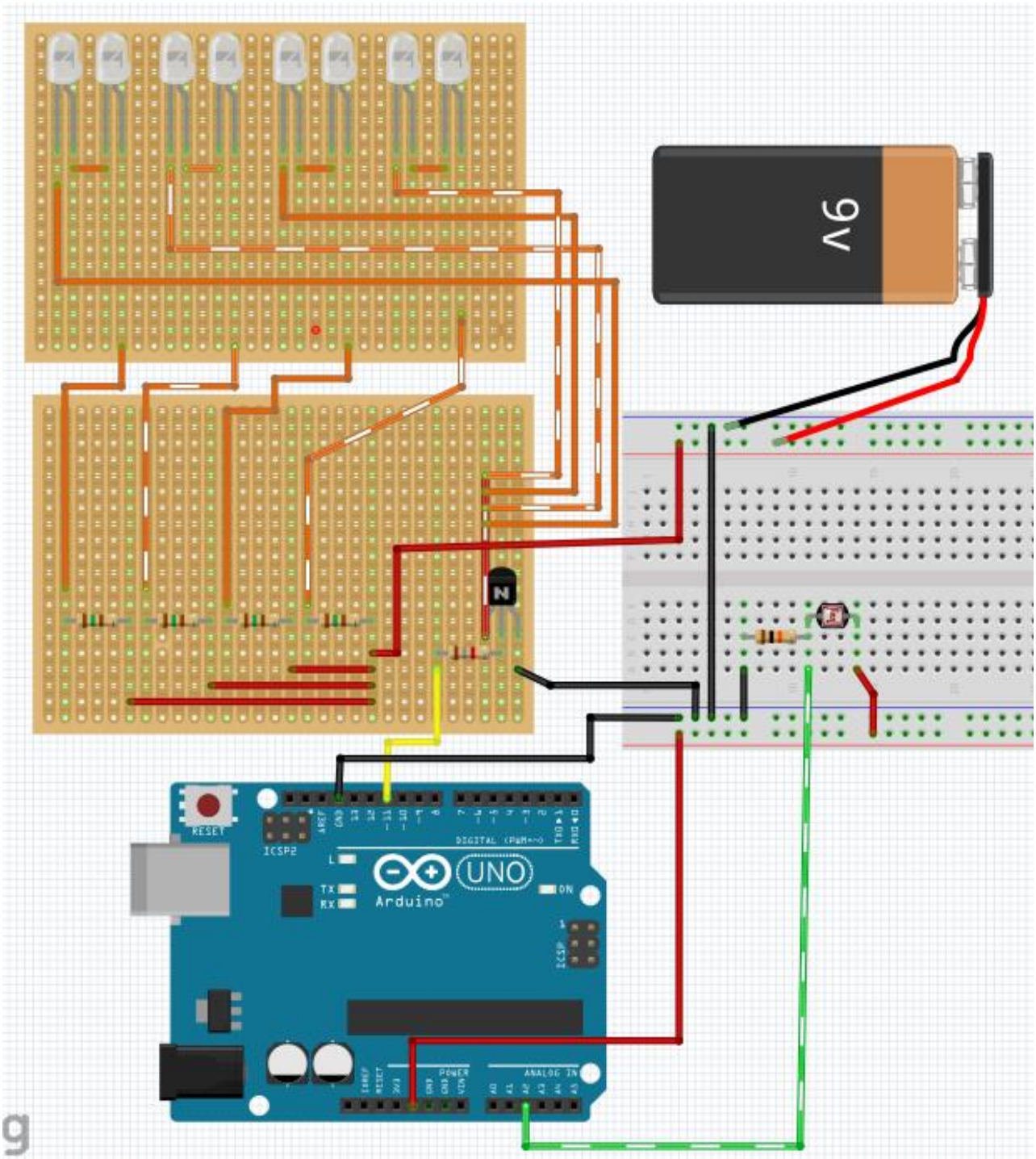
Hier wähle ich einen kleineren Wert aus der E-12 Widerstandsreihe, damit I_B den Transistor übersteuert:

$$\boldsymbol{R_B = 1{,}8k\Omega}$$

Durch den gewählten Basisvorwiderstand R_B ergibt sich der Basisstrom I_B wie folgt:

$$I_B = \frac{U_{b\,Fun} - U_{BE\,sat}}{R_B} = \frac{5V - 0{,}6V}{1{,}8k\Omega} = 2{,}44mA$$

Anschlussplan:



Programmierung:

Der Programmteil des Scheinwerfers befindet sich in der großen `if(Serial.find(STARTBEFEHL)){` „if-Schleife“. Dadurch wird das Scheinwerferprogramm scheinwerfersteuerung() nur bei bestehender Bluetooth Verbindung mit der App ausgeführt.

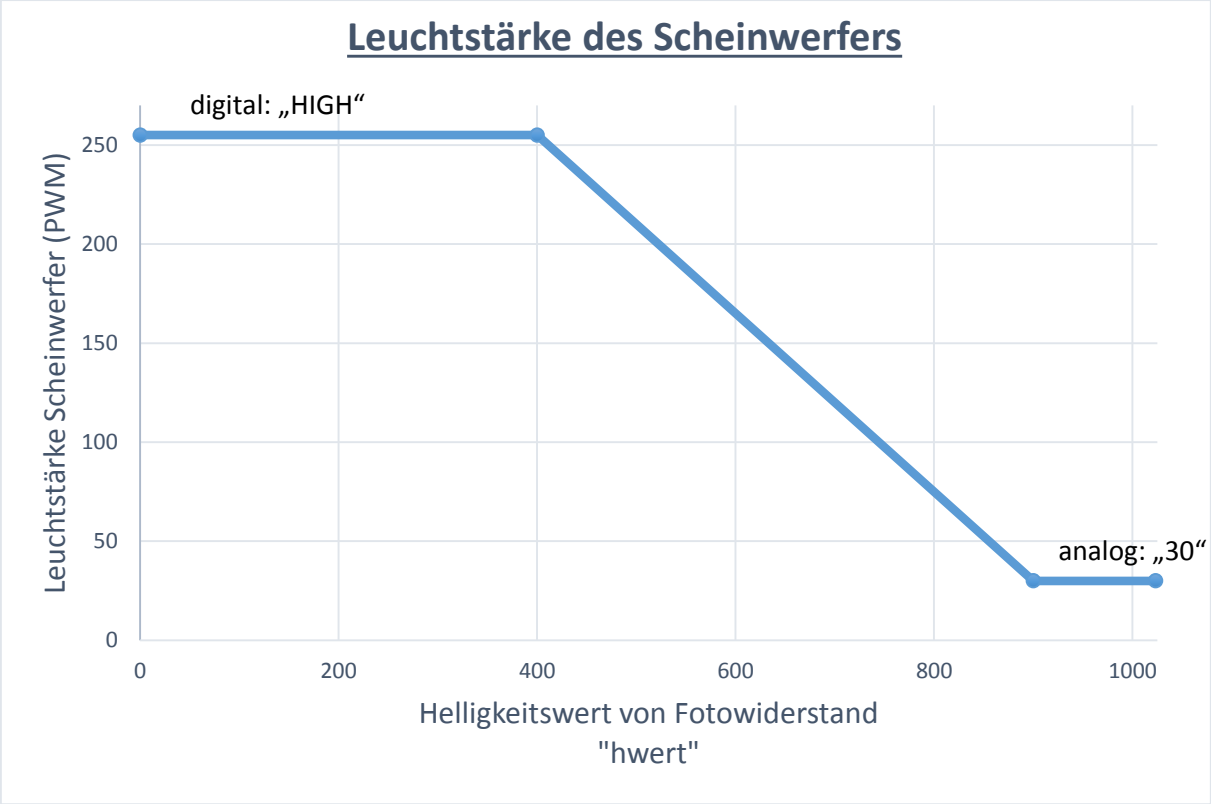
scheinwerfersteuerung():

Zuerst wird der Wert des Fotowiderstandes eingelesen und in der Variable „hwert“ gespeichert.

Ist dieser Wert ≥ 900, soll der Scheinwerfer gleichbleibend schwach, durch das PWM-Signal 30, leuchten.

Ist der „hwert“ ≤ 400, soll der Scheinwerfer mit der maximalen Leuchtkraft scheinen.
→ digital.Write(scheinwerfer,HIGH)

Befindet sich der „hwert“ zwischen 400 und 900, also zwischen dem „Hell-“und „Dunkelbereich“, wird die Scheinwerferhelligkeit stufenlos angepasst. Durch Transformation des „hwert“ in den Wertebereich [254; 31] mithilfe der „map-Funktion“ wird ein PWM-Signal für den „scheinwerfer-Pin“ erstellt. Die Leuchtstärke des Scheinwerfers, bzw . die Höhe des PWM-Signals verhält sich hier umgekehrt proportional zu der Umgebungshelligkeit.



Da der Programmteil des Scheinwerfers keine delay(), beinhaltet, dauert der Durchlauf dieses Programmteils nur eine sehr kurze Zeit. Deswegen kann es bei jedem Durchlauf bzw. Zyklus des Funduino bearbeitet werden.

```
void scheinwerfersteuerung()
{
  hwert =analogRead(hsensor);
  if (hwert <= dunkel){
    digitalWrite(scheinwerfer, HIGH);
  }
  else if (hwert > dunkel && hwert < hell){
    pwmwert = map(analogRead(hsensor), 401,899,254,31);
    analogWrite(scheinwerfer,pwmwert);
  }
  else if (hwert >= hell && hwert <= 1023){
    analogWrite(scheinwerfer, 30);
  }
```

<u>Der Scheinwerfer:</u>

2.3.5 Die Kühlung

Verwendete Bauteile:

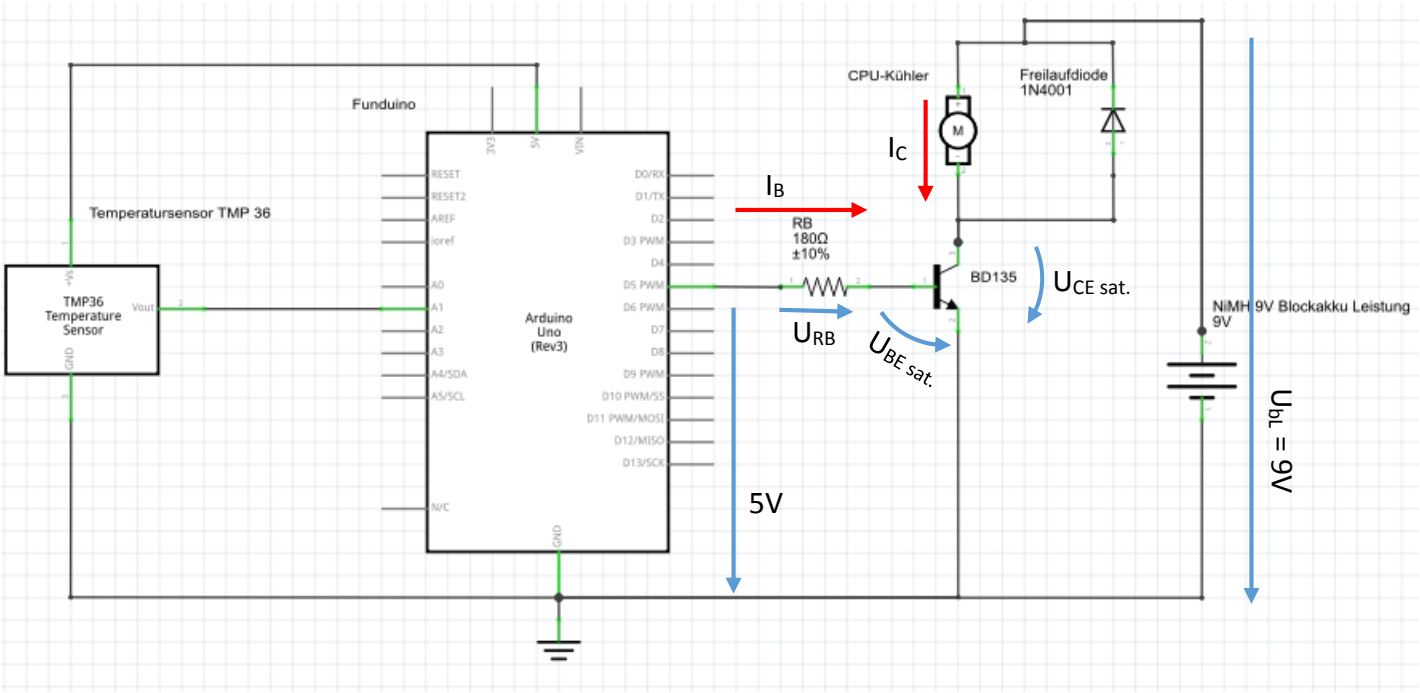
- CPU-Kühler bzw. Lüfter
  Der CPU-Kühler hat vier Pins von den ich nur den Schwarzen (=GND) und den Gelben (=VCC) benötige
- NiMH 9V Block Akku
- TMP 36-Temperatursensor
- BD 135 NPN Transistor
- Freilaufdiode 1N4001

Funktion:

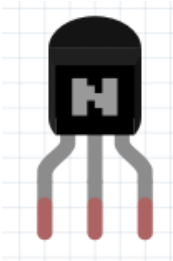
Der Funduino kann aus den Werten des Temperatursensors die Temperatur berechnen. Ist diese Temperatur im Fahrzeuginnenraum höher als eine festgelegte Temperaturgrenze, so soll der CPU-Kühler einschalten.

Schaltplan:

Achtung: Parallel zum Lüfter Motor muss hier eine Diode geschalten werden. Sie dient zum Schutz des Transistors bei nachlaufendem Motor, denn dieser erzeugt nach Abschalten/Kurzschluss der Spannung eine Spannungsspitze, welche den Transistor zerstören würde. Durch die Freilaufdiode wird nach Abschalten der Spulenstromkreis über die Diode geschlossen, und dadurch der Transistor vor Überspannung geschützt.



Berechnung der Lüfter Schaltung:



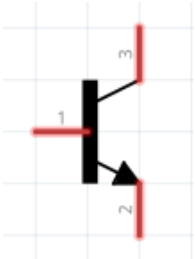
Kennwerte des BD 135 NPN Transistor

B = 40

ü = 5

$U_{CE\ sat}$ = 0,5 V

$U_{BE\ sat}$ = 1 V



1: Basis B
2: Emitter E
3: Kollektor C

2	3	1
E	C	B

Berechnung des Innenwiderstands des Lüfter Motors:

$$R_{Motor} = \frac{12V}{230mA} = 52{,}2\Omega$$

Berechnung des Basisvorwiderstands R_B:

$$I_C = \frac{U_{bL} - U_{CE\ sat}}{R_{Motor}} = \frac{9V - 0{,}5V}{52{,}2\Omega} = 162{,}8mA$$

$$I_B = \frac{I_C}{B} \cdot \text{ü} = \frac{162{,}8mA}{40} \cdot 5 = 20{,}4mA$$

$$R_B = \frac{U_{b\ Fun} - U_{BE\ sat}}{I_B} = \frac{5V - 1V}{20{,}4mA} = 196{,}1\Omega$$

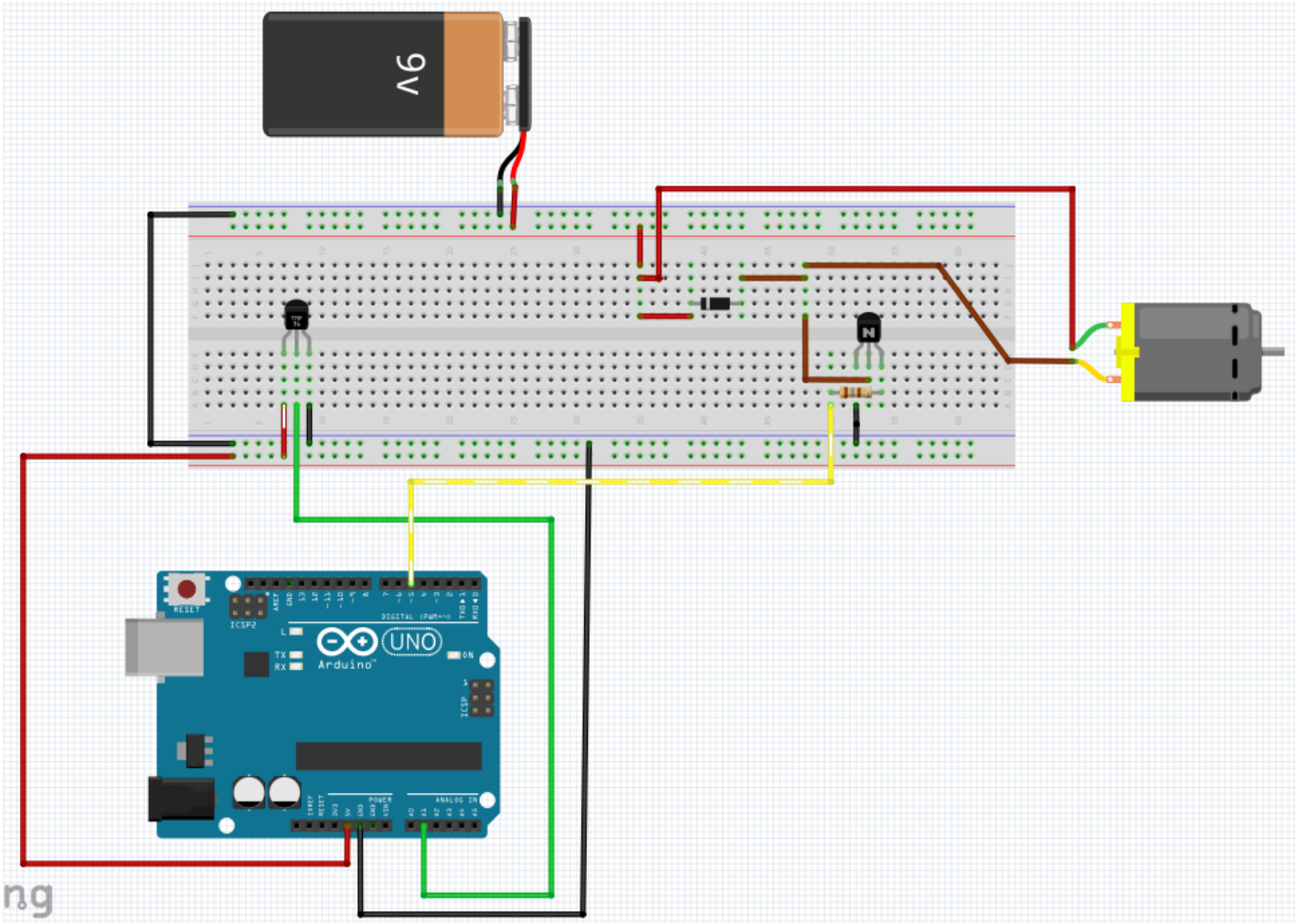
Als Basisvorwiderstand R_B wähle ich den nächst kleineren Wert aus der E12-Widerstandsreihe, damit der Transistor durch den Basisstrom I_B auf jeden Fall übersteuert wird.

$$\boldsymbol{R_B = 180\Omega}$$

Mit dem gewählten Basisvorwiderstand R_B ergibt sich der Basisstrom I_B wie folgt:

$$I_B = \frac{U_{b\ Fun} - U_{BE\ sat}}{R_B} = \frac{5V - 1V}{180\Omega} = 22{,}22mA$$

Anschlussplan:



Programmierung:

Der Programmteil des CPU-Kühlers befindet sich als einziger Programmteil außerhalb der großen `if(Serial.find(STARTBEFEHL)){` „if-Schleife“. Dadurch ist er unabhängig von der Bluetooth Kommunikation und das Unterprogramm „tempmessen()“ wird immer bearbeitet, wenn der Funduino eingeschalten ist.

```
if (wecker_temp == zeit )          //  |
{                                  //  | Temperatur nur zu bestimmten Zeiten gemessen. NICHT immer!
  wecker_temp = (zeit + 4000);     //  |
  tempmessen();                    //  |
}                                  //  |
```

tempmessen() :

Das Unterprogramm zur Temperaturermittlung soll nur alle 4s erfolgen. Da hier wieder keine delay()-Pausen verwendet werden dürfen, wird dieser zeitliche Ablauf durch Vergleiche eines Weckers mit der durchlaufenden „zeit-Variable“ geregelt. Der Wecker wird in der Variable „wecker_temp“ angegeben. Entspricht die „wecker_temp-Variable“ der Variable „zeit“, werden die Werte des TMP 36 Sensors eingelesen, bearbeitet und der „wecker_temp“ wird durch Addition mit der Zahl 4000 neu eingestellt.

Die eingelesenen Werte des Temperatursensors werden durch eine Transformation von dem Wertebereich [0; 410] in den Wertebereich [-50; 150] in eine °C Temperatur umgewandelt.

Übersteigt die Temperatur die Temperaturgrenze von 15°C, wird der Lüfter/Ventilator eingeschalten.

```
void tempmessen ()
{
  int temperatur = map(analogRead(TMP36), 0, 410, -50, 150);
  if (temperatur >= 15){
    digitalWrite (ventilator,HIGH);
  }
  else {
    digitalWrite (ventilator, LOW);
  }
```

2.3.6 Die Einparkhilfe

Verwendete Bauteile:

- 1 x Ultraschallsensor HC–SR04
- Piezo Speaker

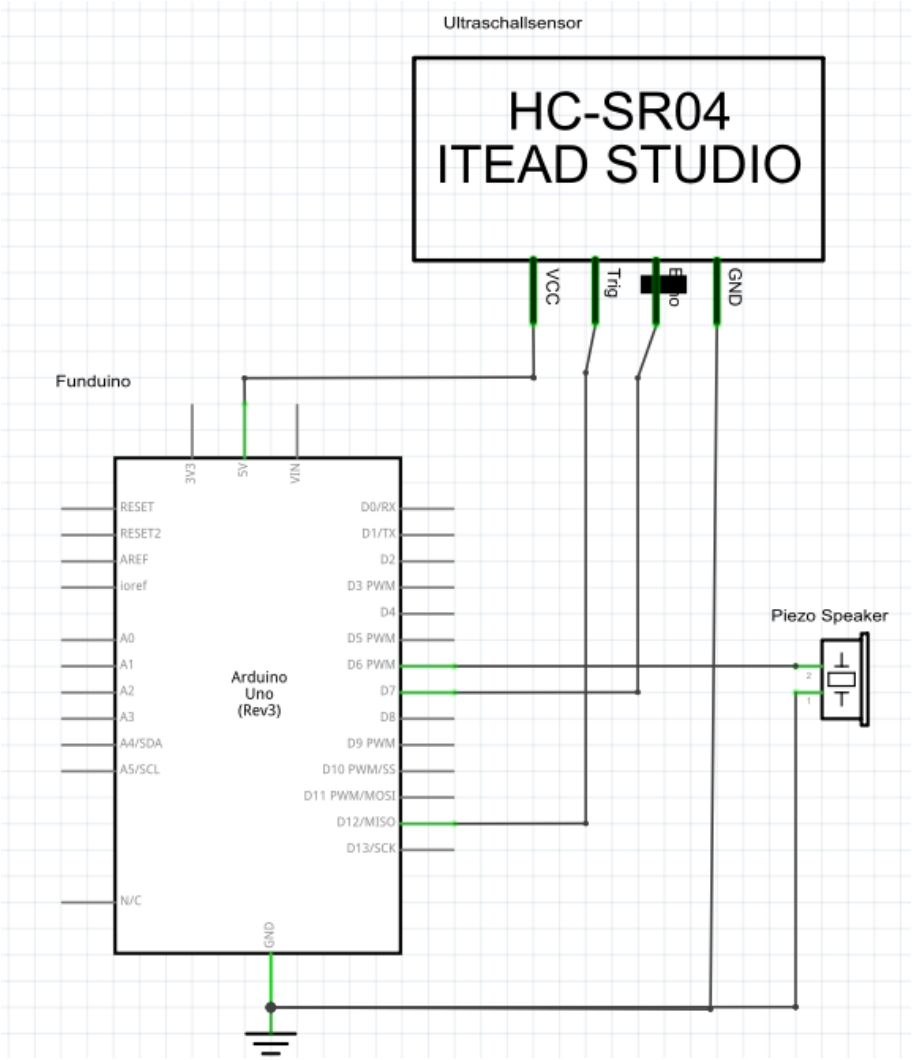
Funktion:

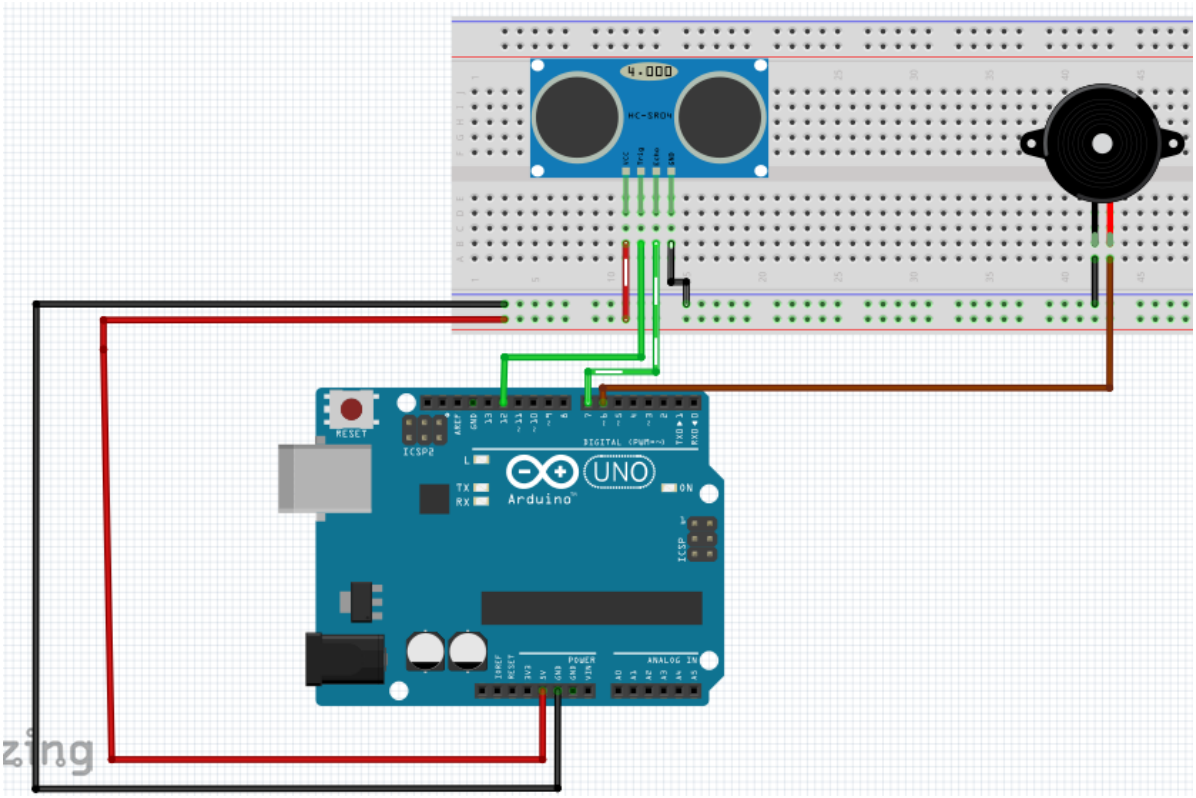
Der Ultraschallsensor soll Werte an den Funduino weiterleiten, durch welche der Abstand in Zentimeter berechnet werden kann. Je nach Größe des Abstandes zu einem Hindernis soll der Piezo Speaker in unterschiedlicher Geschwindigkeit einen Signalton ausgeben.

Anschlussplan und Schaltplan:

Achtung: Die Darstellung des HC-SR04 Sensors habe ich aus dem Internet heruntergeladen, da sie nicht bei den standartmäßigen Bauteilen der Fritzing Software enthalten war. Der Ersteller dieser Grafik ist „daminbob“.

→ siehe Seite 93 für die Quellenangabe dieses Downloads





Programmierung:

Die Programmierung des Ultraschallsensors hat mir einige Probleme bereitet, da das Ermitteln des Abstandes und das Ausgeben eines schnellen bzw. langsamen Pieps Signals ohne delay()-Funktion aufwendig ist. Es muss also ein Weg gefunden werden um die gewünschte Funktion der Einparkhilfe zu erreichen, ohne eine Verzögerung des Gesamtprogramms zu erhalten. Dies habe ich wie folgt gelöst:

Das Unterprogramm einparkhilfe() wird immer nur nach Ablauf einer bestimmten Zeit aufgerufen. Hierzu verwende ich dieselbe Methode, wie für das Blinken der Signal LED (Seite 39). Die Zeit, nach deren Ablauf das Unterprogramm aufgerufen wird, ist jedoch abhängig vom zuvor ermittelten Abstand. Sie wird in der Variable „sendedauer“ gespeichert. Da diese Variable von dem zuvor ermittelten Abstand abhängt, wird bei einem kleinen Abstand das Unterprogramm öfter/schneller aufgerufen als bei großen Abständen.

```
sendedauer = abstand*30;
if(zeit - sendewechsel > sendedauer)
{
  einparkhilfe();
}
```

einparkhilfe():

Im Programmteil der Einparkhilfe wird mit Hilfe der „NewPing“ Library der Abstand ermittelt. Der Ersteller dieser Library ist Tim Eckel, ich habe sie mir über die Arduino Website heruntergeladen (siehe Quellenangabe Seite 93).
Durch die Verwendung dieser Library wird kein delay() für das „HIGH-“ und „LOW-“ Setzen des TRIGGER-Pins des Ultraschallsensor benötigt.
Ist der Abstand größer als 40cm, soll der Piezo Speaker keinen Signalton ausgeben. Ist der ermittelte Abstand jedoch kleiner als 40cm, wird der Status des Piezo Speakers umgekehrt. Wird nun der ermittelte Abstand immer kleiner, wird das Unterprogramm immer öfter ausgeführt und der Status des Piezo Speakers wird immer öfter gewechselt. Das Signalpiepsen des Speakers wird also immer schneller, je kleiner der Abstand ist.

```
void einparkhilfe()
{
  sendewechsel = zeit;
  unsigned int uS = sonar.ping();
  abstand = (uS / US_ROUNDTRIP_CM);
  if(abstand >1 && abstand <40 )
  {
    if (pips == false){
      digitalWrite(sound,HIGH);
      pips = true;
    }
    else {
      digitalWrite(sound,LOW);
      pips = false;
    }
  }
  else
  {
    digitalWrite (sound,LOW);
    pips = false;
  }
```

2.3.7 Der Sensorentest

Wenn Fehler im Programm auftreten, ist es sehr mühsam und zeitaufwendig die Ursache dieser Fehler zu finden. Oft liegen diese Fehler daran, dass die Sensoren falsche Werte an den Funduino liefern.
Um die Werte meiner 3 verwendeten Sensoren zu kontrollieren, habe ich für jeden Sensor jeweils einen Testmodus erstellt:

Zu testender Sensor	Name des Testmodus
TMP 36 - Temperatursensor	DEBUG_TEMPERATUR
Fotowiderstand - Helligkeitssensor	DEBUG_HELLIGKEIT
HC-SR04 - Ultraschallsensor	DEBUG_ABSTAND

Um die Testmodi zu starten muss im Variablendeklarationsteil der jeweilige Testmodus definiert werden. Dies geschieht indem man das Kommentarmerkmal „//“ entfernt.
Nach der Definition eines Sensorentestmodus muss das Programm neu auf den Arduino geladen werden.

```
boolean testlauf = false;              // Variable ob Sensorentestmodi ausgeführt wird
//# define DEBUG_TEMPERATUR             // Testmodus-Temperatursensor
//# define DEBUG_HELLIGKEIT             // Testmodus-Helligkeitssensor
//# define DEBUG_ABSTAND                // Testmous-Ultraschallsensor
```

Im Programm befinden sich mehrere „#ifdef….“ Befehle. Diese „if-Funktion“ wird nur dann ausgeführt, wenn ein Sensorentestmodus definiert ist. Ist dies der Fall, wird im setup()-Teil des Programmes eine serielle Datenübertragung mit 19200 baud gestartet und eine „testlauf-Variable“ auf „true“ gesetzt. Dadurch wird die serielle Datenübertragung auf 9600 baud nicht gestartet, da die Bedingung dafür der „false-Zustand“ der „testlauf-Variable“ ist. Dasselbe geschieht mit dem kompletten Programmteil im loop()-Abschnitt, in welchem die Bluetooth Codes eingelesen und bearbeitet werden. Er wird auch nicht ausgeführt.
Da nun die kompletten Bluetooth Codes nicht über die serielle Schnittstelle an PIN 0, 1 eingelesen werden, steht diese serielle Schnittstelle zur Datenübertragung mit dem PC zur Verfügung.

z.B.:

```
#ifdef DEBUG_ABSTAND              // |
  Serial.begin(19200);            // |
  testlauf = true;                // |
#endif                            // |

  if ( testlauf == false){        // wenn kein Sensorentestlauf --> serielle Übertragung mit BT- Modul
    Serial.begin(9600);
  }
```

Im Programm befinden sich weitere „ifdef…“Funktionen die bei Definition des jeweiligen Testmodus das passende Unterprogramm zu dem Sensor aufrufen. Die ermittelten Sensorwerte werden nun über die serielle Schnittstelle an den Serial Monitor gesendet und dort können diese über die Übertragungsrate 19200 baud ausgelesen werden.

```
#ifdef DEBUG_ABSTAND              //  |
  if (abstand>= 50){              //  |
    Serial.println(">50 cm");     //  |
    delay(500);                   //  |
  }                               //  | wenn Sensortestmodus für Abstand defined:
  else{                           //  | Abstand an Serialmonitor senden
    Serial.print(abstand);        //  |
    Serial.println("cm");         //  |
    delay(500);                   //  |
  }                               //  |
#endif                            //  |
```

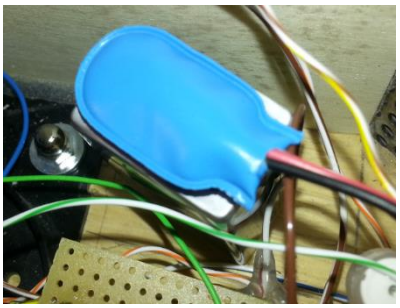
Dies ist eine einfache Testmöglichkeit um die ermittelten Werte der Sensoren einzusehen, da man am Hardwareteil des Arduino Car nichts verändern muss. Auch im Programm ist diese Testmöglichkeit mit geringem Aufwand zu bewältigen, da man nur ggf. die „//“ entfernen oder hinzufügen muss.
Der größte Vorteil an dieser Testmethode ist, dass der Compiler*[6] der Arduino Software das komplette Programm für den Sensorentestmodus ignoriert, wenn diese nicht definiert ist. Dadurch tritt kein Speicherplatzverlust bei normaler Verwendung des Programms auf.

*[6] Compiler: Programm in der Arduino Software, welches den Quelltext des Programmes zusammenfasst, umwandelt und an den Arduino ausgibt.

2.4 Spannungsversorgung

Für mein Projekt benötige ich verschiedene Spannungsquellen, da die Bauteile unterschiedliche Spannungen benötigen. Das Problem ist jedoch, dass ich keine Netzteile verwenden kann, sondern mobile Spannungsquellen verwenden muss. Diese müssen die benötigte Spannung und auch eine relativ hohe Stromstärke für die Motoren liefern.

Um die benötigte Spannung und Stromstärke zu erhalten habe ich vier verschiedene Spanungsversorgungen gewählt:



1. NiMH 9V Block Akku, 250 mAh

Der 9V Block Akku ist die Spannungsquelle die mit dem Funduino verbunden ist. Sie liefert Strom für die Steuervorgänge im Funduino, die Funduino-Pins und die Sensoren. Da diese Steuervorgänge nur geringe Stromstärken benötigen, reicht hier der 9V Block Akku mit 250mAh aus, um diese eine ausreichende Zeit lang zu betreiben.

Der Block Akku liefert auch die Spannung für die 5V und 3,3V Funduino-Pins.

2. NiMH 9V Block Akku, 250 mAh

Ein zweiter 9V Block Akku dient als Spannungsquelle für die Verbraucher am Funduino, welche nur kleine Stromstärken benötigen. Der 9V Block Akku liefert Strom für die beiden Rücklichter, den Scheinwerfer und den CPU-Kühler.
Die benötigte Stromstärke dieser Verbraucher beträgt etwa:

$$I_{benötigt} \approx 2 \cdot I_{F\ Rücklicht} + I_{C\ Scheinwerfer} + I_{C\ CPU-Kühler}$$

$$I_{benötigt} \approx 2 \cdot 16{,}67mA + 58{,}68mA + 162{,}8mA$$

$$I_{benötigt} \approx 254{,}82\ mA$$

Da der Block Akku eine Nennladung von 250mAh hat, müsste dieser den benötigten Strom ca. eine Stunde lang liefern können.

→ Die zwei NiMH 9V Block Akkus können mit dem Universal Profi-Schnell-Ladegerät (für NiMH-Akkus) aufgeladen werden und dadurch minimieren sich Aufwand und Kosten um diese Spannungsversorgung aufrecht zu erhalten.

3. 4,8V Ni-Cd Battery Pack, 650mAh

Dieser Battery Pack dient zur Spannungsversorgung des Servos MT 995, da er die hohe benötigte Stromstärke des Servos liefern kann.

Der Battery Pack von NIKKO ist über eine Aufladestation direkt an 230V aufladbar.

4. Reihenschaltung aus 8x 1,5V Varta Energy AA Mignon Alkaline Batterie, ca. 2500 mAh

Die Reihenschaltung aus 8 x 1,5V Alkaline-Zellen dient als Spannungsversorgung für den Antriebsmotor des Arduino Car.
Bei der Erstellung dieser passenden Spannungsversorgung für den Motor gab es jedoch mehrere Probleme:
Zuerst sollte der Motor durch eine Reihenschaltung aus 8 x NiMH AA Akkus 1,2V angetrieben werden. Dieser selbstgebastelte Akkublock sollte die Wiederaufladbarkeit der einzelnen AA Akkus gewährleisten und einen möglichst langen Betrieb des Motors ermöglichen. Es stellte sich jedoch heraus, dass dieser Akkublock NICHT für den Motor geeignet war.
Der Motor lief nicht mit dieser Spannungsquelle an und die Spannung am Motor bzw. am Motortreibermodul L298N brach ein. Deswegen lagen am Motor statt 9,6V nur noch ca. 4V an. Ein Grund für diesen Spannungseinbruch war, dass der Motor zu große Stromstärken beim Anfahren benötigt und der Akkublock diese nicht liefern konnte. Ein weiterer möglicher Grund könnte der hohe Innenwiderstand der acht AA NiMH Akkus sein.

Als Notlösung verwende ich eine Reihenschaltung aus 8 x AA Mignon Alkaline Batterien, 1,5V. Da diese ebenfalls in Reihe geschaltet sind, bleibt die Nennladung von ca. 2500mAh gleich wie bei einer Batterie, aber die Spannung der selbstgebastelten Spannungsquelle beträgt 12V. Durch die Reihenschaltung der einzelnen Alkaline-Zellen erreicht man also eine höhere Spannung bei gleichbleibender Nennladung.
Mit diesem Batterieblock kann nun der Motor betrieben werden. Er läuft viel kraftvoller und die Spannung bricht auch beim Anfahren nicht ein.

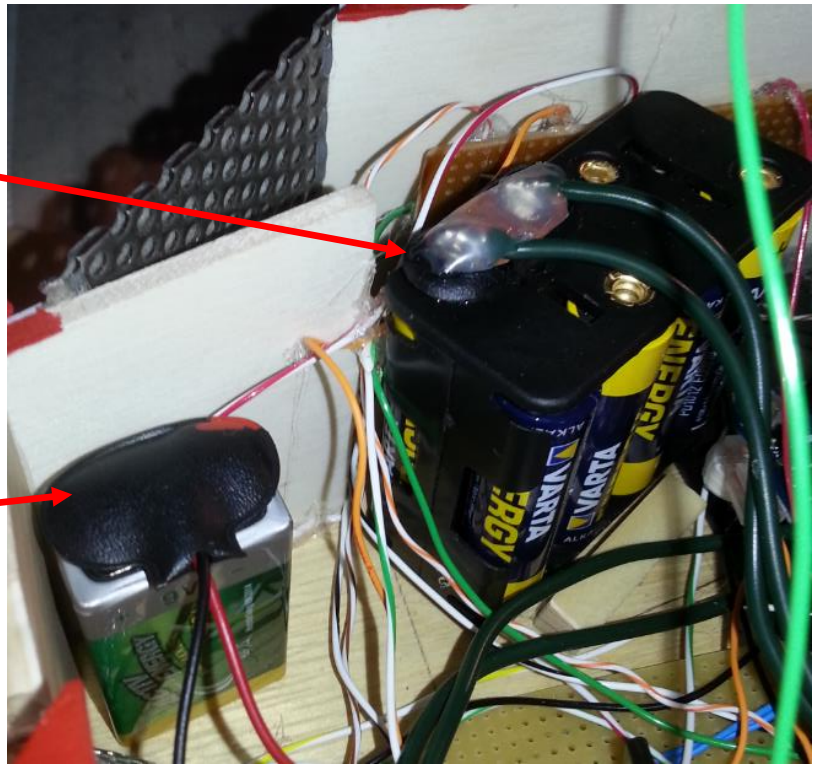
$$I_{benötigt} \approx \emptyset\ 1{,}3A$$

Die Reihenschaltung aus 8 x AA Mignon Alkaline-Zellen sollte bei einer Nennladung von ca. 2500mAh die benötigte Stromstärke des Motors von ca. 1,3A etwas weniger als 2 Stunden liefern können.

8x 1,5V Varta Energy AA Mignon Alkaline Batterie

NiMH 9V Block Akku, 250 mAh

→U_{bL} = 9V



Da diese Spannungsquelle aus 8 x AA Mignon Alkaline-Zellen mit 12V nicht geplant war, ergaben sich dadurch auch einige andere Probleme:

- Die neue Spannungsquelle liefert 12V. Da der alte Akkublock mit 9,6V jedoch zusätzlich Spannungsquelle für die Rücklichter, den Scheinwerfer und den CPU-Kühler sein sollte, sind die dazugehörigen Schaltungen auch mit 9,6V Betriebsspannung berechnet worden. Mit 12V können sie nicht betrieben werden, da dann zu viel Spannung an den LEDs anliegen würde und auch der Strom zu stark ansteigen würde. Deswegen musste ich für diese Verbraucher eine extra Spannungsquelle benutzen. Hierfür habe ich einen zweiten NiMH 9V Block Akku verwendet (siehe Seite 60).
  Die Betriebsspannung U_{bL} für die Rücklichter, den Scheinwerfer und den CPU-Kühler beträgt nun 9V. Die Vorwiderstände die schon fertig eingebaut waren, sind jedoch auf 9,6V ausgelegt und dadurch sind für die verbesserten, errechneten Widerstandswerte für U_{bL}= 9V nicht die optimalen Widerstände gewählt. So ist auf Seite 42 z.B. satt des optimalen nächst kleineren Widerstandswertes aus der E-12 Reihe der übernächst kleinere Widerstandswert für R_B gewählt worden. Da dies jedoch die Stromstärke durch die LEDs nur verringert und kein Schaden verursacht, habe ich diese Widerstände beibehalten.

- Die 8 x AA Mignon Alkaline-Zellen sind nicht wieder aufladbar.

Alle verwendeten Spannungsquellen:

8x 1,5V Varta Energy AA Mignon Alkaline Batterie

4,8V Ni-Cd Battery Pack, 650mAh

NiMH 9V Block Akku, 250 mAh

NiMH 9V Block Akku, 250 mAh

→U_{bL} = 9V

2.5 Belegungsplan der Funduino-Pins

Projekteinzelteile	Bauteile	Pin-Belegung
Lenkung / Motorsteuerung	Servo MT 995	Pin 8
	HC-05 Modul	RXD an Pin 1 TXD an Pin 0
	L298N Modul	IN1 an Pin 4 IN2 an Pin 2 ENA an Pin 3
Signal LED	Schaltung Signal LED	Pin 13
Rücklicht	Rücklichtschaltung links	Basis Pin 9
	Rücklichtschaltung rechts	Basis Pin 10
Scheinwerfer	Fotowiderstand	Pin A2
	Scheinwerferschaltung	Basis an Pin 11
Kühlung	TMP 36	Pin A1
	Lüfter Schaltung	Basis an Pin 5
Einparkhilfe	Piezo Speaker	Pin 6
	HC-SR 04	Trigger Pin an Pin 12 Echo Pin an Pin 7

III. Chassis und Gehäuse

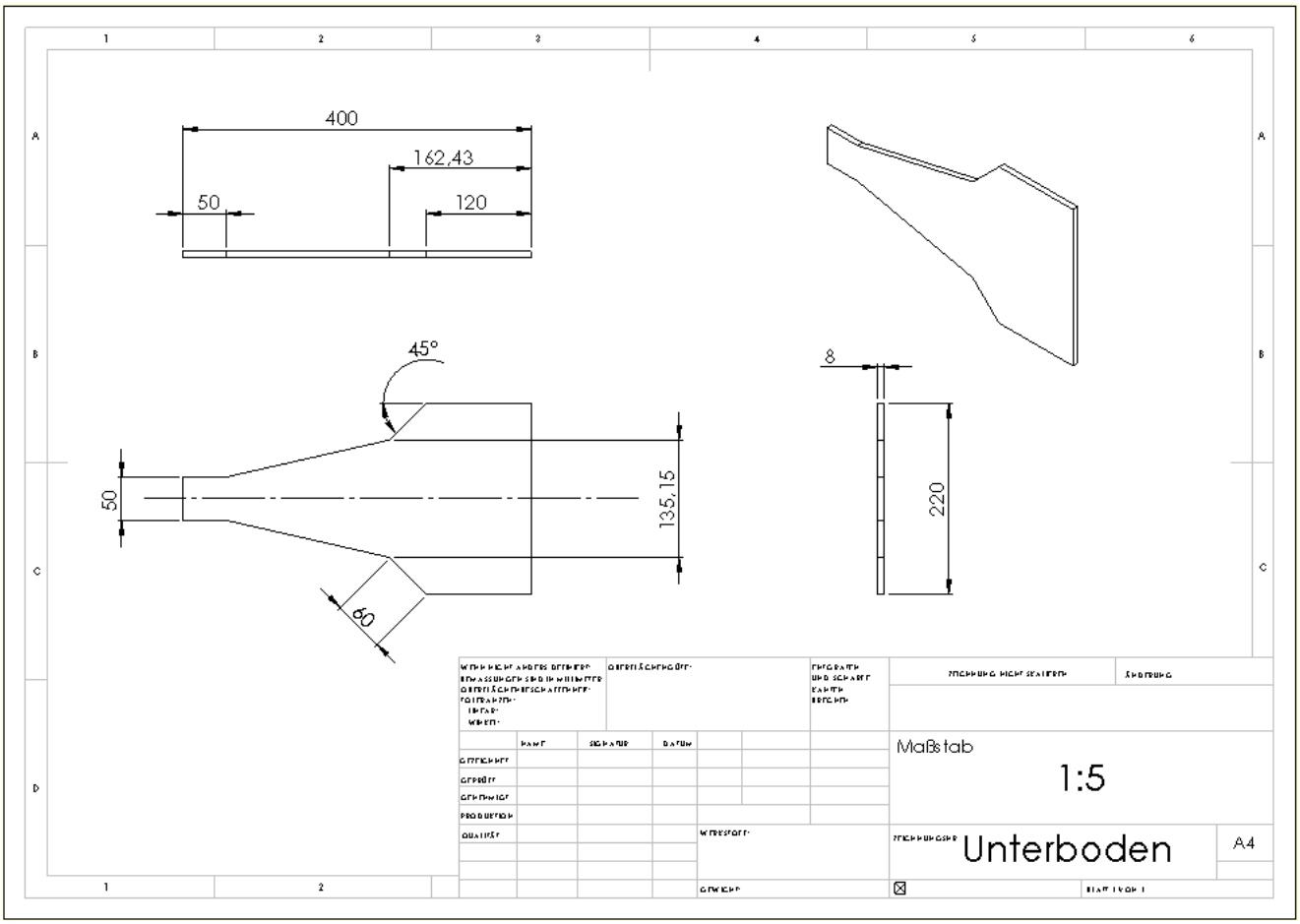
Das Chassis und das Gehäuse meines Arduino Car wird aus Holz gebaut. Gründe dafür sind, dass Holz relativ leicht und gut zu bearbeiten ist. Außerdem stehen mir die benötigten Werkzeuge und Maschinen zur Verfügung und sind mir vertraut.

Übernommene Bauelemente:

Die Form des Arduino Car wird hauptsächlich an die Vorder- und Hinterachse angepasst, da ich diese aus zwei verschiedenen ferngesteuerten Autos ausgebaut habe.
Die Vorderachse ist ursprünglich von einem alten RC-Car der Marke NIKKO. Hier habe ich die Lenkeinrichtung für die Räder größtenteils entfernt und davon nur die Lenkstange an den Vorderrädern beibehalten.
Die Hinterachse stammt aus einem großen ferngesteuerten Auto, von welchem Fernsteuerung und Spannungsversorgung defekt waren. Übernommen habe ich die komplette Hinterachse inkl. der zwei Räder, den Motor und die passende Übersetzung zwischen Motor und Hinterräder.
Für das Chassis meines Fahrzeugs habe ich die Hinter- und Vorderachse durch einen Unterboden miteinander verbunden. Dieser Unterboden besteht aus einem 8 mm starken Pressspanholz, welcher durch mehrere Schraubverbindungen fest mit der Vorder- und Hinterachse verbunden ist.

Dekupiersäge

Zeichnung des Unterbodens:



Vorgehen:

Zur Erstellung der einzelnen Holzteile für die Karosserie und das Chassis gehe ich immer nach demselben Muster vor. Zuerst entwerfe ich eine räumliche Zeichnung in 3D von dem zu fertigenden Teil. Daraus erstelle ich dann eine 2D Zeichnung. Dies mache ich mit der Software SolidWorks 2013 Edition.
Mit dieser Zeichnung als Vorlage säge ich die Umrisse der einzelnen Teile aus. Für lange, gerade Strecken verwende ich eine Kreissäge und bei kurvigen und kleinen Umrissabschnitten ist eine Dekupiersäge besser für die Bearbeitung. Anschließend wird das Teil durch Schleifen genau an das Auto angepasst. Zur Verbindung von mehreren Einzelteilen verwende ich Holzleim, Silikon, Spachtelmasse und ggf. notwendig Schraubverbindungen.

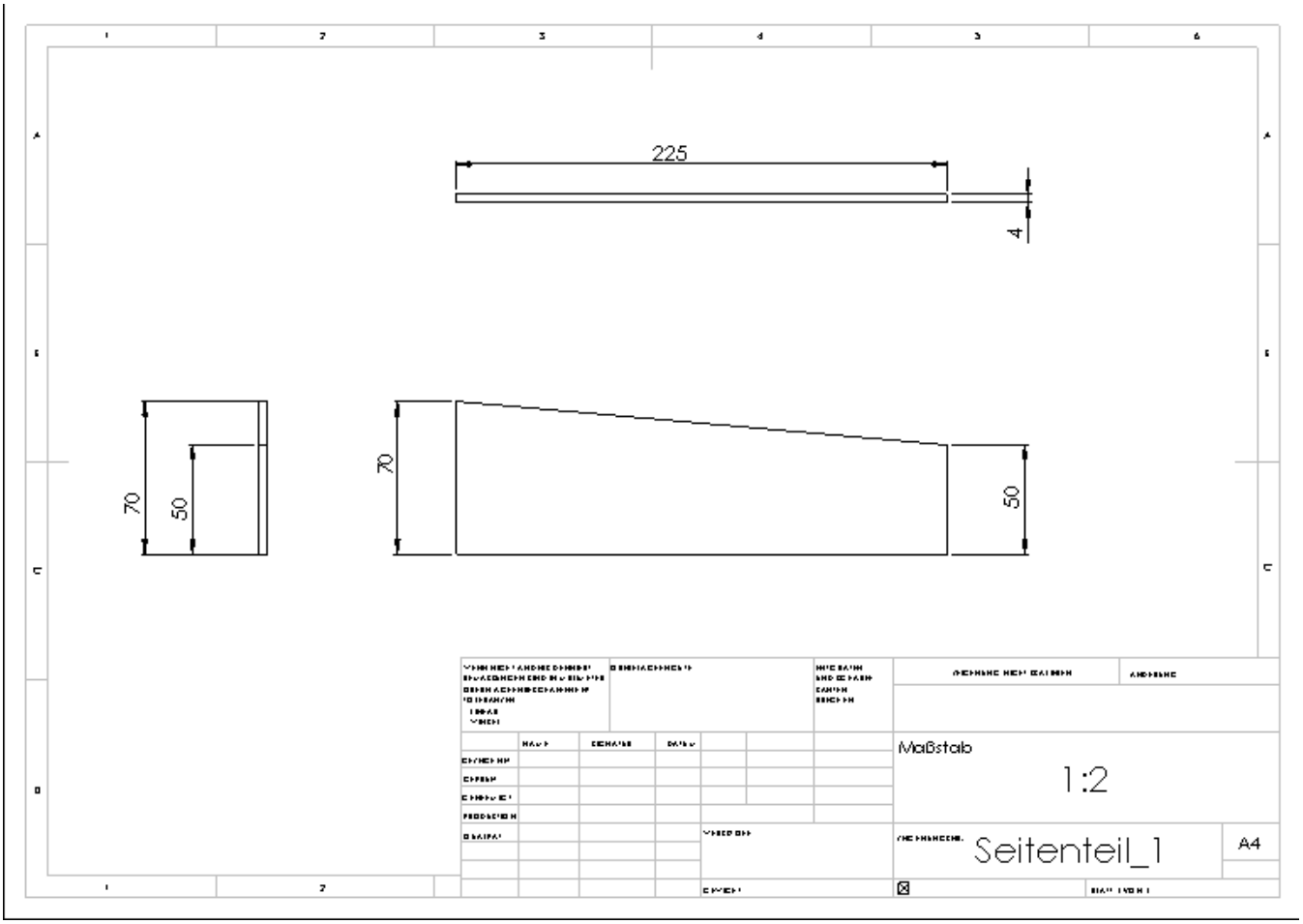
Karosseriebau:

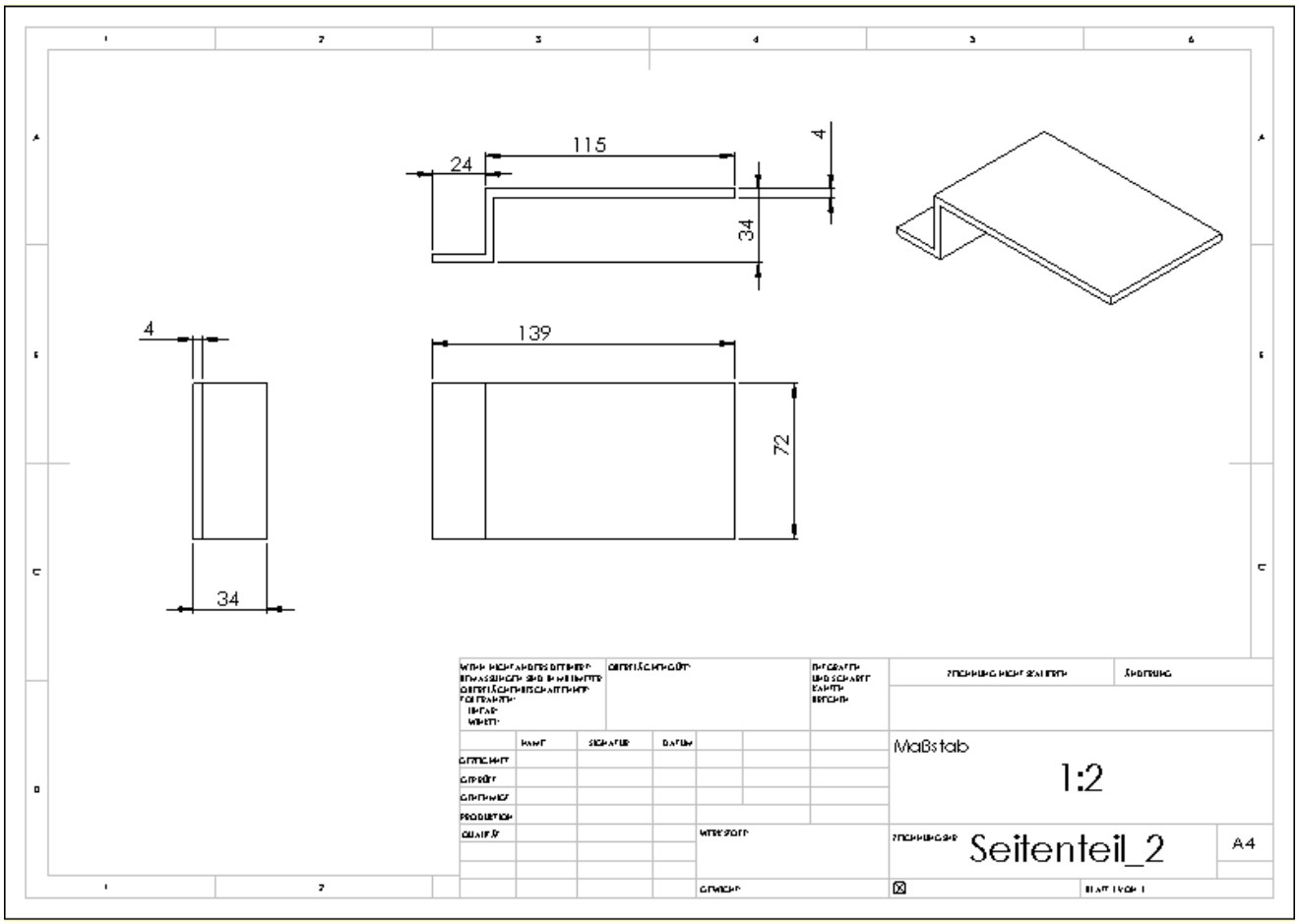
Als nächsten Schritt habe ich die Karosserie des Autos angefertigt. Die Form der Karosserie soll dabei der Form eines Sportwagens nahe kommen.
Sie besteht aus mehreren einzelnen Elementen die später zu der gesamten Karosserie zusammengesetzt werden. Jedes dieser Einzelteile besteht aus einer leichten, 4 mm starken Pressspanplatte.

Seitenteile:

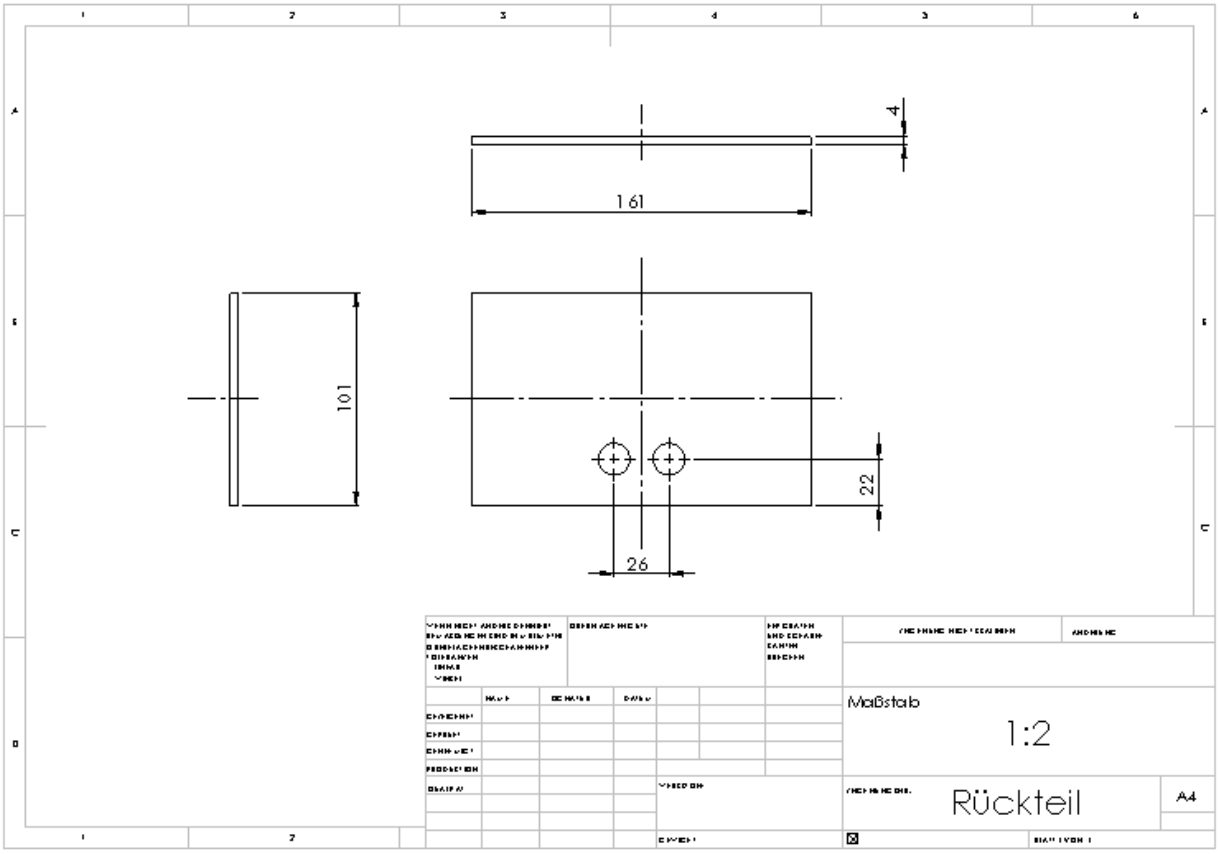
Es gibt insgesamt zwei Seitenteile für jede Seite des Autos. Das Seitenteil an der hinteren Achse des Autos besteht aus mehreren einzelnen Holzelementen, da die Größe der Reifen eine Verschmälerung des Unterbodens verlangt. Die zwei Seitenteile werden durch ein Metallgitter miteinander verbunden. Dieses Metallgitter ist mit Silikon an den Seitenteilen befestigt und erfüllt die Funktion der Luftzirkulation beim Einschalten des CPU-Kühlers.

Zeichnungen:





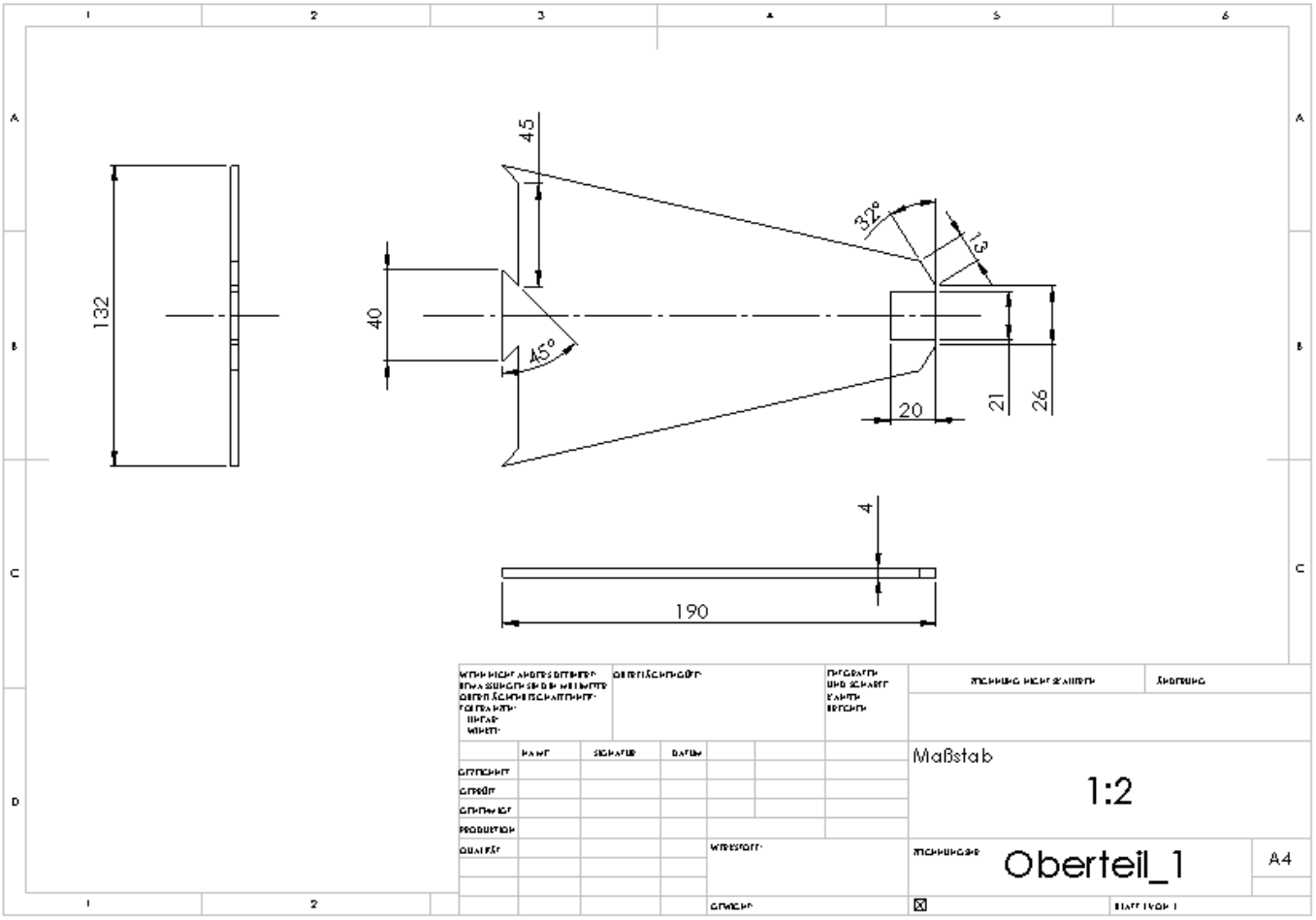
Zeichnung des Heckteils:

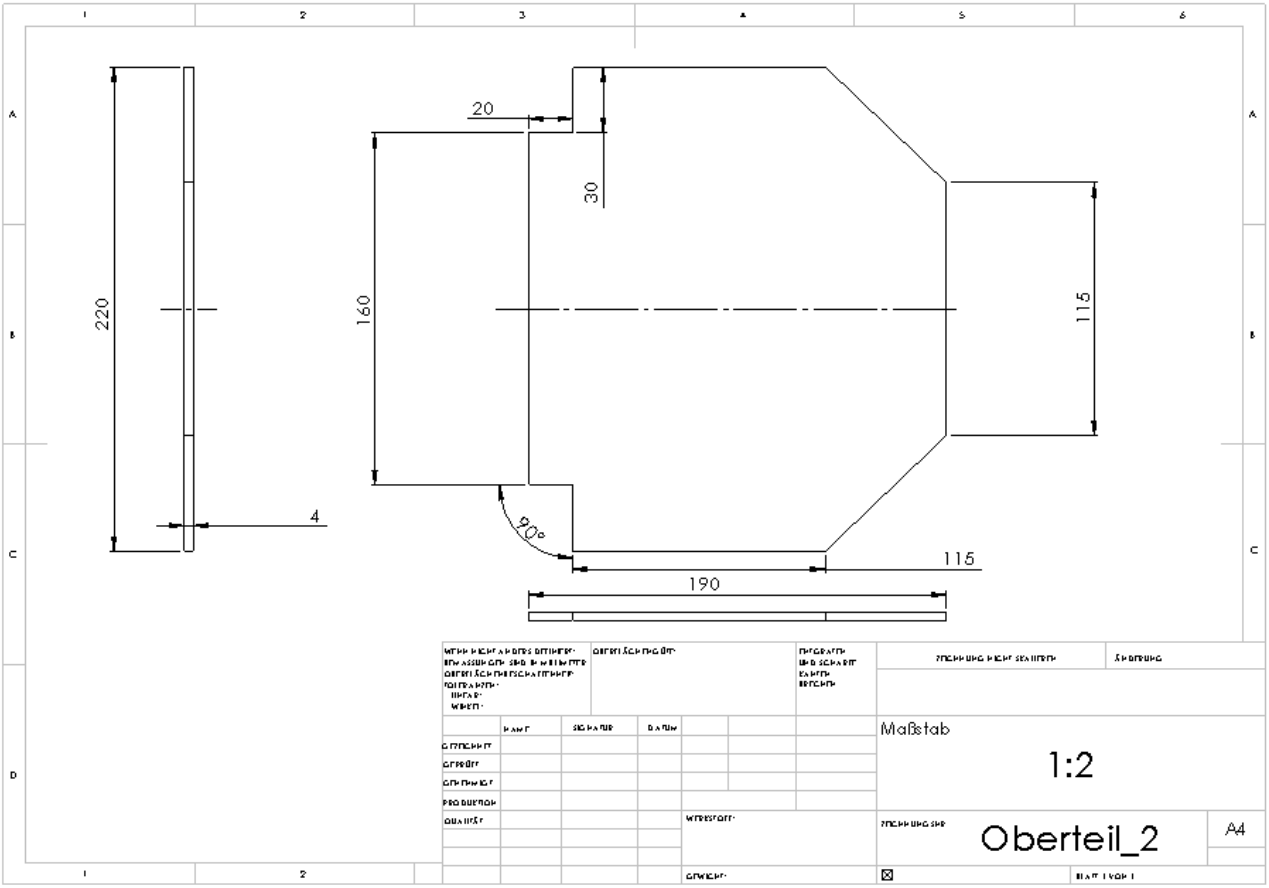


Abdeckung:

Die Abdeckung besteht ebenfalls aus zwei Elementen, welche genau auf den Grundriss des Unterbodens passen. Sie können durch ihre Form an der Verbindungsstelle zusammengesteckt werden. Im hinteren Element der Abdeckung ist ein Rennfahrer eingelassen.
Da im Innenraum der Karosserie später die gesamte Elektronik verbaut ist und sich dort auch die Akkus befinden muss dieser Bereich immer zugänglich sein. Die Abdeckung des Autos ist deswegen nicht fest mit der Karosserie verbunden. Sie sitzt trotzdem spielfrei auf der Karosserie, da sie durch eine Steckverbindung und eine Einkerbung für den Servo fest angepasst ist.

Zeichnungen:

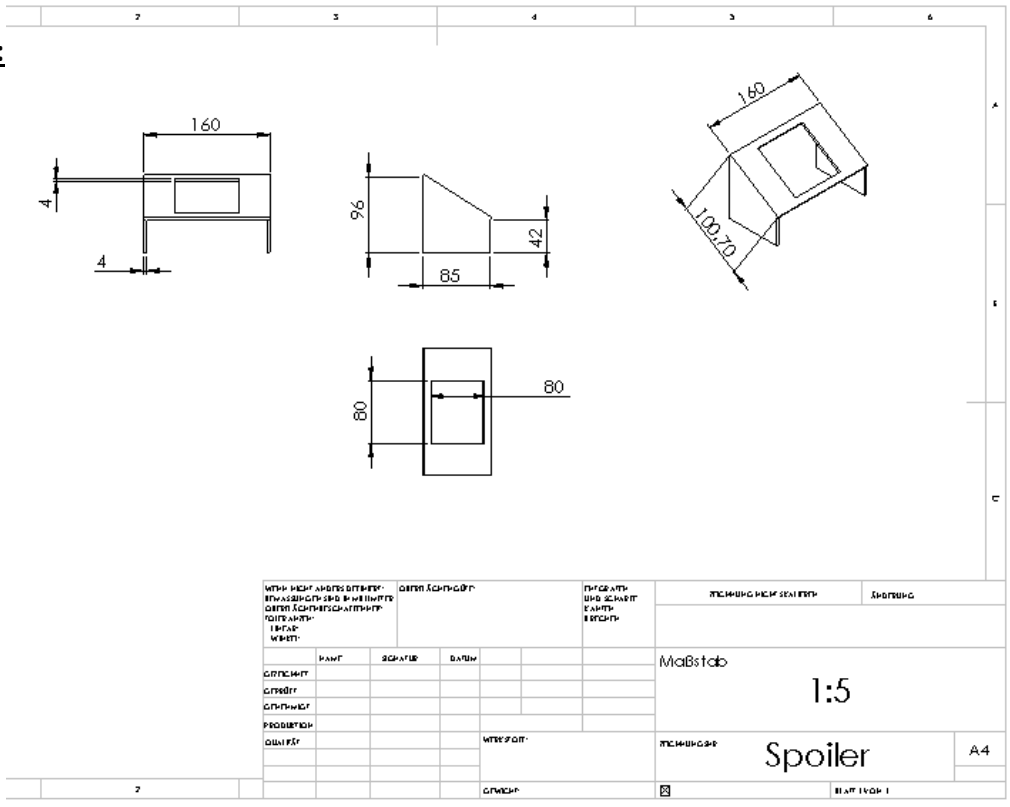




Spoiler:

Am Heck des Arduino Car befindet sich ein Spoiler. In dessen Oberfläche ist der CPU-Kühler eingebaut.

Zeichnung:



In der Karosserie eingelassene Elemente:

In der Karosserie sind weitere elektronische Bauteile eingebaut:

- der Fotowiderstand
- der Piezo Speaker
- das HC 05-Bluetooth Modul
- der Ein-/Ausschiebschalter (siehe Seite 82)
- die Signal LED
- Rücklichter
- Scheinwerfer
- HC-SR04 Ultraschallsensor
- USB A Stecker des kurzen USB Kabels

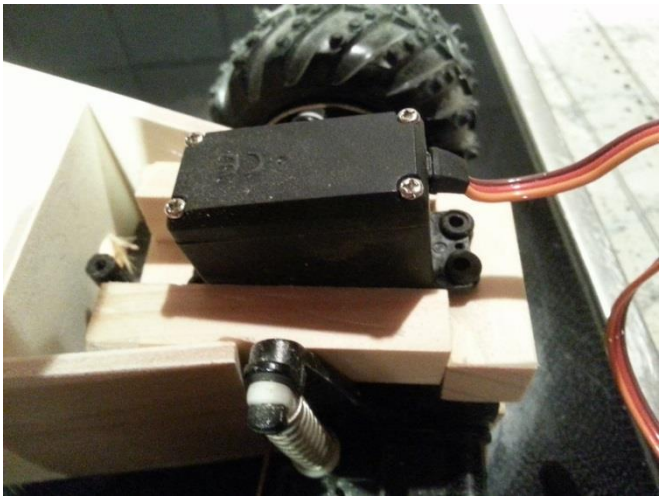
Meist sind diese Bauteile auf eine etwas größere Lochrasterplatte gelötet. Diese Lochrasterplatte wird nun von innen mit Silikon und/oder Heißkleber so befestigt, dass das Bauteil durch einen passenden Ausschnitt an der Karosserie an der Oberfläche zu sehen ist. Dadurch sind Aktionen, wie das Einschalten der Spannung für den Funduino von außen durch den Schiebeschalter durchführbar. Man kann nun auch deutlich Signale von zum Beispiel dem Piezo Speaker oder der Signal LED wahrnehmen.

3.1 Schwierigkeiten beim Karosseriebau

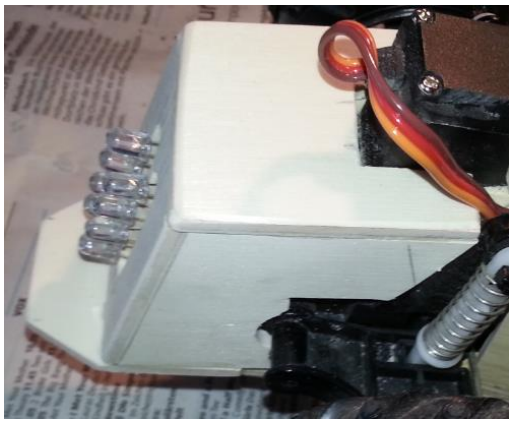
Die größte Schwierigkeit beim Karosseriebau war es, den Servo so zu platzieren, dass er direkt über der Lenkachse der Vorderräder steht und keinen Bewegungsspielraum hat. Dies wird durch mehrere kleine Holzblöcke erreicht, die fest mit der Karosserie und dem Chassis verklebt und verschraubt sind. Sie halten den Servo direkt an seiner Position.

Der Aufsatz des Servos ist fest durch eine kleine Schraube und Montagekleber mit der Lenkachse verbunden. Durch diese feste Verbindung und das Festhalten des Servos an seinem Platz wird die Servobewegung direkt auf die Lenkachse übertragen.

Es gab jedoch nicht nur beim Einbau des Servos, sondern auch bei der Auswahl Probleme. Ursprünglich wollte ich den Servo aus demselben Auto, aus dem ich den Motor und die Hinterachse übernommen habe, verwenden. Es stellte sich jedoch heraus, dass dieser Servo mit 5 Anschlüssen kein gewöhnlicher Servo war. Es war ein Motor der durch ein Potenziometer und eine Steuerelektronik auf einen bestimmten Winkel eingestellt wird. Diesen „Servo“ konnte ich dadurch nicht mit der Servo Library ansteuern. Er war nicht für mein Projekt geeignet.
Als nächstes habe ich versucht den im Funduino Kit mitgelieferten Servo SG 90 zu verwenden. Die Stellkraft von diesem Servo hat jedoch nicht ausgereicht um meine Vorderräder zu bewegen. Deswegen habe ich mir den stärkeren Servo MT 995 besorgt. Er hat bei 4,8V angelegter Spannung eine Stellkraft von ca. $9\mathrm{kg} \cdot \mathrm{cm}$. Dieser Servo ist somit stark genug die Lenkachse mit den Vorderrädern zu verstellen, wenn er passend und spielfrei eingebaut ist.



Ein weiteres Problem war es auch zum Abschluss des Karosseriebaus eine Halterung für den Scheinwerfer anzufertigen. Dieser musste in die Karosseriefront eingelassen werden, doch dort war nicht ausreichend Platz durch die Servobefestigung. Da das Anfertigen eines Anbaus der Karosseriefront sehr schwierig war, habe ich hier Hilfe von meinem Vater erhalten. Dieser Anbau musste passgenau zu der Servohalterung und der Abdeckung passen, um deren Funktion nicht einzuschränken. Um dies zu realisieren, haben wir einen offenen Holzquader gebaut. Danach haben wir die Schnittstellen zur Vorderachse bzw. zu der Servobefestigung mit einem Dremel-Schleifer in die passende Form gebracht.



Nach intensiver Bemalung des Autos sieht das Ergebnis des Karosseriebaus wie folgt aus:

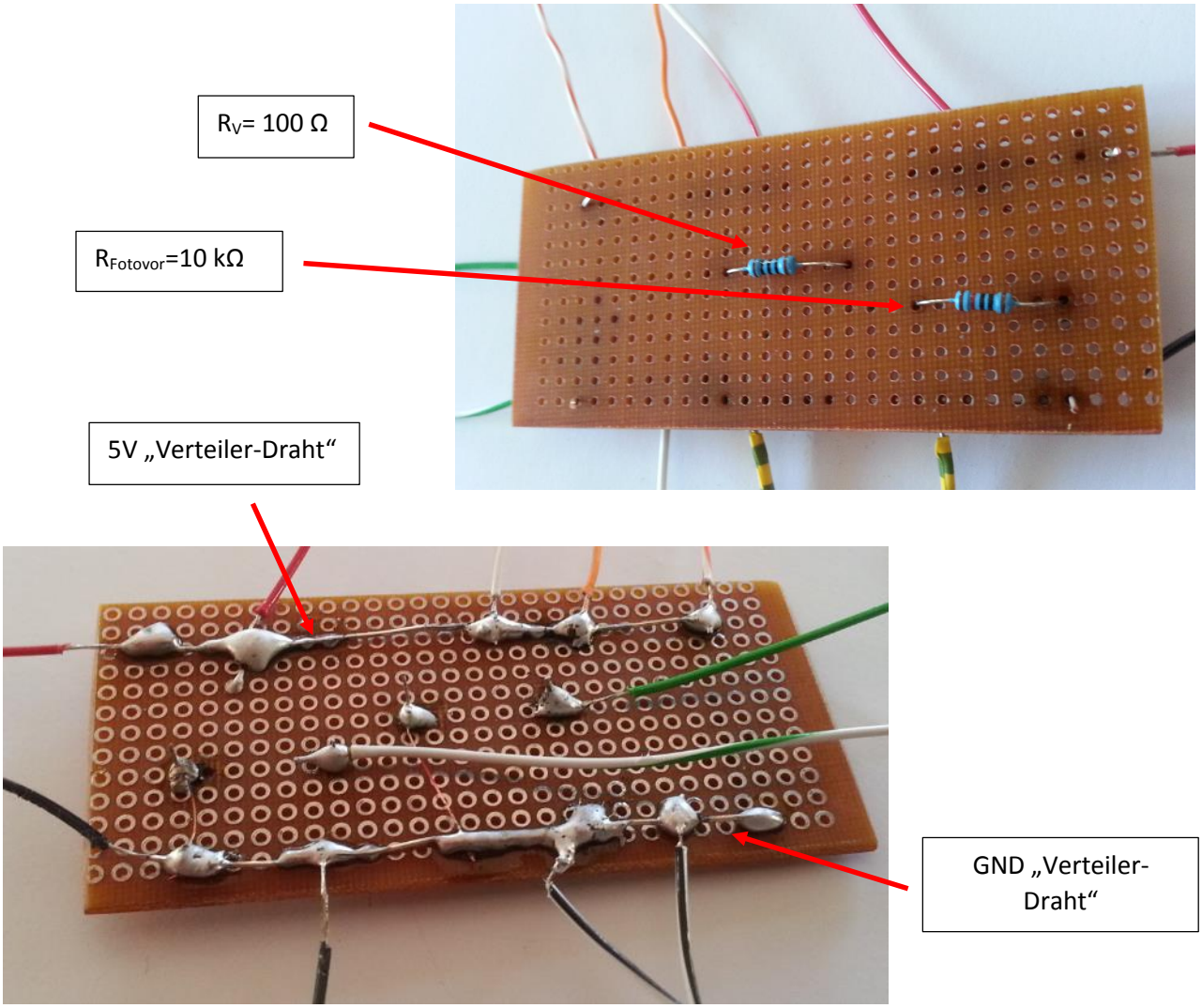
IV. Installation der Elektronik

Im Kapitel 2.3 meiner Projekteinzelteile habe ich schon die einzelnen Schaltungen zu den dazugehörigen Unterprogrammen meines Projekts erläutert. Diese Schaltungen sind jedoch meist mit Hilfe des Steckbretts aus dem Funduino Kit aufgebaut. Die Anschlusspläne dafür befinden sich ebenfalls in Kapitel 2.3.
Diese zusammengesteckten Schaltungen haben nur zu Testzwecken für die Projekteinzelteile gedient. Für die Installation der Elektronik in das Arduino Car sind diese Testschaltungen nicht geeignet.
Gründe dafür sind, dass das Steckbrett viel Platz benötigt und die nur gesteckten Bauteile während der Fahrt aus dem Steckbrett rutschen könnten. Um die geplante und benötigte Elektronik fest in meinem Auto zu installieren, baue ich einzelne Platinen, auf denen sich jeweils die Schaltungen befinden, die an die gleiche Spannungsquelle angeschlossen sind und dadurch die selbe Betriebsspannung benötigen. Als Grundelement der Platinen verwende ich Lochrasterplatten. Diese kann ich mit der Dekupiersäge und durch Schleifwerkzeuge so bearbeiten, dass sie später genau in ihrer Größe meinem Projekt angepasst sind. Auf diese Lochrasterplatten werden die entsprechenden Bauteile gelötet, um meine theoretisch erstellten Schaltungen in die Praxis umzusetzen.

Für mein Projekt fertige ich zwei Platinen an:

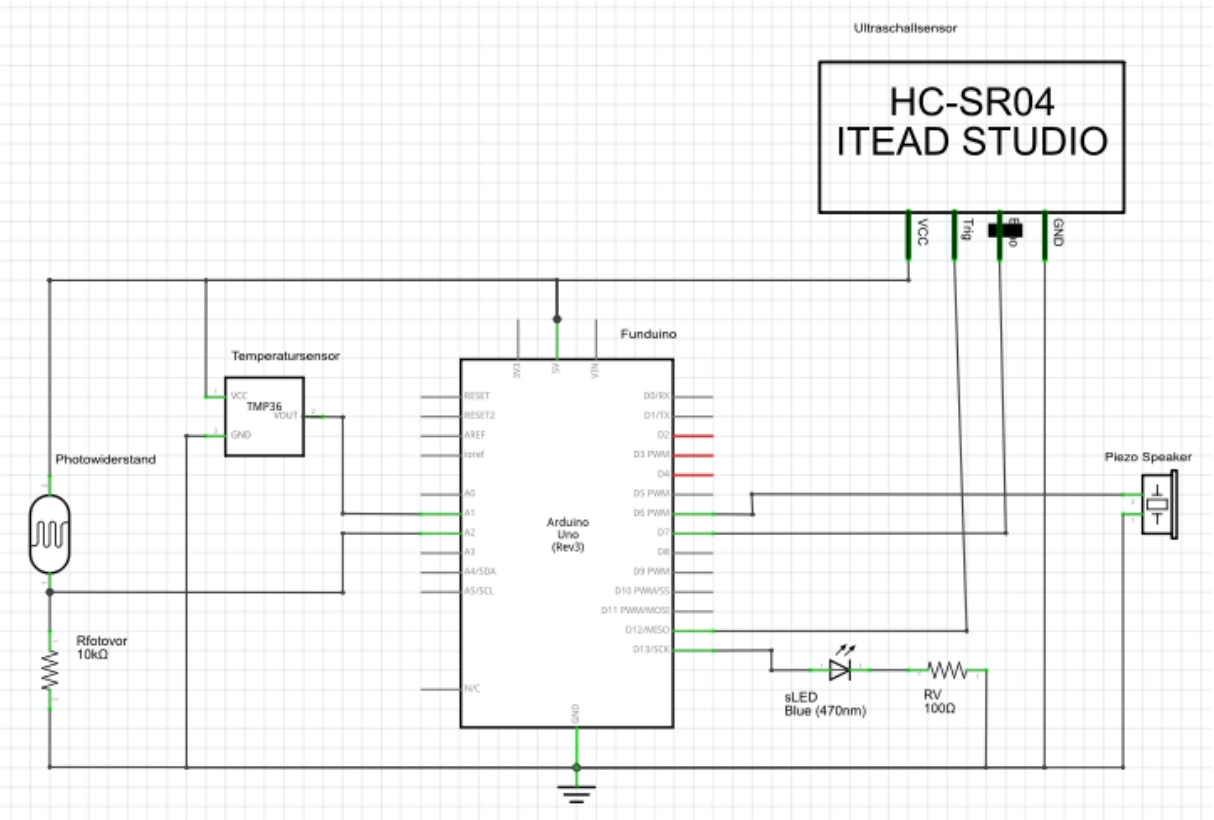
4.1 Platine für 5V Betriebsspannung

Auf dieser Lochrasterplatte befinden sich die Widerstände der Schaltungen, an denen die Betriebsspannung $U_{b\ Fun}$ = 5V anliegt. Dies ist der Vorwiderstand der Signal LED, R_V = 100Ω und der 10kΩ Fotovorwiderstand des Helligkeitssensors.
Des Weiteren sind auf der Rückseite dieser Platine zwei Stücke Silberdraht befestigt. Diese sind mit dem Ground und dem 5V Pin des Funduino verbunden. Alle Sensoren, welche eine Betriebsspannung von 5V benötigen, sind nun mit diesem 5V Silberdraht zusammengelötet. Der GND Anschluss der Bauteile wird mit dem Silberdrahtstück verlötet, welches mit dem Ground Pin des Funduino verbunden ist. Dadurch liegen an allen angeschlossenen Sensoren und Schaltungen die Betriebsspannung $U_{b\ Fun}$ = 5V an. Die Silberdrähte haben hier die praktische Funktion, dass sie durch ihre Länge viel Platz für die daran zu lötenden Anschlüsse bieten. Durch sie werden die einzelnen Schaltungen parallel zum 5V und Ground Pin des Funduinos geschalten und dies erklärt, warum überall die Spannung 5V anliegt.



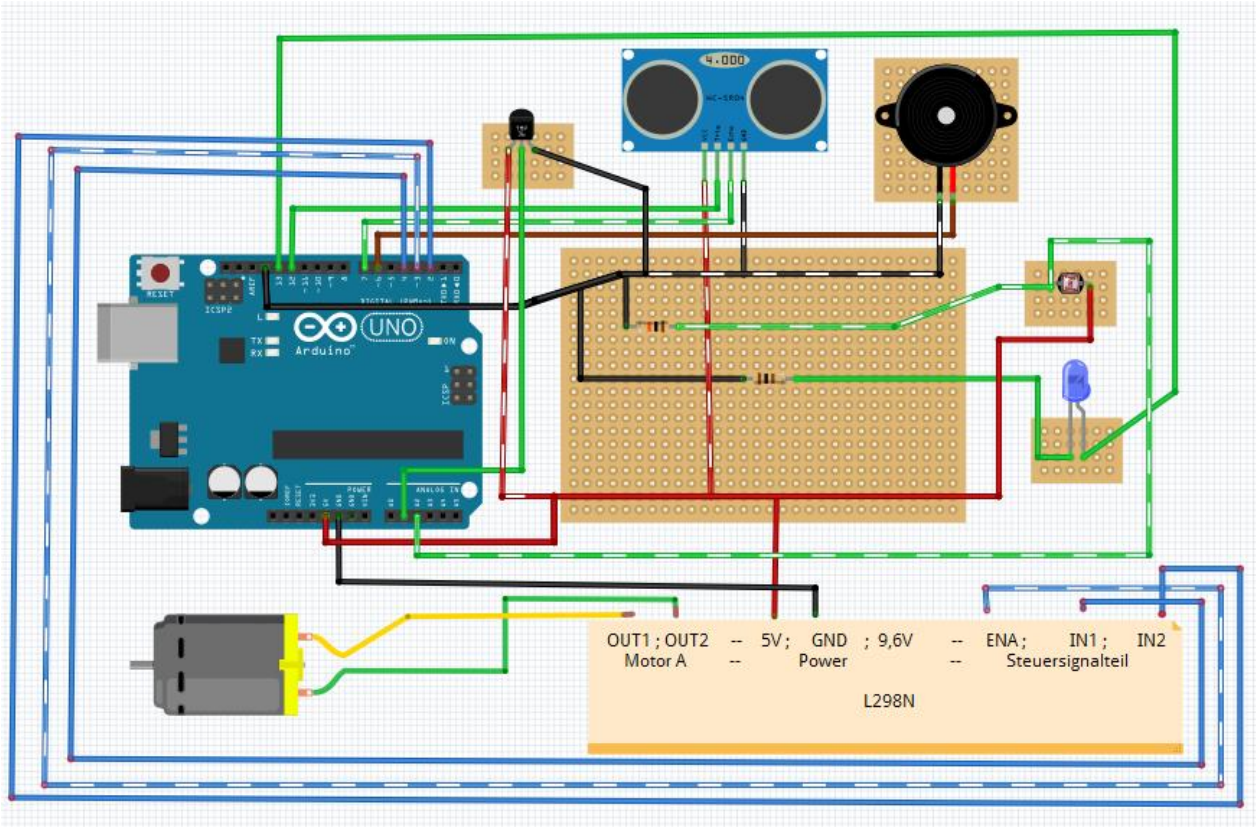
Schaltplan:

Der Schaltplan ist aus den einzelnen Schaltplänen aus dem Kapitel 2.3 zusammengesetzt:



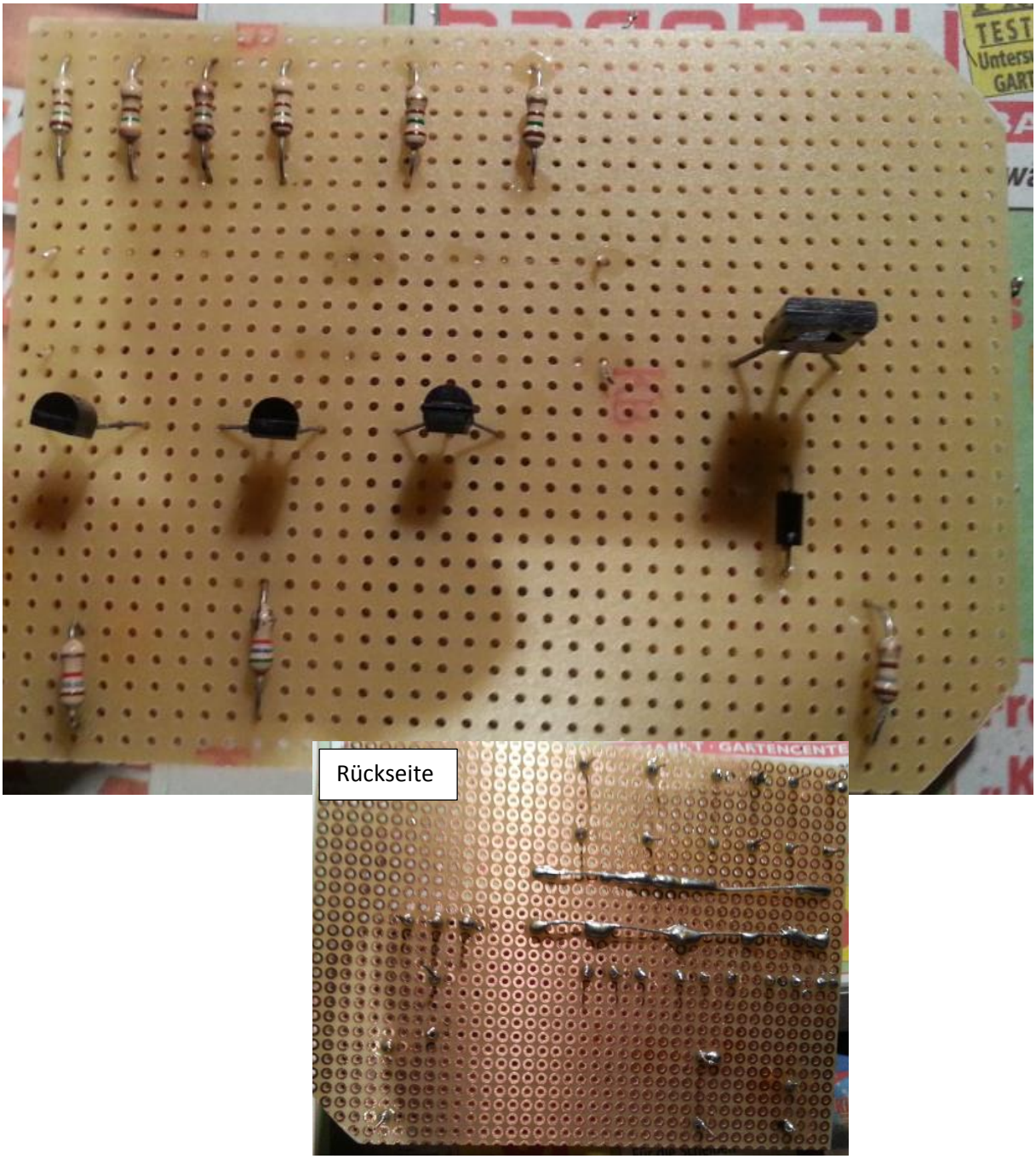
Anschlussplan:

Im Anschlussplan erkennt man, wie meine selbst gebaute Platine aufgebaut ist und die nötige Verkabelung wird veranschaulicht.



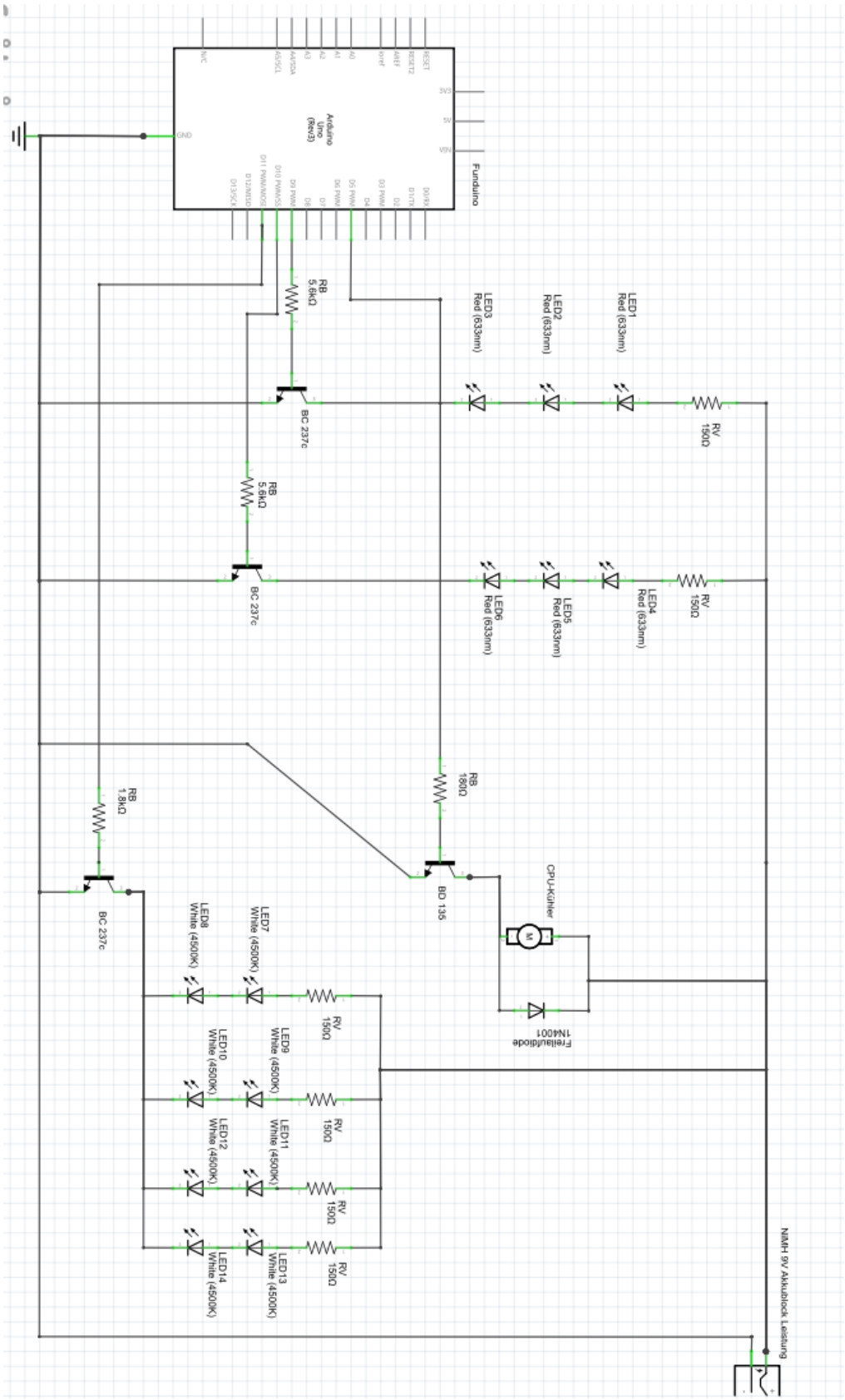
4.2 Platine für 9V Betriebsspannung

Auf dieser Lochrasterplatte befinden sich alle Transistorschaltungen meines Projekts. Hierzu gehören unter anderem die zwei Rücklichtschaltungen, die Scheinwerferschaltung und die Schaltung für die Kühlung. All diese Schaltungen, wie sie im Kapitel 2.3 aufgeführt sind, sind auf die Lochrasterplatte gelötet, um die dafür verwendeten Widerstände, Transistoren und die Diode auf einer Steuerplatine zusammenzufassen.
Auf dieser Platine befindet sich ebenfalls, wie auf der 5V Platine, ein Silberdrahtabschnitt, an welchem die Betriebsspannung für den Leistungsteil U_{bL} = 9V anliegt und ein mit Ground verbundener Silberdrahtabschnitt. Durch sie kann die Betriebsspannung U_{bL} = 9V leichter an die entsprechenden Schaltungen angelegt werden.



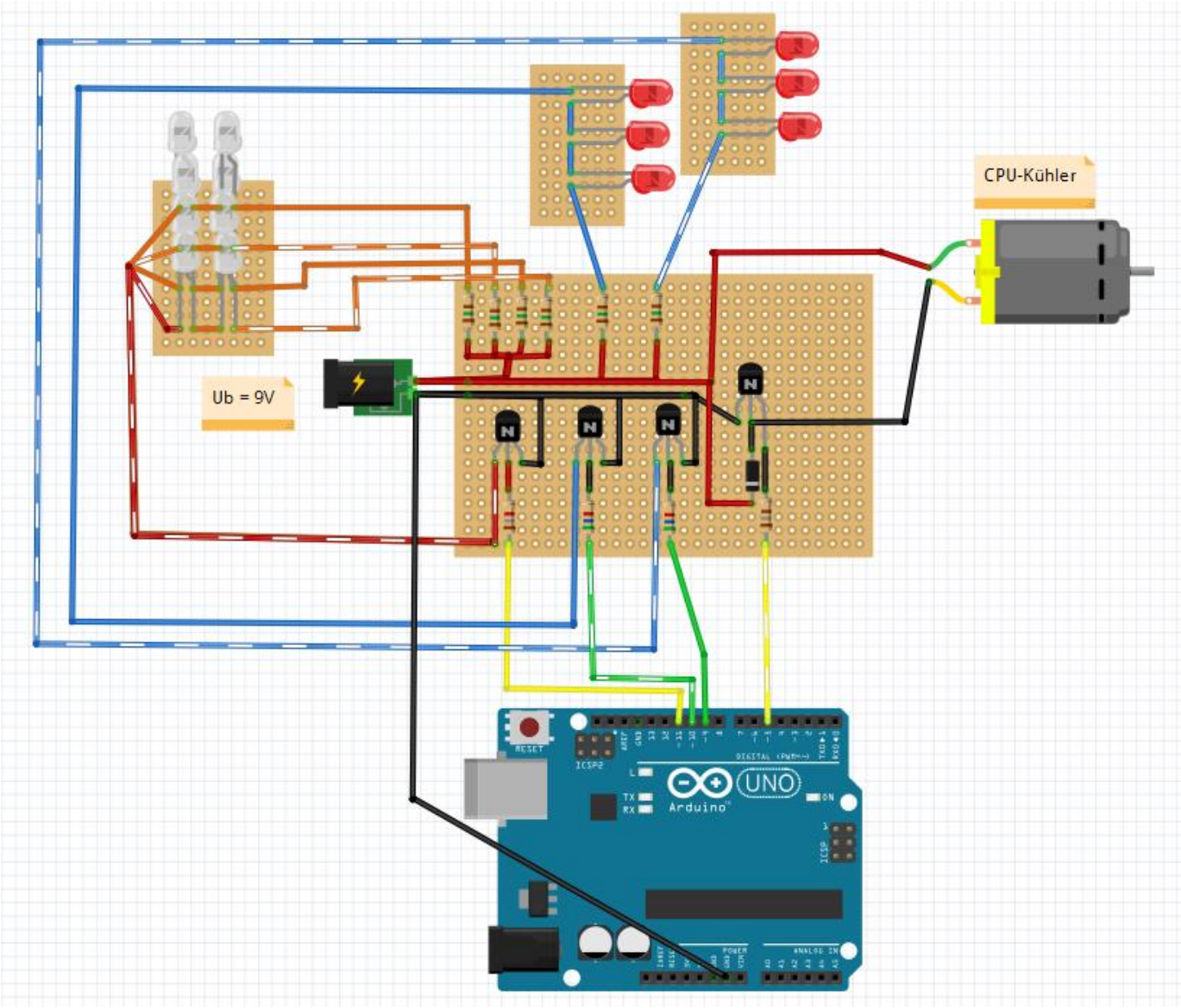
Schaltplan:

Der Schaltplan ist aus den einzelnen Schaltplänen aus dem Kapitel 2.3 zusammengesetzt:



Anschlussplan:

Im Anschlussplan erkennt man, wie diese Platine aufgebaut ist und die nötige Verkabelung wird veranschaulicht.

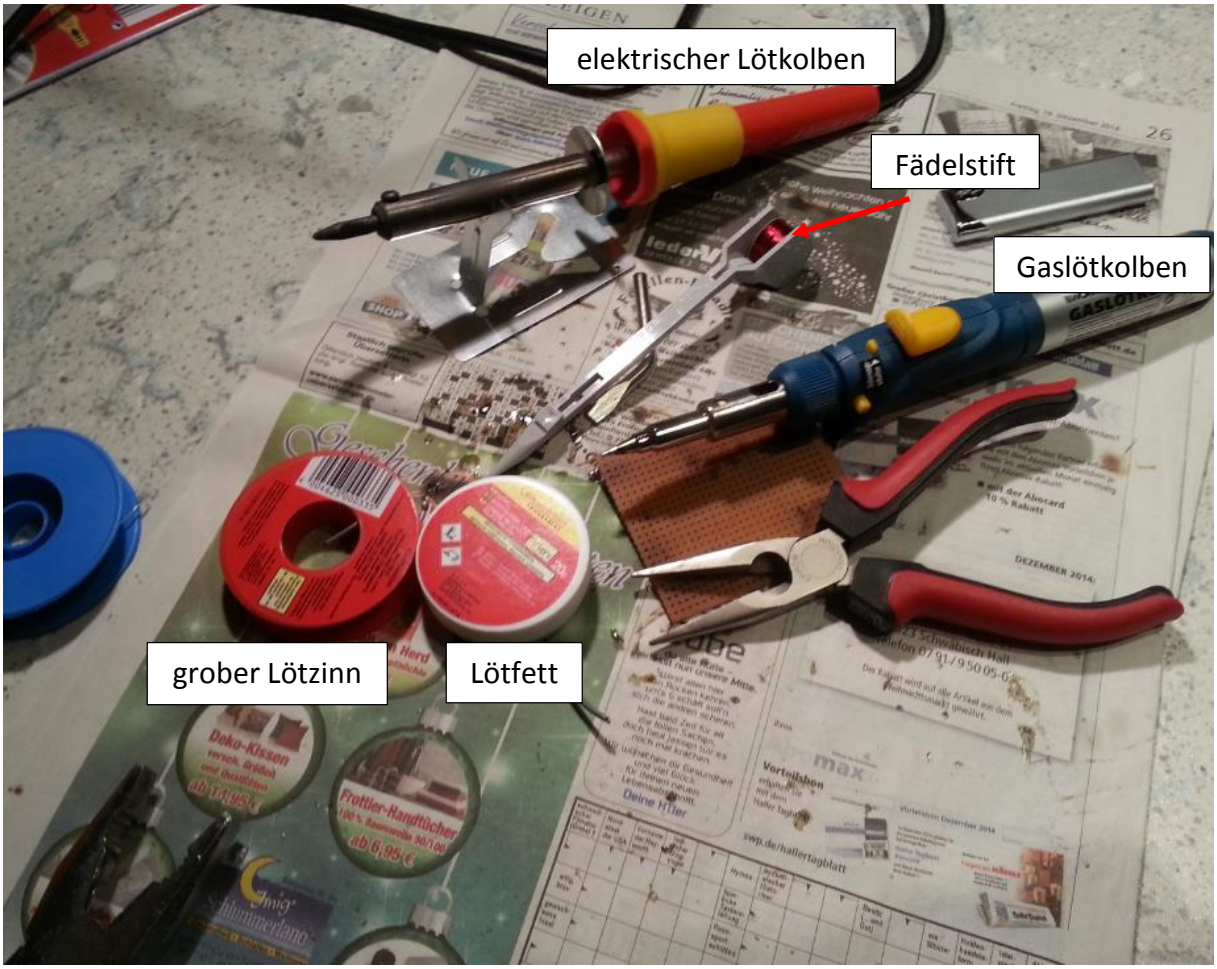


4.3 Vorgehensweise bei der Installation der Elektronik:

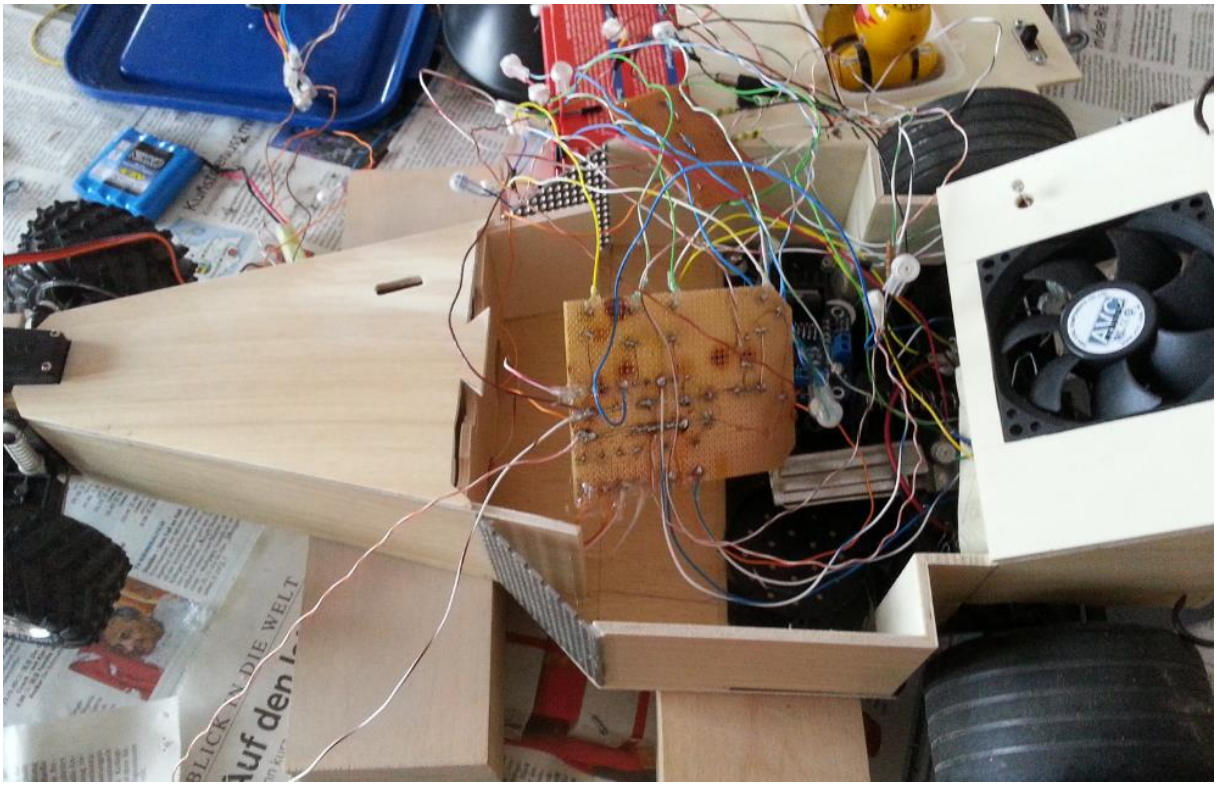
Zuerst habe ich meine 2 Platinen mit der Fritzing Software im PC virtuell erstellt, um mir eine Vorlage bzw. eine Planung zu verschaffen. Danach habe ich die Lochrasterplatten mit meinen Bauteilen bestückt und diese passend verlötet.

Hier gab es jedoch zuerst Probleme, da ich nur einen Lötkolben mit sehr dicker Lötspitze zur Verfügung hatte. Mit diesem elektrischen Lötkolben war die feine Lötarbeit sehr mühsam, da immer zu große Tropfen des Lötzinns auf die Lochrasterplatine gelangten und man mit der Lötspitze das Lötzinn schlecht an der Lötstelle platzieren konnte.
Für die weiteren Lötarbeiten musste also ein feinerer Lötkolben verwendet werden. Ich beschaffte mir daher einen Gaslötkolben mit einer feinen/dünnen Lötspitze. Mit diesem Gaslötkolben und Lötfett, war es nun deutlich einfacher das Lötzinn genauer und passender dosiert an die gewünschte Lötstelle anzubringen. Ich konnte nun eine geeignete Platine erstellen.

Um einzelne Lötstellen auf der Lochrasterplatte miteinander zu verbinden, habe ich einen Fädelstift verwendet. Mit diesem kann man den aufgewickelten Draht gut an der einen Lötstellen anlöten, zum nächsten Lötkontakt ziehen und auch dort nach kurzem Verzinnen des Kabels anlöten.



Am Schluss der Elektronikinstallation habe ich die Platinen mit den Spannungsquellen, dem Funduino und den einzelnen Bauteilen verkabelt. Die einzelnen Litzen habe ich aus einem alten WLan-Kabel entnommen.
Die Kabellängen mussten nun noch auf eine passende Länge erweitert oder gekürzt und an das Auto angepasst werden. Danach habe ich diese Kabel je mit einer Kabelhälfte, der im Funduino Kit enthaltenen Kabel, verbunden. Dadurch haben meine Kabel einen passenden Aufsatz für die Funduino Pins und können mit diesen verbunden werden. Das Verbinden dieser Kabeln habe ich auf zwei unterschiedliche Wege gelöst: durch Zusammenlöten bzw. durch Einsatz eines EVK2 Kabelverbinders.
Nun mussten man nur noch nicht isolierte Lötstellen mittels Heißkleber oder Isolierband isolieren und schließlich die fertig verkabelten Bauteile in den Hohlraum des Autos einsetzen. Was hier jedoch so einfach klingt, war bei den vielen Kabel und Anschlüssen gar nicht so einfach zu erledigen, da leicht der Überblick verloren geht. Um den Überblick doch länger zu behalten, habe ich die einzelnen Litzen mit Heißkleber an der Karosserie fixiert und versucht eine gewisse Ordnung in die Elektroinstallation meines Arduino Car zu bringen.

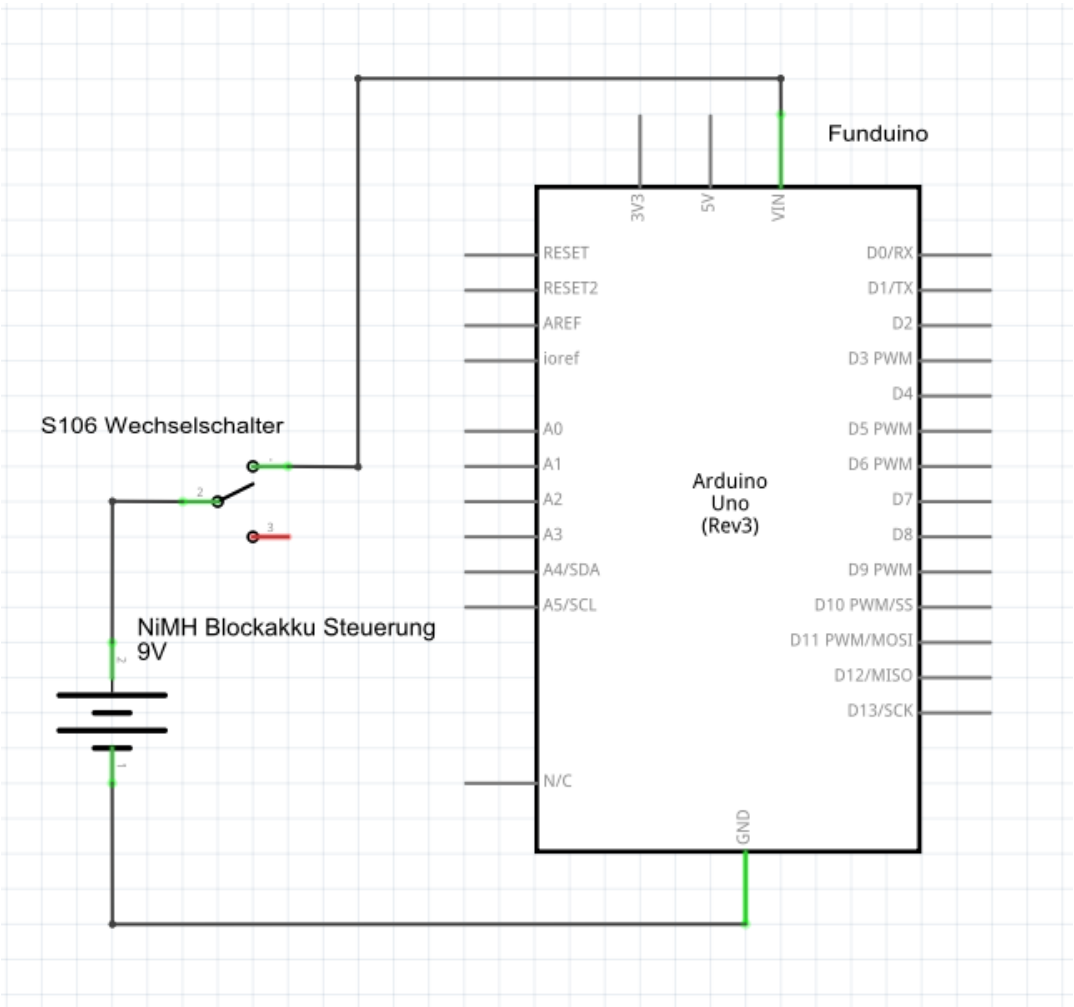


Abschluss der Elektroinstallation

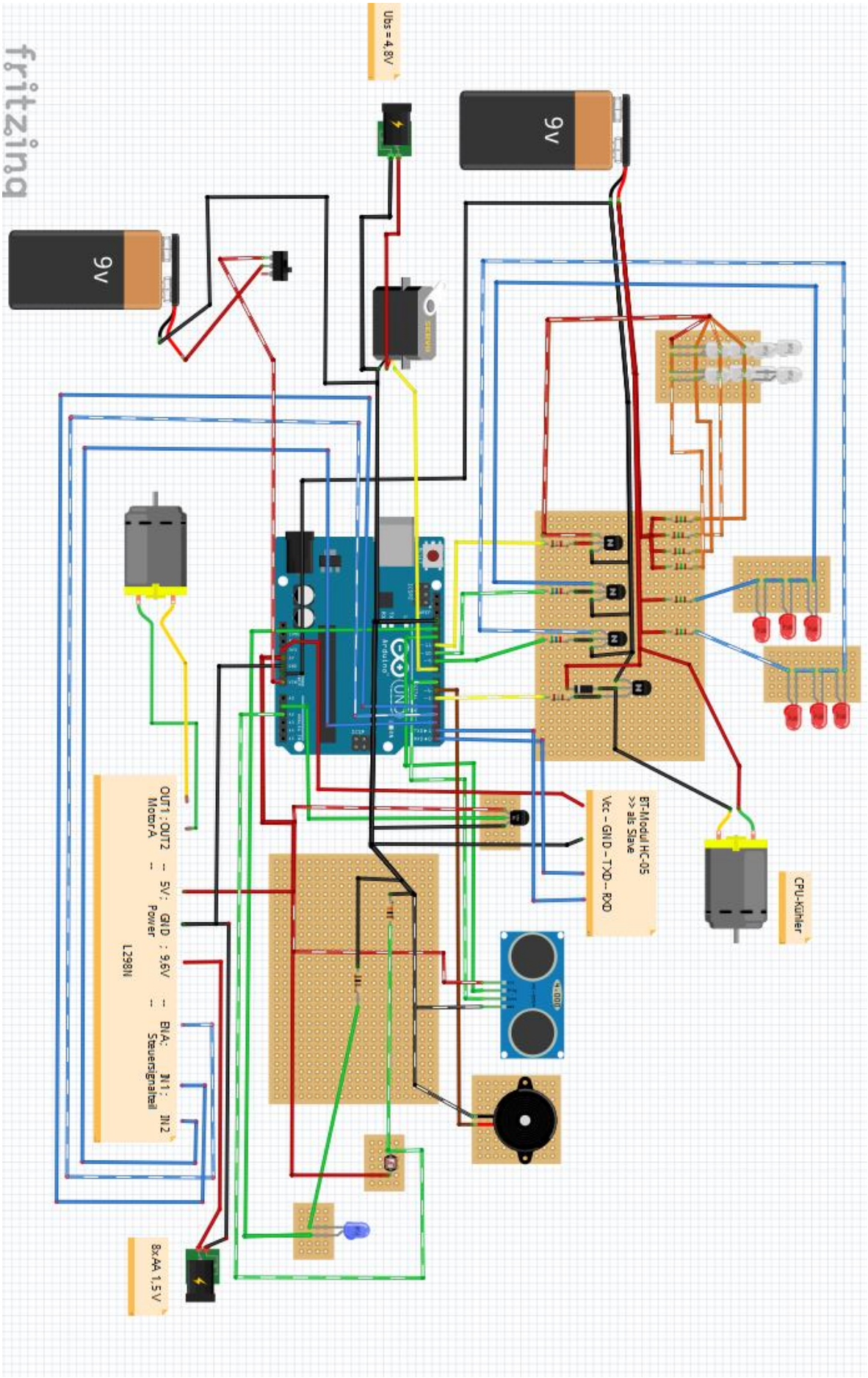
Zum Abschluss meines CT-Projektes habe ich einen Schiebeschalter und ein kurzes USB Kabel in meine Karosserie eingebaut. Das USB Kabel ermöglicht das Programmieren des Funduino von außen.

Als An-/Aus Schalter verwende ich den Schiebeschalter S106. Er besitzt einen Wechsler Kontakt mit 3 Anschlüssen. Er wird wie im Schaltplan dargestellt, in die Batterieklemme für den 9V NiMH Block Akku integriert. Da ich nur 2 Pins des Schiebeschalters verwende, verhält er sich wie eine An-/Aus Schalter. Mit ihm kann man die Betriebsspannung für den Funduino ein- und ausschalten und dadurch alle Arbeitsprozesse des Funduino Car starten oder beenden.

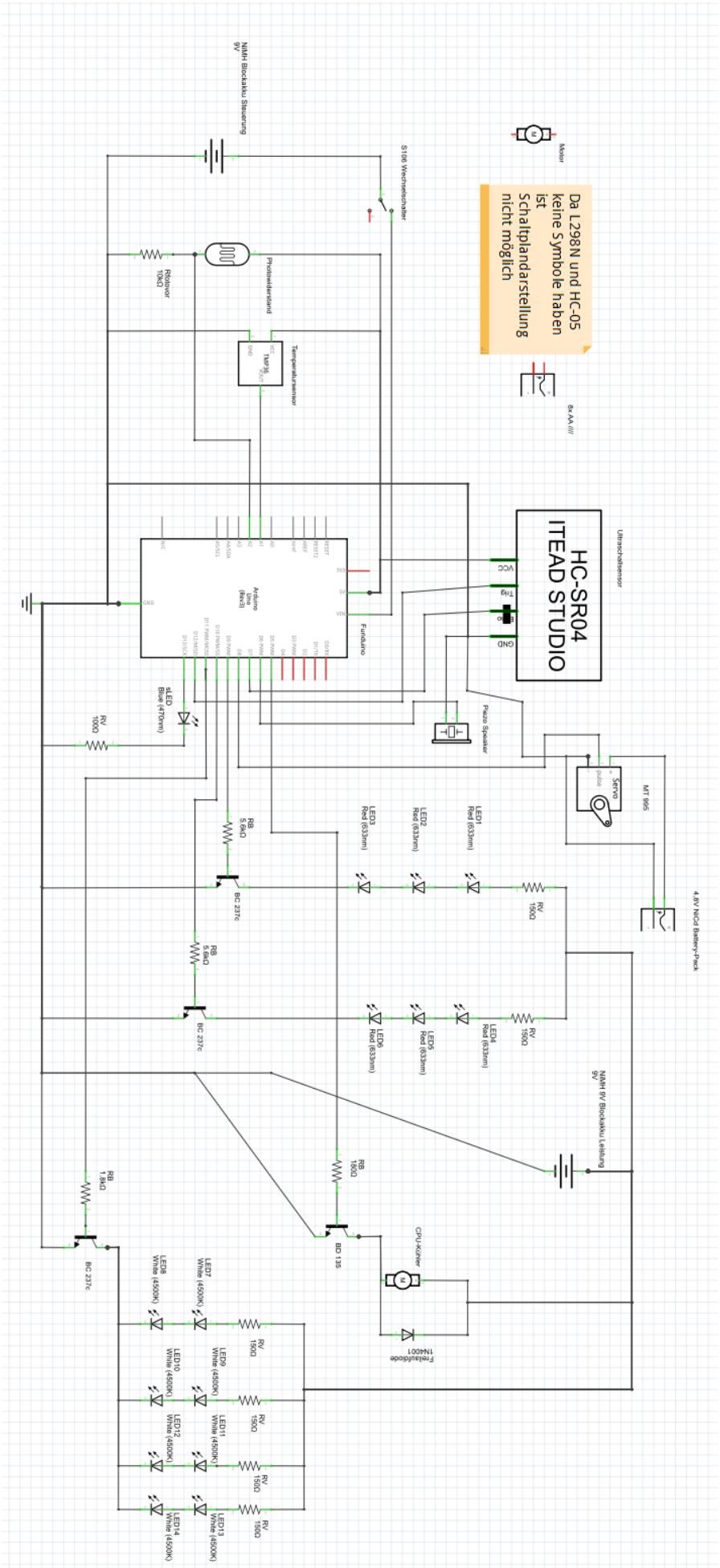
Schaltplan des An-/Aus Schalters:



Anschlussplan für die gesamte Elektronik:



Schaltplan der gesamten Elektronik:



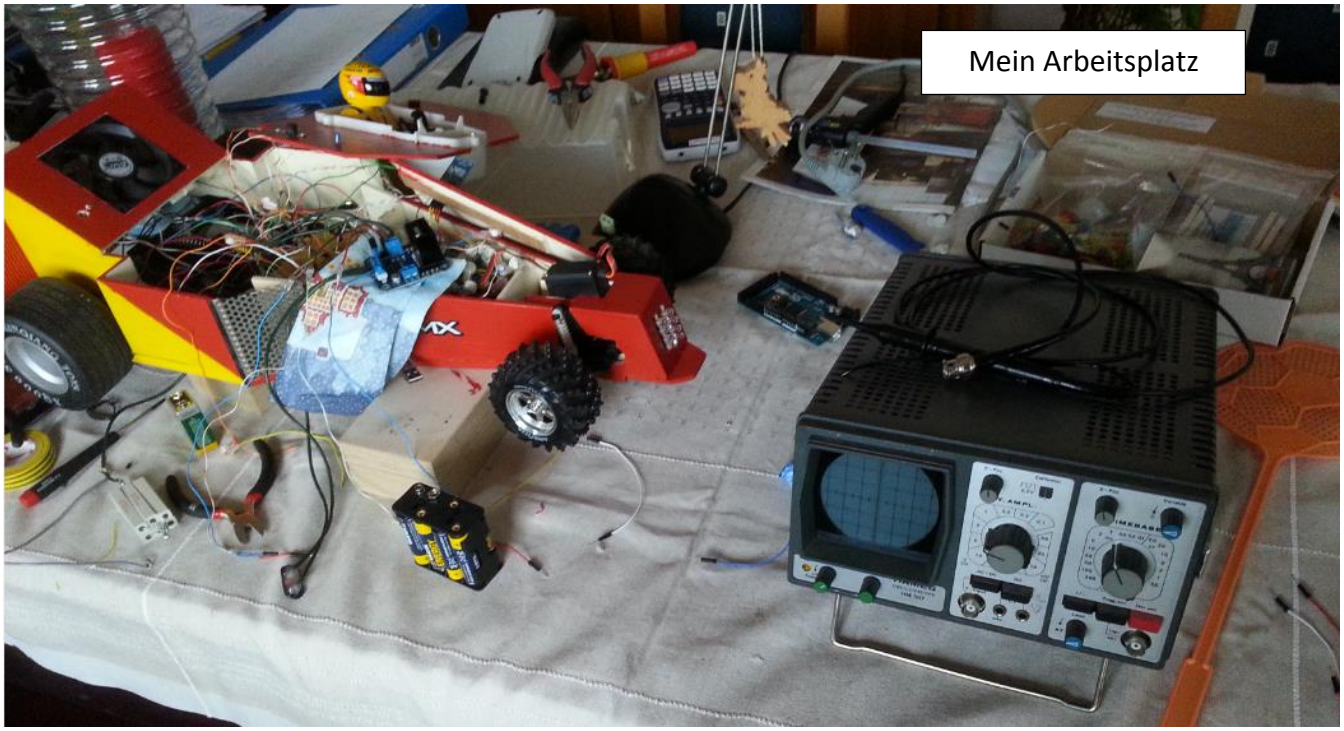
V. Fazit

Ich bin mit dem Ergebnis meines Projektes sehr zufrieden, da ich am Ende selbst ein ferngesteuertes Auto entworfen und gebaut habe.

Am Anfang hatte ich etwas Probleme mit der mir zuvor unbekannten Arduino Software, aber nach einigen geschriebenen Testprogrammen und vielen Versuchen mit dem Funduino, habe ich mich immer mehr an das Programm und den Mikrokontroller gewöhnt. Ich wurde immer sicherer im Umgang mit dem Funduino und habe viele mögliche Funktionen entdeckt. Dies war wichtig um eine Vorstellung davon zu bekommen, wie ich die gewünschten Funktionen meins Arduino Car am einfachsten realisieren und umsetzen kann. Letztendlich habe ich somit mein Programm immer erweitert und verbessert und bin zu einem Ergebnis gekommen, mit dem ich am Projektanfang nicht gerechnet habe.

Ich war auch überrascht darüber, wie viele kleine Probleme und unvorhersehbare Arbeitsschritte so ein Projekt versteckt hält. Diese traten vor allem bei der Umsetzung des geplanten, theoretischen Teils in die Praxis auf. Wenn man sich jedoch genau mit diesen Problemen befasst, findet man immer eine Lösung und man macht Schritt für Schritt Vorschritte.

Zum Abschluss des Projekts kann ich festhalten, dass ich im Projektverlauf immer mehr Spaß an meinem Projekt bekommen und es gerne gemacht habe. Die Zeit, die ich in mein Arduino Car investiert habe, hat sich also gelohnt.


Mein Arbeitsplatz

5.1 Verbesserungsmöglichkeiten

In meinem Projekt gibt es noch Dinge, in denen ich Verbesserungspotenzial sehe. Diese Verbesserungsmöglichkeiten ergeben sich meist daraus, dass für die optimale Lösung der zeitliche Rahmen des Projektes zu klein ist, oder andere Bauteile zur Verbesserung benötigt werden.

1. Lenkeinschlag

Dadurch, dass der Lenkeinschlag der übernommenen Vorderachse nicht sehr groß ist, benötigt das Arduino Car große Radien um eine Kurve zu fahren. Der Lenkeinschlag sollte zur Verbesserung vergrößert werden, aber dazu bräuchte ich eine komplett neue Vorderachse mit neuer Lenkachse. Da mein ganzes Projekt und die ganze Karosserie jedoch auf diese Vorderachse angepasst ist, müsste ich bei einem Umtausch den ganzen Autoaufbau neu planen. Dies ist aus zeitlichen Gründen unmöglich.

2. Spannungsversorgung

Optimieren könnte man meiner Meinung nach auch die Spannungsversorgung, indem man richtige, fertig gekaufte Akkus mit höherer Nennkapazität verwenden würde. Da ich jedoch für mein Projekt viele Bauteile aus anderen alten und defekten ferngesteuerten Autos recyceln möchte, habe ich dies nicht gemacht. Außerdem wollte ich im Rahmen dieses Projektes einmal ausprobieren, eine Spannungsversorgung, welche den Anforderungen der Bauteile im Arduino Car gerecht werden, selbst zu planen.
Das größte Defizit sehe ich in der Spannungsquelle für den Motor, welche ja schon viele Probleme bereitet hat (siehe Seite 83). Hier muss noch eine Lösung gefunden werden, um den Motor ausreichend stark, mit einer aufladbaren Spannungsquelle betreiben zu können. Als Lösung könnte ich mir LiPo-Akkumulatoren vorstellen, dies möchte ich nachtragend wenn möglich testen.

3. Bluetooth Reichweite

Mit einem anderen Bluetooth Modul könnte man möglicherweise die Reichweite der Steuerung erhöhen.

4. Löten

In diesem Projekt habe ich das erste Mal versucht Schaltungen auf Lochrasterplatten zu löten. Mit einem Lötkolben mit sehr dicker Lötspitze und ohne Lötfett war dies anfangs sehr mühsam, da immer zu große Tropfen des Lötzinns auf die Lochrasterplatine gelangten. Als ich später einen Gaslötkolben mit dünner Lötspitze mit Lötfett benutzt habe, konnte ich das

Lötzinn deutlich genauer und passender dosiert an die gewünschte Lötstelle anbringen.
Mein Verbesserungsvorschlag: Das nächste Mal von Anfang an diesen Lötkolben inklusive Lötfett verwenden!

5. Funduino MEGA

Bei der Erstellung meines Programms, bin ich immer öfter auf das Problem gestoßen, dass ich bestimmte zeitliche Abläufe regeln musste. Ein Beispiel hierfür ist das Blinken der Signal LED oder der Rücklichter und die wiederkehrende Temperaturermittlung alle 4 Sekunden.

Zuerst wollte ich diese zeitlichen Abläufe durch Interrupt-Timer Libraries kontrollieren, dies stellte sich jedoch als Problem heraus. Immer wenn ich eine Interrupt-Timer Library verwendet habe, der den Timer 0, Timer 1 oder Timer 2 des Funduino benutzt, kam es zu Fehlfunktionen in anderen Programmteilen. Ich habe mich daraufhin im Internet über die Timer des Funduino UNO informiert und dabei herausgefunden, dass der Funduino UNO nur die oben genannten 3 Timer besitzt. Jedem dieser Timer ist eine Aufgabe im Funduino zugeordnet. Wird nun durch eine Library dieser Timer verändert, kann es zu Fehlern in seinem ursprünglichen Aufgabenbereich kommen.

→ siehe Informationsmaterial Seite 88: „Never touch timer0“

Dies war auch der Grund, warum mein Servo nicht mehr die gewünschte Aktion ausführte. Die Interrupt-Timer Library arbeitete nämlich mit Timer1 und dadurch wurde die Servo-Library gestört.

Letztendlich habe ich die zeitlichen Abläufe im Programm auch ohne Interrupt-Timer realisieren können (siehe Programmierung der Signal LED Seite 38), doch ich würde trotzdem für ein folgendes Projekt den größeren Funduino MEGA bevorzugen. Dieser hat mehr Pins und mehr interne Timer, was einem mehr Möglichkeiten der Programmierung eröffnet.

6. Optimale Widerstände

Da Aufgrund der notwendigen Änderung der Spannungsquelle die Betriebsspannung mancher Schaltungen des Projekts geändert wurde, sind einige Widerstände nicht mehr optimal auf diese neue Spannung angepasst. Der Grund dafür ist, dass diese mit der alten Betriebsspannung 9,6V berechnet wurden.
Letztendlich habe ich diese Widerstände nicht ersetzt, da ich die optimalen Widerstände für die neue Betriebsspannung U_{bL} = 9V nicht zur Verfügung hatte und sich mit den alten Widerständen keine Beschränkung der Funktion von den Schaltungen ergab.

VI. Informationsmaterial

Im Folgenden befinden sich die Internettextausschnitte, welche mir besonders bei meinem Halbjahresprojekt geholfen haben:

Zusatzinformation über die Timer des Arduino:

- Straub, Michael: Never touch timer0; URL: http://www.micha.st/?Arduino-Notizbuch:Erfahrungen:Never_touch_timer0 (eingesehen am 19.12.2014).

Never touch timer0

Der AVR des Arduino Uno hat drei Timer. Man kann die Timer programmieren, im Web gibt es viele Beispiele. Die beziehen sich alle auf Timer 1 und Timer 2. Die Timer werden u.a. für die PWM-Ausgänge benutzt. Es gibt hier eine feste Zuordung:

Timer	Port
0	5 und 6
1	9 und 10
2	3 und 11

Mir war klar, dass ich das PWM-Verhalten von Port 5 und 6 verändere, wenn ich an Timer 0 herumfummele. Wenn man 5 und 6 als normale Ein- oder Ausgänge nutzt, ist das erstmal nicht schlimm - dachte ich. Weit gefehlt.

Das Arduino-Monitorprogramm benötigt einen Systemtakt, genau dafür wird Timer 0 offenbar genutzt. Wenn man daran irgendwas verstellt oder gar Softwareinterrupts zuordnet, bringt man das ganze System durcheinander. Es gibt merkwürdige Effekte. Ich kann nur davor warnen, Timer 0 anzufassen. Das geht nicht gut.

Auf der anderen Seite heißt das, wenn PWM, dann bitte auf jeden Fall vorzugsweise über die Ports 5 und 6, weil man am Timer ja eh' nicht drehen sollte. Die beiden anderen Timer sind offenbar harmlos.

Es empfiiehlt sich nicht, groß selbst an den Timern zu manipulieren. Dafür gibt es fertige Bibliotheken. Mit gutem Erfolg habe ich **TimerOne** ausprobiert, die ich hier auch empfehle.

Der Autor dieses Textes ist Michael Straub.

- RobotFreak: Arduino 101: Timers and Interrupts. Aus: letsmakerobots; URL: http://letsmakerobots.com/node/28278 (eingesehen am 19.12.2014).

[.....]

Timer0:
Timer0 is a 8bit timer.
In the Arduino world timer0 is been used for the timer functions, like delay(), millis() and micros(). If you change timer0 registers, this may influence the Arduino timer function. So you should know what you are doing.

Timer1:
Timer1 is a 16bit timer.
In the Arduino world the Servo library uses timer1 on Arduino Uno (timer5 on Arduino Mega).

Timer2:
Timer2 is a 8bit timer like timer0.
In the Arduino work the tone() function uses timer2.

Timer3, Timer4, Timer5:
Timer 3,4,5 are only available on Arduino Mega boards. These timers are all 16bit timers.

[.....]

The Arduino has 3Timers and 6 PWM output pins. The relation between timers and PWM outputs is:
Pins 5 and 6: controlled by timer0
Pins 9 and 10: controlled by timer1
Pins 11 and 3: controlled by timer2

Der Autor dieses Textes ist RobotFreak.

Zusatzinformation über die Variablentypen der Arduino Software:

- B_E_N: Data Types in Arduino. Aus: sparkfun; URL: https://learn.sparkfun.com/tutorials/data-types-in-arduino (eingesehen am 19.12.2014).

Defining Data Types

The Arduino environment is really just C++ with library support and built-in assumptions about the target environment to simplify the coding process. C++ defines a number of different data types; here we'll talk only about those used in Arduino with an emphasis on traps awaiting the unwary Arduino programmer.

Below is a list of the data types commonly seen in Arduino, with the memory size of each in parentheses after the type name. Note: **signed** variables allow both positive and negative numbers, while **unsigned** variables allow only positive values.

- **boolean** (8 bit) - simple logical true/false
- **byte** (8 bit) - unsigned number from 0-255
- **char** (8 bit) - signed number from -128 to 127. The compiler will attempt to interpret this data type as a character in some circumstances, which may yield unexpected results
- **unsigned char** (8 bit) - same as 'byte'; if this is what you're after, you should use 'byte' instead, for reasons of clarity
- **word** (16 bit) - unsigned number from 0-65535
- **unsigned int** (16 bit)- the same as 'word'. Use 'word' instead for clarity and brevity
- **int** (16 bit) - signed number from -32768 to 32767. This is most commonly what you see used for general purpose variables in Arduino example code provided with the IDE
- **unsigned long** (32 bit) - unsigned number from 0-4,294,967,295. The most common usage of this is to store the result of the `millis()` function, which returns the number of milliseconds the current code has been running
- **long** (32 bit) - signed number from -2,147,483,648 to 2,147,483,647
- **float** (32 bit) - signed number from -3.4028235E38 to 3.4028235E38. Floating point on the Arduino is not native; the compiler has to jump through hoops to make it work. If you can avoid it, you should. We'll touch on this later.

Der Autor dieses Textes ist B_E_N.
Dieser Text ist auf der Website von SparkFun veröffentlicht.

Zusatzinformation über Befehle der Arduino Software:

- Kainka, Fabian: Befehlsliste Arduino (2012); URL: http://www.fkainka.de/Arduino/Befehlsliste.html (eingesehen am 19.12.2014).

  Auf dieser Internetseite befindet sich eine Auflistung der grundlegenden, wichtigen Befehle für den Arduino bzw. Funduino die ich verwendet habe. Der Ersteller dieser Auflistung ist Fabian Kainka.

Befehl	Schreibweise
Grundlegendes	
Grundstruktur	*void setup() { }* *void loop() { }*
Variable deklarieren (Typ Integer, Wert = 0)	*int led = 0;*
Konstante deklarieren (Typ Integer, Wert = A0)	*const int analogInPin = A0;*
Array deklarieren (Typ Integer, 6 Werte)	*int Arrayname[6] = {2, 4, -8, 3, 2};*
Pin-Funktion festlegen	*pinMode(led, OUTPUT);*
Serielle Schnittstelle initialisieren	*Serial.begin(9600);*
Library einbinden	*#include <libX.h>*

[……]

Ausgaben/Eigaben	
Pin auf HIGH schalten	*digitalWrite(led,HIGH);*
PWM-Pin auf maximal Wert setzen	*analogWrite(led, 255);*
Digitalen Zustand von Pin *schalter* lesen	*digitalRead(schalter)*
Funktion: Analog Pin A0 auslesen	*analogRead(A0)*
Text seriellen Ausgeben	*Serial.print("Hallo");*
Text seriellen Ausgeben + neue Zeile	*Serial.println("Hallo");*
Funktion: seriell Einlesen	*Serial.read();*
Ton Ausgabe an Pin8, Frequenz x, länge y	*tone(8, x, y)*
Ton Ausgabe an Pin8 stoppen	*noTone(8)*

[……]

Funktion: Wertebereich transformieren (z.B. 10bit Sensor auf 8bit PWM)	map(sensorWert, 0, 1024, 0, 255);

[……]

Abfragen/Schleifen	
If-Abfrage mit Else	*if (button == HIGH){ }* *Else{ }*
Switch-Abfrage	*switch (x) {* *case 1:* *case 2:* *default:*
Wartezeit (500 ms)	*delay(500);*
For-Schleife	*for (x = 2; x < 7; x++) { }*
While-Schleife	*while (schalter== HIGH) {* *}* **ODER:** *do{* *} while (schalter==HIGH);*
Schleife vorzeitig verlassen	*break;*

[……]

Vergleiche	
gleich	==
ungleich	*!=*
größer, kleiner	<, >
Größer oder gleich, kleiner oder gleich	<= , >=
Funktion: kleinere Zahl von x und y	*min(x, y)*
Funktion: größere Zahl von x und y	*max(x, y)*
Boolsche Operatoren	
Und	*&&*
Oder	||
Nicht	*!*

VII. Quellenangabe

Verwendete Icons für das Layout der ArduinoCar_Controller_BT1 App:

- Base, Designerz: „FatiCons by Designerz Base”; URL: https://www.iconfinder.com/icons/186407/arrow%2Cup%2Carrow_up_icon#size=128 (eingesehen am 16.10.2014).
- Base, Designerz: „FatiCons by Designerz Base”; URL: https://www.iconfinder.com/icons/186411/arrow%2Cdown_icon#size=128 (eingesehen am 16.10.2014).
- Caesar, Jim: „ Car silhouettes by Jim Ceasar “; URL: https://www.iconfinder.com/icons/285811/auto_automobile_car_sportcar_vehicle_icon#size=128 (eingesehen am 16.10.2014).

HC-SR 04 Grafik :

- damibob, (2014). Arduino ultrasonic distance measurement. Aus: fritzing.org. Download am 17.12.2014 von http://fritzing.org/projects/arduino-ultrasonic-distance-measurement ;
  dazugehörige Website des Autors: URL: http://damibob.blogspot.de/2013/09/arduino-ultrasonic-distance-measurement.html (eigesehen am 30.12.2014).

„NewPing-Library“ :

- Eckel, Tim, (2012). NewPing. Libary for Arduino. Aus: arduino.cc. Download am 14.11.2014 von http://playground.arduino.cc/Code/NewPing#Download

Datasheets:

- Fairchild Semiconductor Corporation (Hg.): BD135 / 137 / 139. NPN Epitaxial Silicon Transistor; URL: https://www.fairchildsemi.com/datasheets/BD/BD135.pdf (eingesehen am 19.12.2014).
- Fairchild Semiconductor International (Hg.): BC237/238/239; URL: http://www.pisotones.com/BigMuffPi/imgs/BC237-8-9.pdf (eingesehen am 19.12.2014).
- STMicroelectronics (Hg.): L298. DUAL FULL-BRIDGE DRIVER. Aus: sparkfun; URL: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf (eingesehen am 05.01.2015).

Informationsmaterial:

Literatur:

- Bartmann, Erik: Die elektronische Welt mit Arduino entdecken. 1.Aufl., Köln 2011, Seite 613 - 619.

Internetquellen:

- Arduino: Language Reference. Aus: arduino.cc; URL: http://arduino.cc/en/pmwiki.php?n=Reference/HomePage (eingesehen am 19.12.2014).
- B_E_N: Data Types in Arduino. Aus: sparkfun; URL: https://learn.sparkfun.com/tutorials/data-types-in-arduino (eingesehen am 19.12.2014).
- Kainka, Fabian: Befehlsliste Arduino (2012); URL: http://www.fkainka.de/Arduino/Befehlsliste.html (eingesehen am 19.12.2014).
- RobotFreak: Arduino 101: Timers and Interrupts. Aus: letsmakerobots; URL: http://letsmakerobots.com/node/28278 (eingesehen am 19.12.2014).
- Straub, Michael: Never touch timer0; URL: http://www.micha.st/?Arduino-Notizbuch:Erfahrungen:Never_touch_timer0 (eingesehen am 19.12.2014).