

DIPLOMARBEIT

Hovercraft

Ausgeführt im Schuljahr 2020/21 von:

Stefan Deimel
Philipp Eilmsteiner
Julia Katharina Stöger

Betreuer/Betreuerin:

Dipl.-Ing. Helge Frank

St. Pölten, am 2. April 2021

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.



Stefan Deimel



Philipp Eilmsteiner



Julia Stöger

Gendererklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Diplomandenvorstellung



Stefan Deimel

GEBURTSDATEN:
26.03.2002 in Tulln

WOHNHAFT IN:
Josef-Mohnl-Gasse 8
3435 Zwentendorf

BERUFLICHER WERDEGANG:
2016–2021:
HTBLuVA St.Pölten, Abteilung für Elektrotechnik
2012–2016:
Gymnasium Tulln

KONTAKT:
stefan.deimel@aon.at



Philipp Eilmsteiner

GEBURTSDATEN:
24.03.2002 in Scheibbs

WOHNHAFT IN:
Josef-Pfeiffer Straße 3
3250 Wieselburg

BERUFLICHER WERDEGANG:
2016–2021:
HTBLuVA St.Pölten, Abteilung für Elektrotechnik
2012–2016:
Gymnasium Wieselburg

KONTAKT:
philipp.eilmsteiner@outlook.at



Julia Katharina Stöger

GEBURTSDATEN:
07.06.2002 in Tulln

WOHNHAFT IN:
Johannesgasse 15
3435 Zwentendorf

BERUFLICHER WERDEGANG:
2016–2021:
HTBLuVA St.Pölten, Abteilung für Elektrotechnik
2012–2016:
Hauptschule Zwentendorf

KONTAKT:
juliastoeger02@gmx.at

Danksagungen

An dieser Stelle möchten wir uns bei all jenen bedanken, die uns bei unserer Diplomarbeit tatkräftig unterstützt und motiviert haben.

Zuerst bedanken wir uns recht herzlich bei Herrn Dipl.-Ing. Helge Frank, der uns während dieser Arbeit mit seinem Fachwissen unterstützt und weitergeholfen hat. Durch sein besonderes Interesse an dem Projekt gelang es ihm, sein Wissen sehr gut an uns zu übermitteln.

Weiters möchten wir uns bei Herrn Fachlehrer Meiseneder und bei Herrn Fachlehrer Binder für die großartige Unterstützung bei fehlendem Werkzeug und Fragen rund um das Thema Elektronik bedanken. Außerdem wollen wir auch allen anderen Lehrerinnen und Lehrer danken, die uns mit ihrem Wissen weitergeholfen haben.

Spezieller Dank gilt auch unseren Familien, die uns stets motiviert haben und mental zur Seite gestanden sind.

Dank gebührt auch Herrn Ing. Andreas Bergen für seine tatkräftige Unterstützung und Ideen zu unserer Diplomarbeit.

Für die sehr gute Zusammenarbeit und die finanzielle Unterstützung bedanken wir uns bei all unseren Sponsoren: Schrack for Students, Schmied & Fellmann, Kustec, EMC und Gottwald.

**DIPLOMARBEIT
DOKUMENTATION**

Namen der Verfasser/innen	Stefan Deimel Philipp Eilmsteiner Julia Stöger
Jahrgang Schuljahr	5BHET 2020/21
Thema der Diplomarbeit	Hovercraft
Kooperationspartner	

Aufgabenstellung	Die Aufgabe der Diplomarbeit ist der Bau eines elektrischen Luftkissenbootes aus einem Schlauchboot. Dieses soll einen erwachsenen Menschen über Land und über Wasser transportieren können.
------------------	--

Realisierung	Der Aufbau ist als Leichtbaukonstruktion über einem Schlauchboot realisiert. Der zum Schweben benötigte Luftstrom wird mit einem 10kW Elektromotor und einem 6-Blatt Propeller erzeugt. Der für den Vortrieb benötigte Luftstrom wird mit einem weiteren 10kW Elektromotor realisiert, zur Lenkung werden mit Servos gesteuerte Fahnen verwendet. Der Fahrer kann das Hovercraft mit einem Lenker und zwei Daumengashebel steuern. Zum Laden der Akkus muss das Ladegerät an den vorbereiteten Kabeln angeschlossen werden.
--------------	--

Ergebnisse	Zum Zeitpunkt der Abgabe war das Hovercraft fertig konstruiert, die Elektronik war eingebaut und getestet. Das Luftkissenboot war somit einsatzbereit und es wurden bereits Probefahrten absolviert.
------------	--

Typische Grafik, Foto etc. (mit Erläuterung)	 
---	---

Teilnahme an Wettbewerben, Auszeichnungen	
--	--

Möglichkeiten der Einsichtnahme in die Arbeit	HTL St. Pölten
--	----------------

Approbation (Datum / Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin Abteilungsvorstand/Abteilungsvorständin
---------------------------------------	-----------------	--

DIPLOMA THESIS

Documentation

Author(s)	Stefan Deimel Philipp Eilmsteiner Julia Stöger
Form Academic year	5BHET 2020/21
Topic	Hovercraft
Co-operation partners	
Assignment of tasks	The aim of this diploma thesis is the construction of an electric hovercraft. The base for the construction is a rubber dinghy. The hovercraft should be able to transport an adult above land and water.
Realisation	The structure is implemented as a lightweight construction on top of a rubber dinghy. The airflow required for hovering is generated by a 10kW electric motor attached to a 6-blade propeller. The second propeller points to the back and therefore produces forward thrust. In order to steer the hovercraft, there are three servo-controlled fins behind the propeller to redirect the airflow. The driver is able to steer the vehicle with a handlebar and two thumb throttles. To charge the batteries, the charger must be connected to the prepared cables.
Results	At the time of delivery, the construction of the hovercraft is completed, and the electronic system implemented and tested. Therefore, the hovercraft is ready to use, and the first test runs have been done.

Illustrative graph, photo (incl. explanation)	
--	---

Participation in competitions Awards	
---	--

Accessibility of diploma thesis	HTL St. Pölten
------------------------------------	----------------

Approval (date / signature)	Examiner	Head of College / Department
--------------------------------	----------	------------------------------

Abstract

The aim of our diploma thesis was to build a functional electrical hovercraft, which is able to move over land as well as water. An electrical motor attached to a propeller facing downwards creates an airflow on which the boot hovers. The second propeller points to the back and therefore produces forward thrust. In order to steer the hovercraft, there are three servo-controlled fins behind the propeller to redirect the airflow. The driver is able to steer the vehicle with a handlebar and two thumb throttles.

Zusammenfassung

Das Ziel unserer Diplomarbeit ist es, aus einem Schlauchboot ein funktionstüchtiges elektrisches Luftkissenboot zu bauen, welches sich sowohl am Land als auch über Wasser fortbewegen kann.

Mit einem von einem Elektromotor angetriebenen Propeller wird ein nach unten gerichteter Luftstrom erzeugt, auf welchem das Boot schwebt. Der zweite Propeller zeigt nach hinten und ist somit für den Vortrieb verantwortlich. Um das Boot lenkbar zu machen, sind drei servo-gesteuerte Fäden zum Umlenken des Luftstroms am Heck montiert.

Der Fahrer ist in der Lage, das Luftkissenboot mit einem Lenker und zwei Daumengashebel zu steuern.

Inhaltsverzeichnis

Diplomandenvorstellung	iv
1 Themenvorstellung	1
1.1 Ausgangslage	1
1.2 Zielsetzung	1
1.3 Individuelle Themenstellung	1
2 Funktionsprinzip	2
3 Erster Prototyp	3
4 Konstruktion	4
4.1 Bodenplatte	5
4.1.1 Stückliste	6
4.1.2 Inventor	7
4.1.3 Gitter	16
4.1.4 Zusammenbau	17
4.1.5 Montage am Boot	19
4.2 Hinterer Aufbau	20
4.2.1 Stückliste	21
4.2.2 Inventor	22
4.2.3 Zusammenbau	46
4.3 Lenkung	47
4.3.1 Stückliste	48
4.3.2 Inventor	49
4.3.3 Zusammenbau	63
4.4 Bestellliste	65
4.5 Gesamter Zusammenbau	66
5 Leistungselektronik	67
5.1 Antrieb	67
5.1.1 Motoren	67
5.1.2 Propeller	68
5.1.3 Motorregler	70
5.2 Servos	72
5.3 Schaltplan	73
5.4 Verkabelung	74
5.5 Hauptrelais	75
5.6 Schmelzsicherungen	75

5.7	Buck-Converter	76
5.8	Schaltrelais für Buck-Converter	77
5.9	Akkus	78
5.10	Ladeelektronik	79
5.10.1	Ladegerät	79
5.10.2	Schaltnetzteil	80
5.11	Stückliste	81
6	Boardelektronik	82
6.1	Übertragungsmedium CAN-Bus	82
6.1.1	Gründe für die Verwendung	82
6.1.2	Verwendete Adressen	83
6.2	Zur Programmierung verwendete Software	83
6.2.1	Visual Studio Code	84
6.2.2	PlatformIO IDE	85
6.3	Steuerungsplatinen	86
6.3.1	Schaltplan – Übersicht	86
6.3.2	Motorregleransteuerung	87
6.3.3	Lenker	92
6.3.4	Fahnenansteuerung & Akkutemperaturen	99
6.3.5	ADC-Platinen	104
6.3.6	Precharge & Relaisansteuerung	107
6.3.7	Platzierung am Boot	110
6.4	Bildschirm	111
6.4.1	Installation des Nextion-Editors	112
6.4.2	Auswahl des richtigen Bildschirmes	113
6.4.3	Entworfene Oberfläche	114
6.5	Stückliste	117
7	Sicherheitskonzept	118
7.1	Sicherheitsmaßnahmen	118
7.1.1	Not-Aus-Schalter mit Schlüssel	118
7.1.2	Daumengas für beide Propeller	119
7.1.3	Gitterabdeckungen	120
7.2	Gerätesicherheit	121
7.2.1	Ladevorgang	121
7.2.2	Bremsschutz	122
7.2.3	Reglerkühlung	122
8	Kostenrechnung	123
8.1	Konstruktion	123
8.2	Leistungselektronik	123

8.3 Boardelektronik	124
8.4 Gesamtkosten	124
9 Anleitungen	125
9.1 Betriebsanleitung	125
9.2 Anleitung Ladevorgang	126
10 Besprechungsprotokolle & Meilensteine	129
10.1 Besprechungsprotokolle	129
10.2 Meilensteine	129
11 Zeitaufzeichnung	131
11.1 Stefan Deimel	131
11.2 Philipp Eilmsteiner	134
11.3 Julia Stöger	137
12 Anhang	139
12.1 Arduino Bibliotheken	139
12.1.1 Jeti Decoder	139
12.2 Datenblätter	149
12.2.1 JetiBox	149
12.2.2 MasterSPIN 220 Pro OPTO	153
Abkürzungsverzeichnis	168
Abbildungsverzeichnis	169
Tabellenverzeichnis	172
Literaturverzeichnis	174

1 Themenvorstellung

1.1 Ausgangslage

Inspiriert von einem YouTube-Video^[1] soll ein voll funktionstüchtiges Luftkissenboot entwickelt werden, das in der Lage ist, eine erwachsene Person einige Minuten lang zu befördern. Als Antriebe sollen zwei Elektromotoren eingesetzt werden. Der Aufbau ist als Leichtbaukonstruktion über einem Schlauchboot zu realisieren.

1.2 Zielsetzung

Das Ziel der Diplomarbeit ist es, ein Schlauchboot zu einem Luftkissenboot umzubauen. Das Hovercraft soll mit zwei 10 kW Elektromotoren mit dazugehörigen Propellern konstruiert werden.

Das fertige Luftkissenboot soll in der Lage sein, eine erwachsene Person über Land sowie über Wasser zu transportieren.

1.3 Individuelle Themenstellung

- Stefan Deimel:
 - Konstruktion
 - Motorregelung
- Philipp Eilmsteiner:
 - Steuerelektronik
 - HMI
- Julia Stöger:
 - Sicherheitskonzept
 - Ladeelektronik

2 Funktionsprinzip

Die Grundlage für das Schweben des Luftkissenbootes ist der auf den Boden gerichtete Luftstrom. Die Luft wird vertikal nach unten geleitet um einen Luftpolster zu erzeugen. Durch ein Loch in der Bodenplatte des Hovercrafts kann die Luft entweichen und somit einen Druck erzeugen, um das Luftkissenboot zum Schweben zu bringen.

Mithilfe von Fahnen, welche hinter dem Propeller am Boot angebracht sind, lässt sich das Hovercraft lenken.

Für eine vereinfachte Darstellung der Funktionsweise siehe Abbildung 2.1. Die blauen Pfeile stellen den Luftstrom dar, der zuerst senkrecht auf den Boden gerichtet ist und danach während des Schwebezustandes unter dem Boot entweicht. Die horizontalen blauen Pfeile stellen den Luftstrom dar, der durch den hinteren Propeller und die Fahnen geleitet wird.

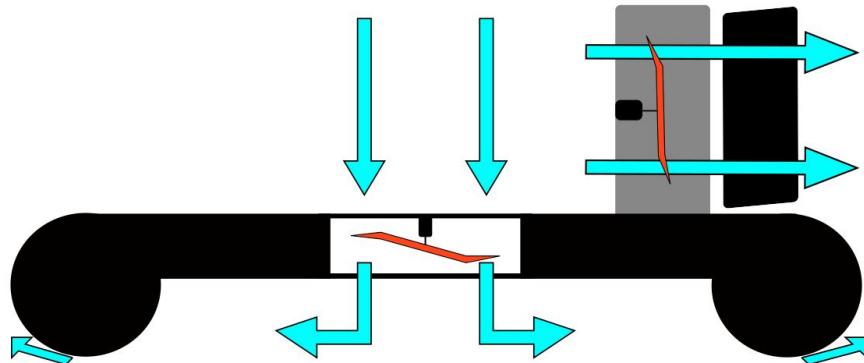


Abbildung 2.1: Funktionsprinzip eines Hovercrafts

3 Erster Prototyp

Um die Umsetzbarkeit dieses Projektes zu überprüfen, wurde ein Prototyp angefertigt.

Aus einem alten Schlauchboot wurde der Boden entfernt. Danach wurde aus einer Holzplatte mittig ein Loch ausgeschnitten und der Motor mit Propeller wurde darüber montiert – siehe Abbildung 3.1.



Abbildung 3.1: Aufbau erster Prototyp

Der Motorregler wurde direkt an zwei in Serie geschaltete 6S-LiPo Akkus angeschlossen und mit einer Fernbedienung gesteuert.

Erkenntnisse

Der Test hat gezeigt, dass unser Motor mit dem verwendeten Propeller genug Kraft hat, um einen erwachsenen Menschen zu tragen. Dies bedeutet, dass der Bau eines Luftkissenboots nach unserem Plan möglich ist. Durch die großen Luftspalten zwischen der Holzplatte und dem Schlauchboot ging sehr viel Luft verloren, was die Effizienz stark verschlechterte. Deshalb schließen wir in unserer fertigen Konstruktion diese Luftspalten mittels Teichfolie.

4 Konstruktion

Die Konstruktion wurde in drei einzelne Teile aufgeteilt: Bodenplatte, hinterer Aufbau und Lenkung. Diese Teile sind mit wenigen Schrauben und Steckverbindungen voneinander trennbar, um den Transport und die Lagerung zu vereinfachen. Der Aufbau wurde in dem 3D-Programm Inventor^[2] geplant und konstruiert.

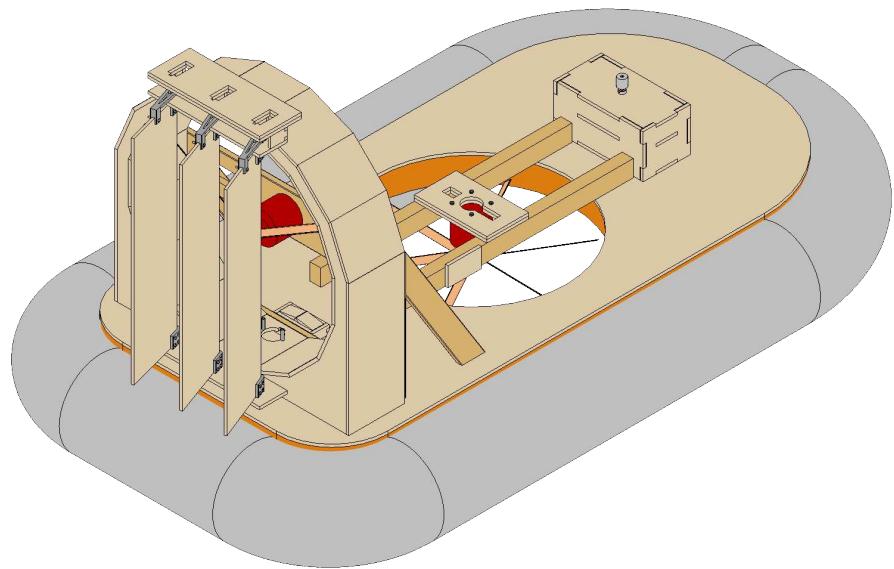


Abbildung 4.1: Konstruktion 3D-Modell gesamt

4.1 Bodenplatte

Die Bodenplatte ist die Grundlage, auf welcher die anderen beiden Teile befestigt sind, und stellt die Verbindung mit dem Boot dar.

Um die Bodenplatte möglichst leicht, aber trotzdem stabil, zu halten, wurde extrudiertes Polystyrol (XPS) in Verbindung mit Pappelsperrholz verwendet.

In der Mitte der Bodenplatte wurde das Loch für den Propeller ausgeschnitten. Im XPS wurde der Lochdurchmesser so gewählt, dass der Propeller darin frei drehen kann. Der Lochdurchmesser in der Holzplatte darüber wurde um 2 cm kleiner gewählt, um zu verhindern, dass am Rand des Loches die Luft direkt wieder hinausströmt.

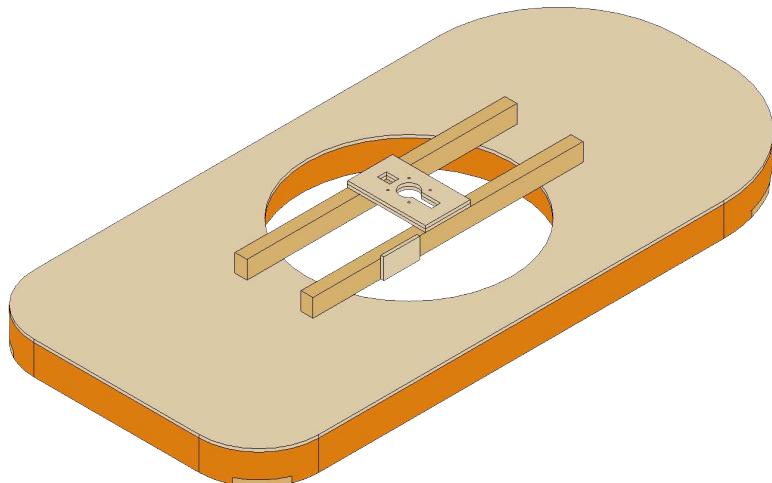


Abbildung 4.2: Bodenplatte 3D-Modell

4.1.1 Stückliste

Stk	Name	Material	Herstellung	Abbildung
1	Bodenplatte Pappel	10mm Pappel	Stichsäge	4.5
1	Bodenplatte XPS	100mm XPS	Messer	4.6, 4.7
1	Verstärkung Holzlatte links	Holzlatte Fichte 20x90mm	Kappsäge	4.8
1	Verstärkung Holzlatte rechts	Holzlatte Fichte 20x90mm	Kappsäge	4.9
2	Staffel Motorhalterung	Staffel gehobelt 40x60mm	Kappsäge	4.10
2	Platte Motorhalterung	10mm Pappel	LaserCutter	4.11
1	Platte Motorregler-Halterung	10mm Pappel	LaserCutter	4.12
1	Gitter unten	Lochplatte	Einkauf	4.13

Tabelle 4.1: Stückliste Bodenplatte

4.1.2 Inventor

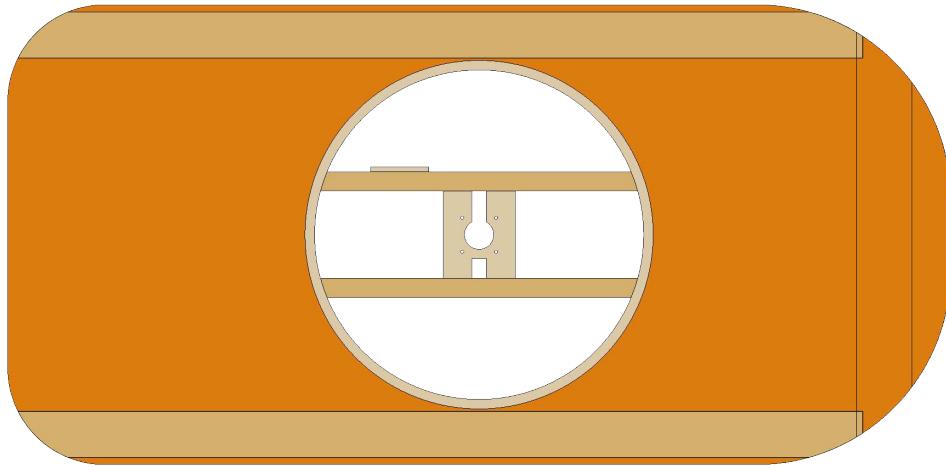


Abbildung 4.3: Bodenplatte Ansicht unten

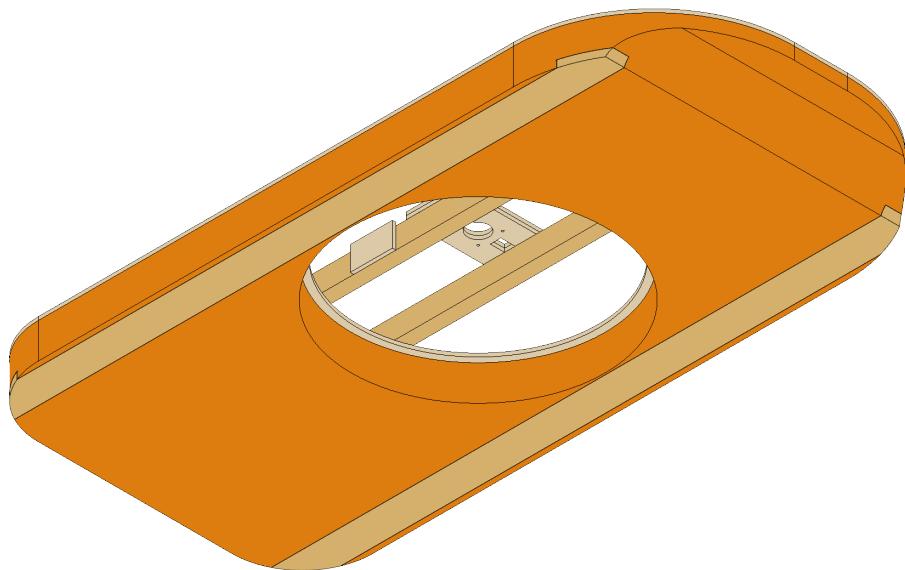


Abbildung 4.4: Bodenplatte Ansicht unten seitlich

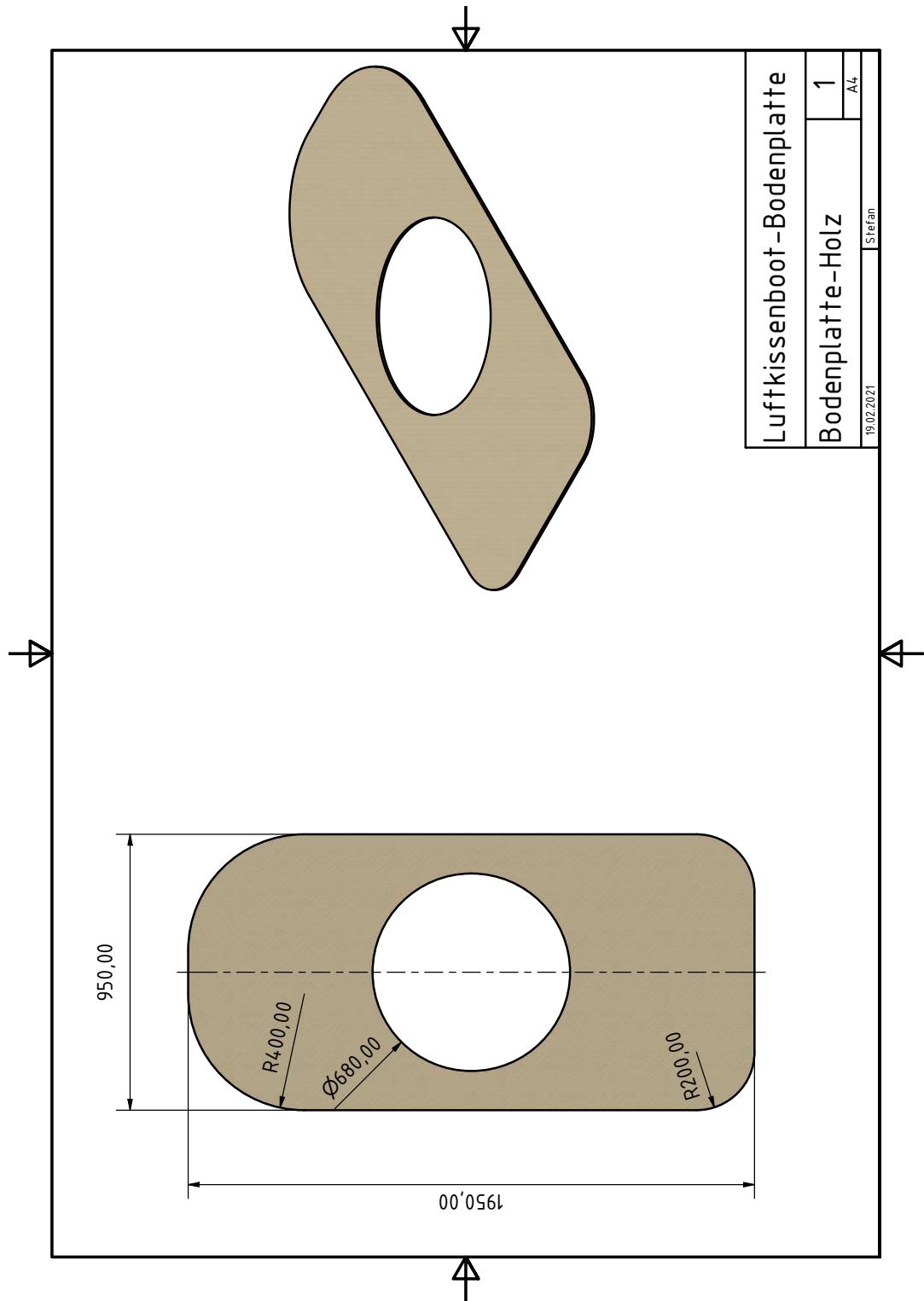


Abbildung 4.5: Zeichnung Bodenplatte Pappel

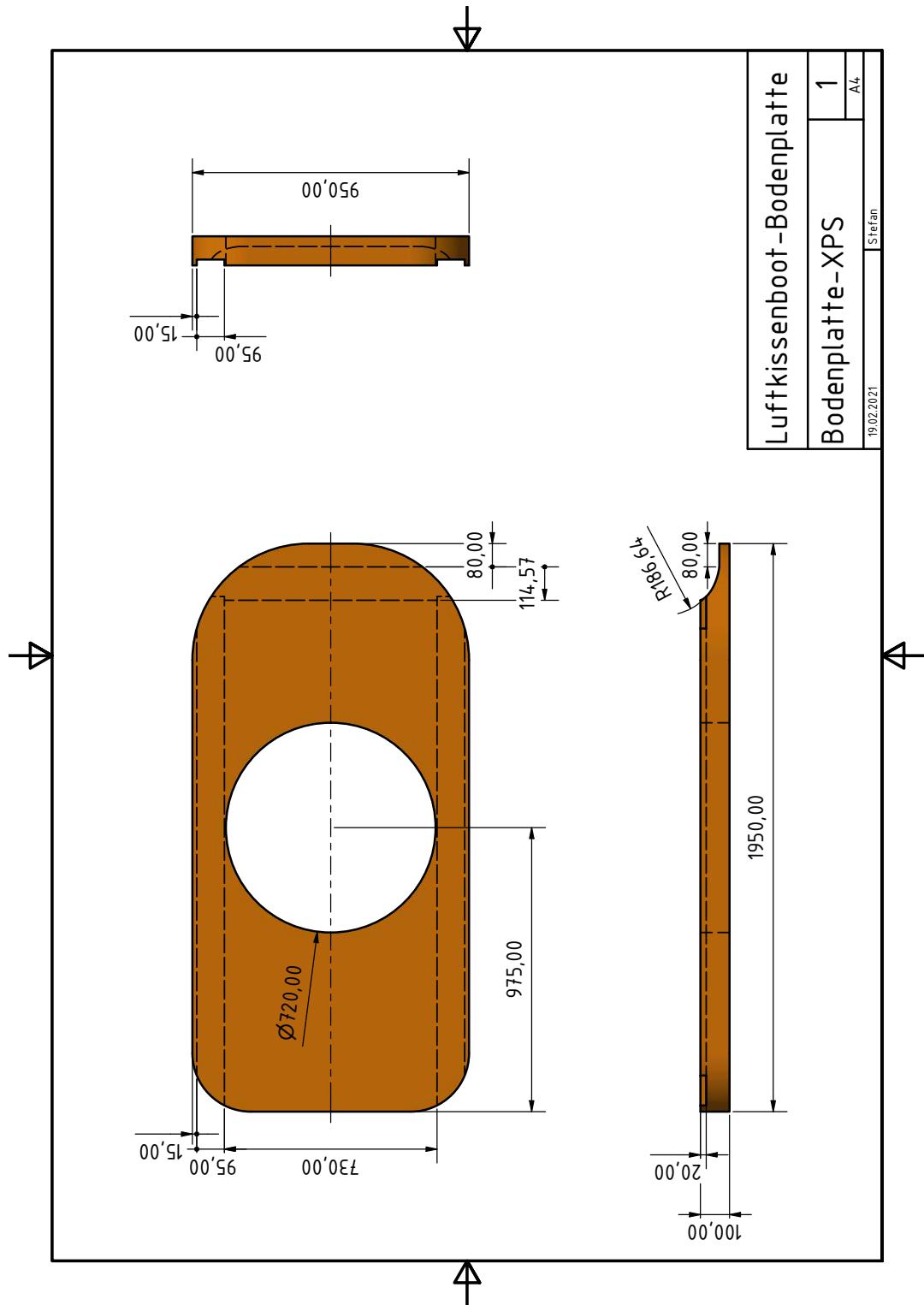


Abbildung 4.6: Zeichnung Bodenplatte XPS 1

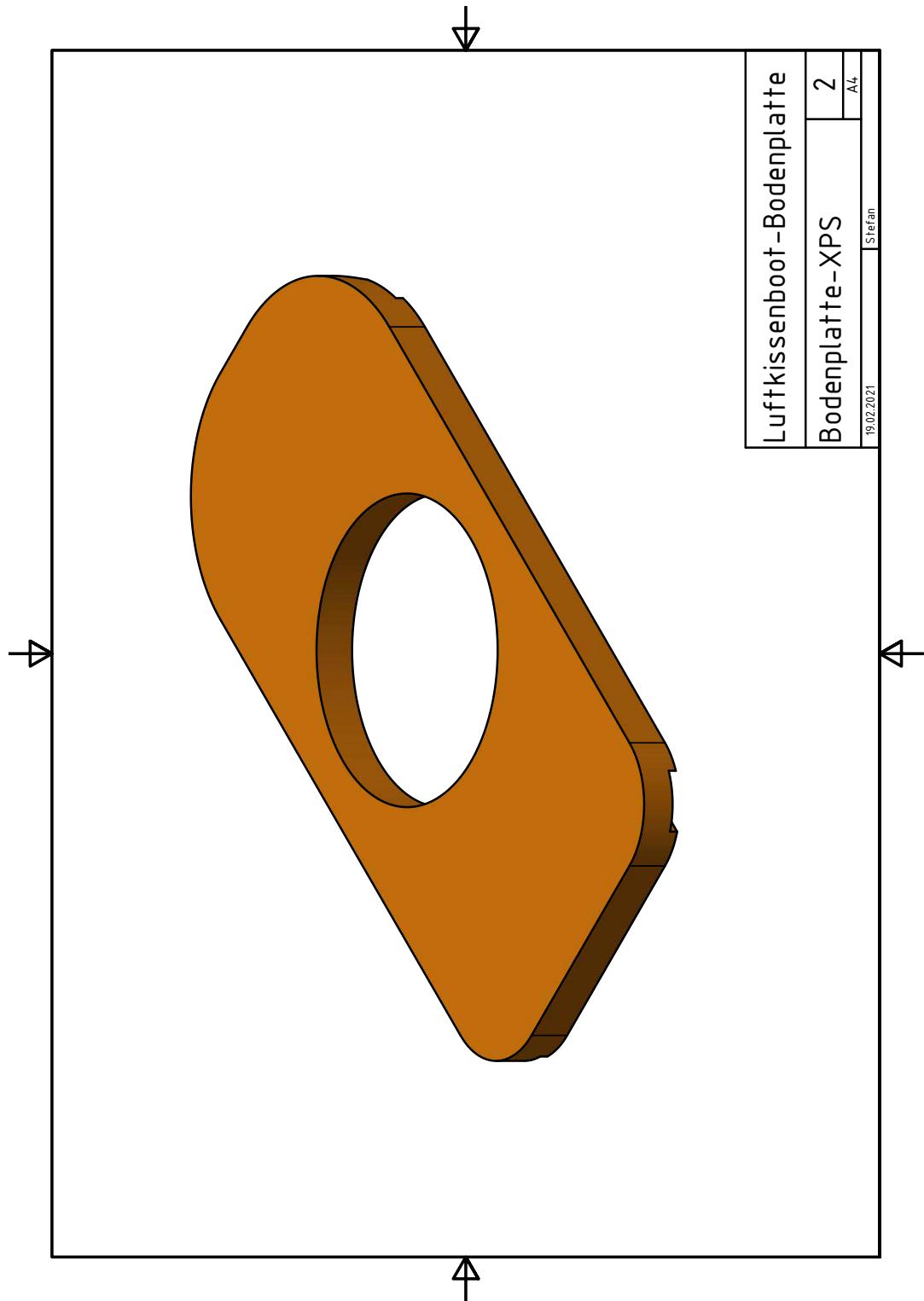


Abbildung 4.7: Zeichnung Bodenplatte XPS 2

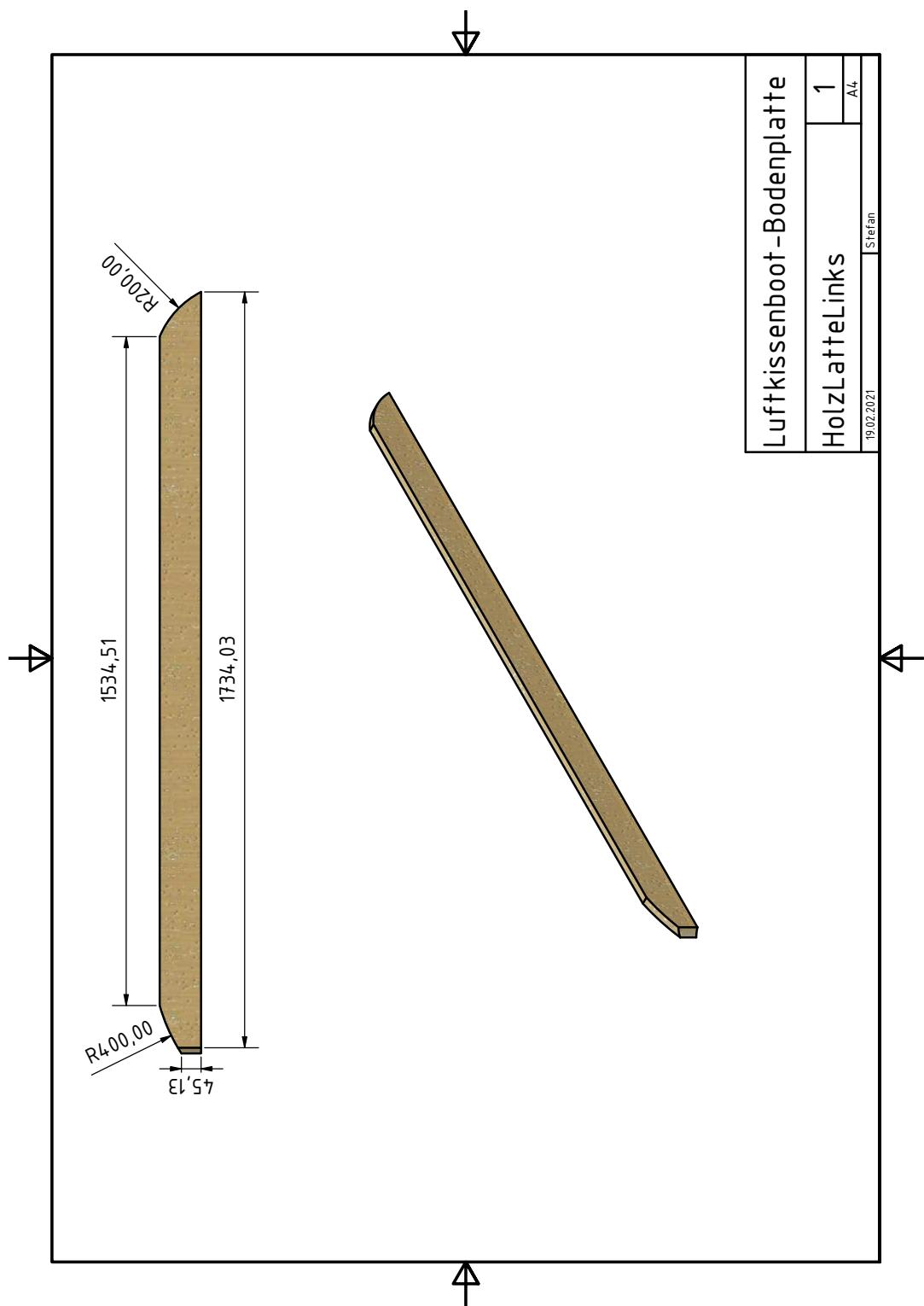


Abbildung 4.8: Zeichnung Verstärkung Holzplatte links

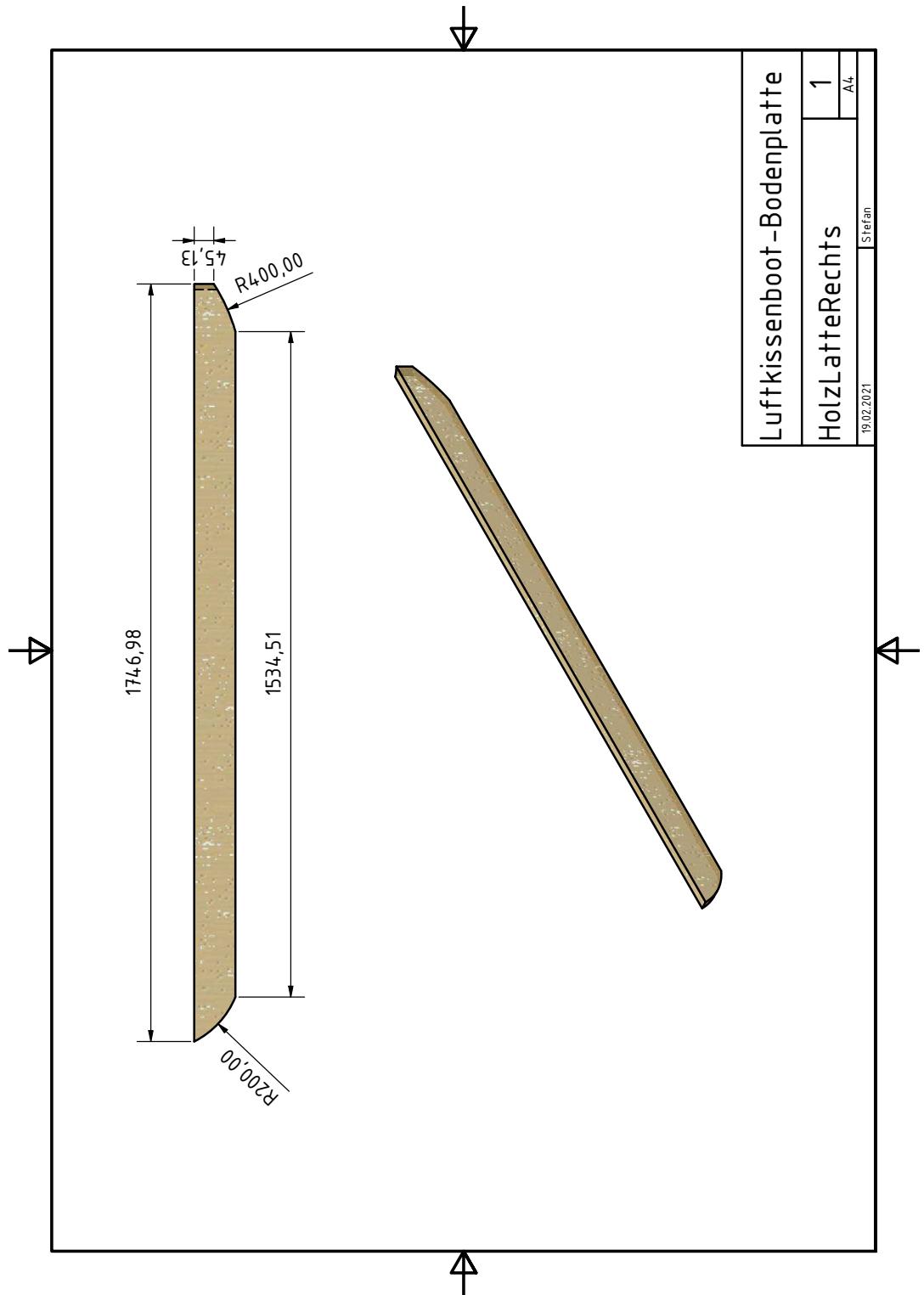


Abbildung 4.9: Zeichnung Verstärkung Holzplatte rechts

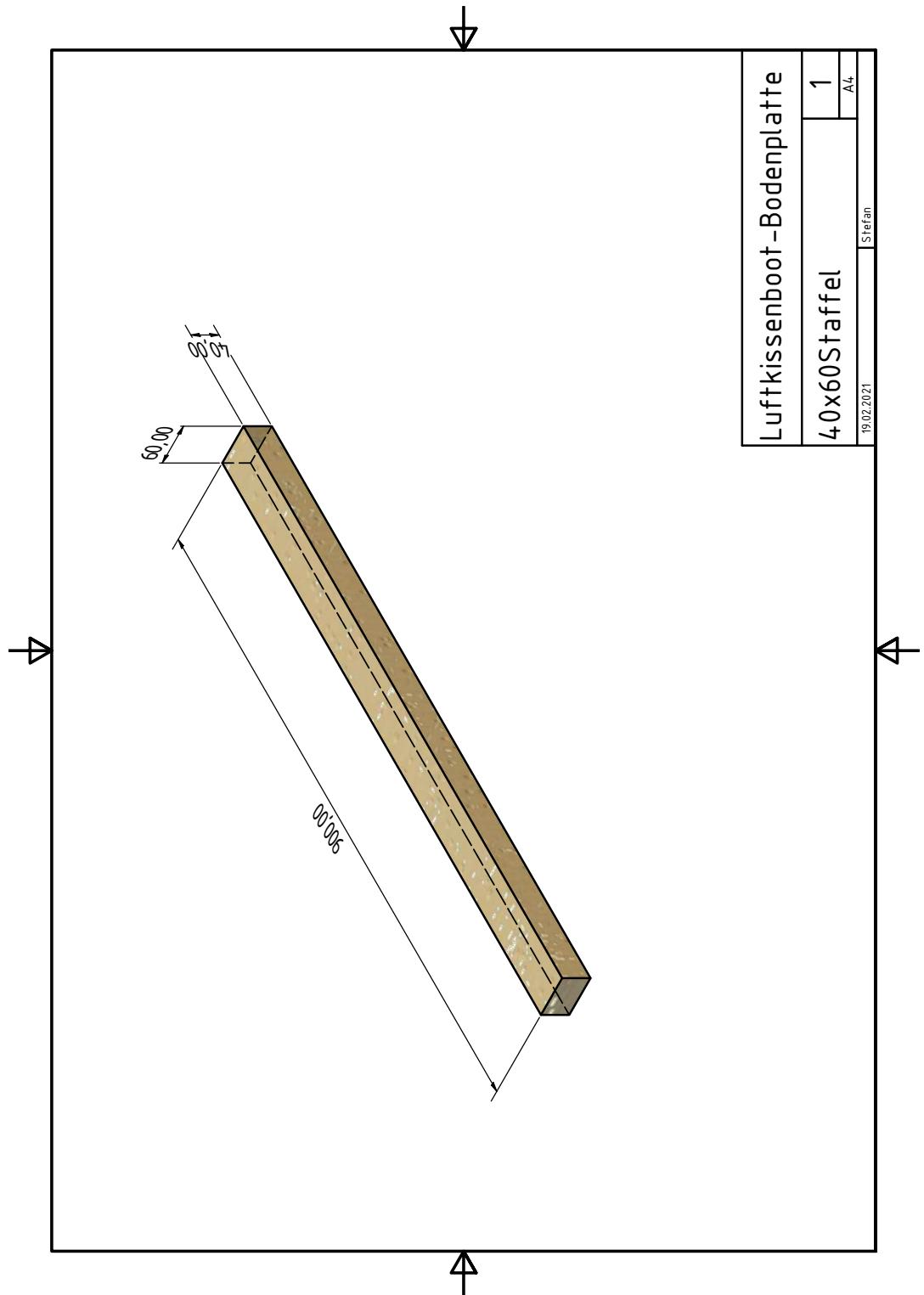


Abbildung 4.10: Zeichnung Staffel Motorhalterung

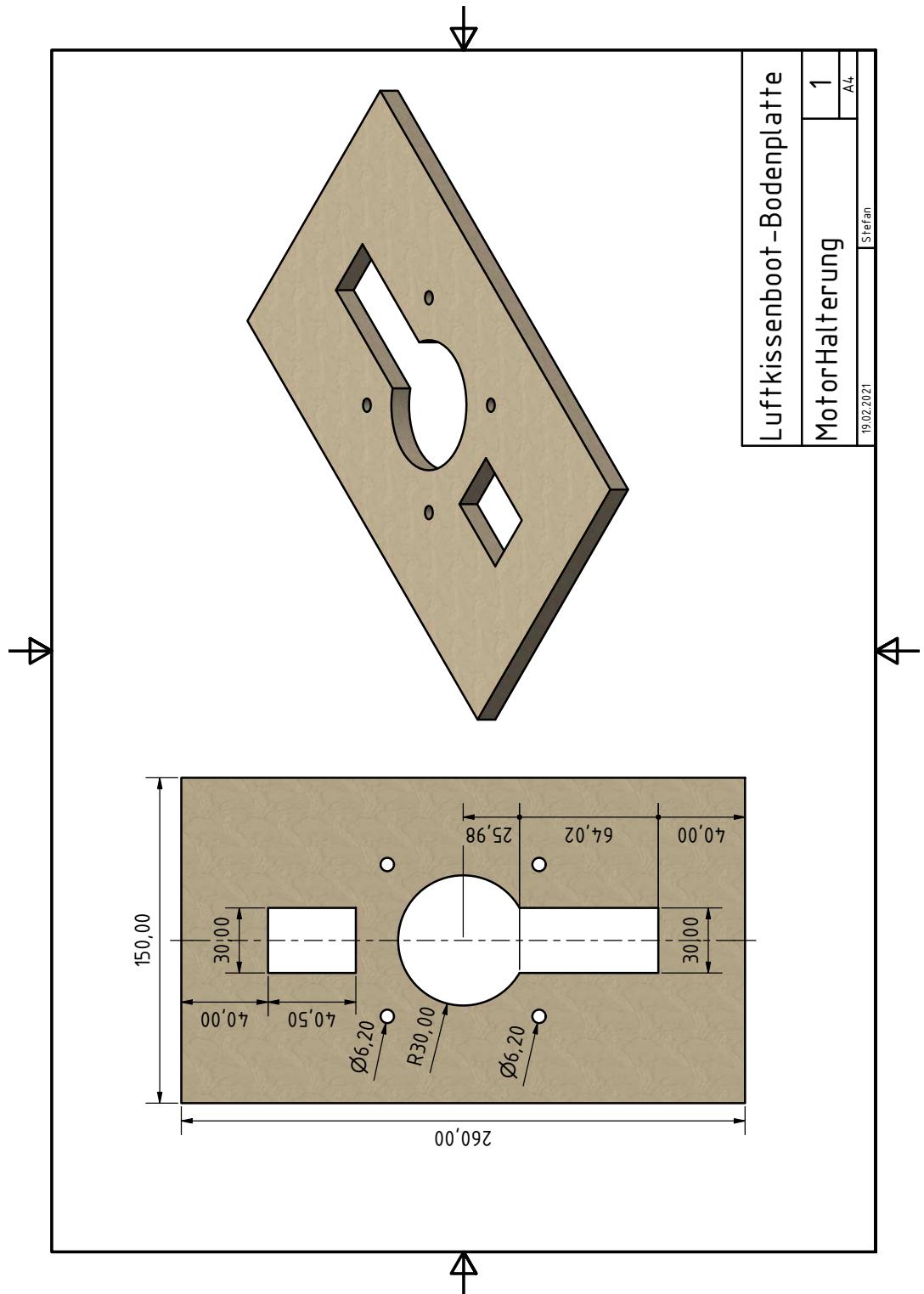


Abbildung 4.11: Zeichnung Motorhalterung

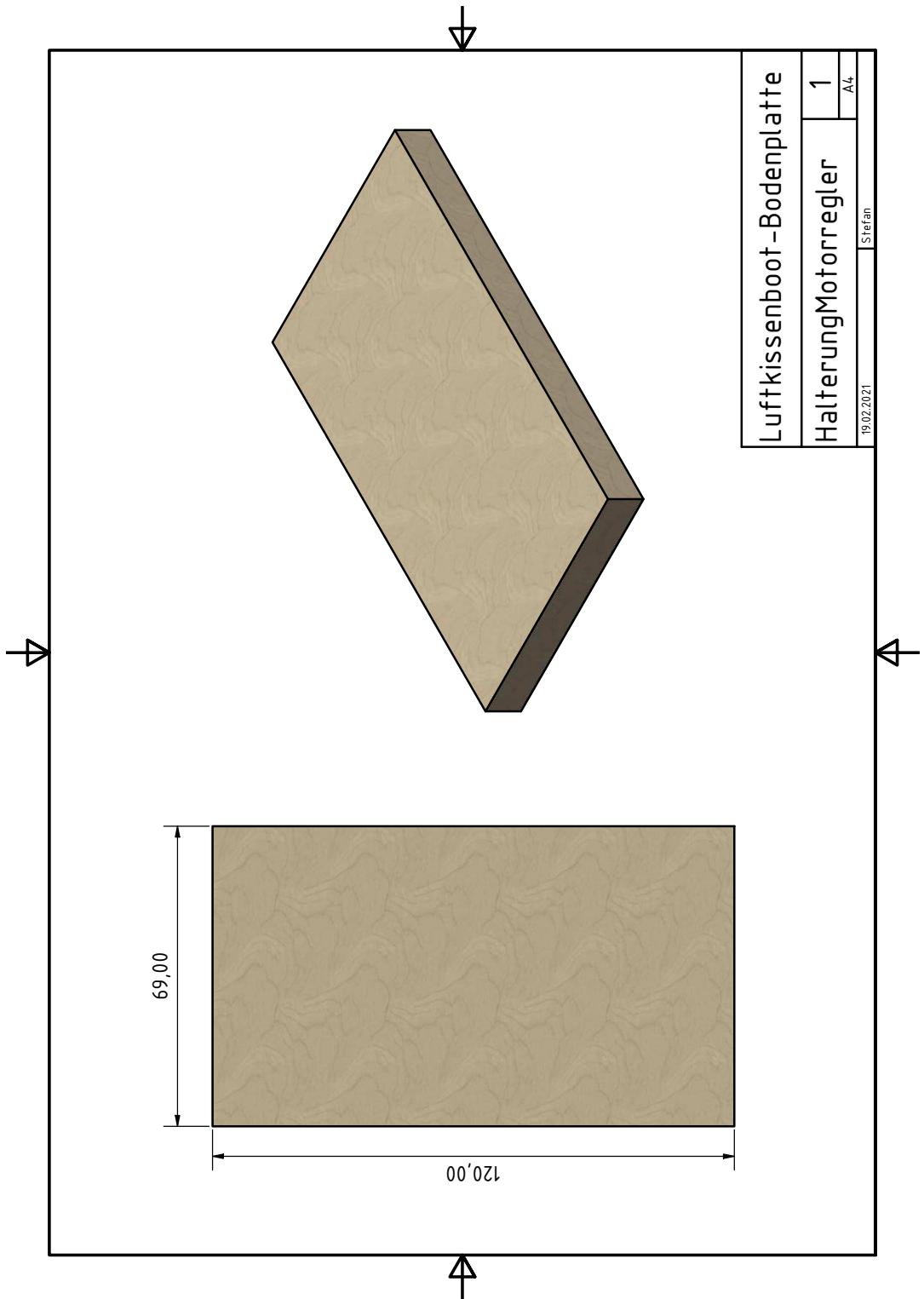


Abbildung 4.12: Zeichnung Motorregler-Halterung

4.1.3 Gitter

Aus Sicherheitsgründen wurde über dem Propeller ein Gitter angebracht. Dieses musste so stabil gewählt werden, dass der Fahrer problemlos darauf sitzen kann. Gleichzeitig muss es aber auch genug Luft durchlassen. Wir haben uns deshalb für ein Lochblech mit quadratischen Löchern mit 5mm Seitenlänge entschieden. Der Steg zwischen den Löchern ist 3mm breit, insgesamt ist das Gitter 73x73cm groß.

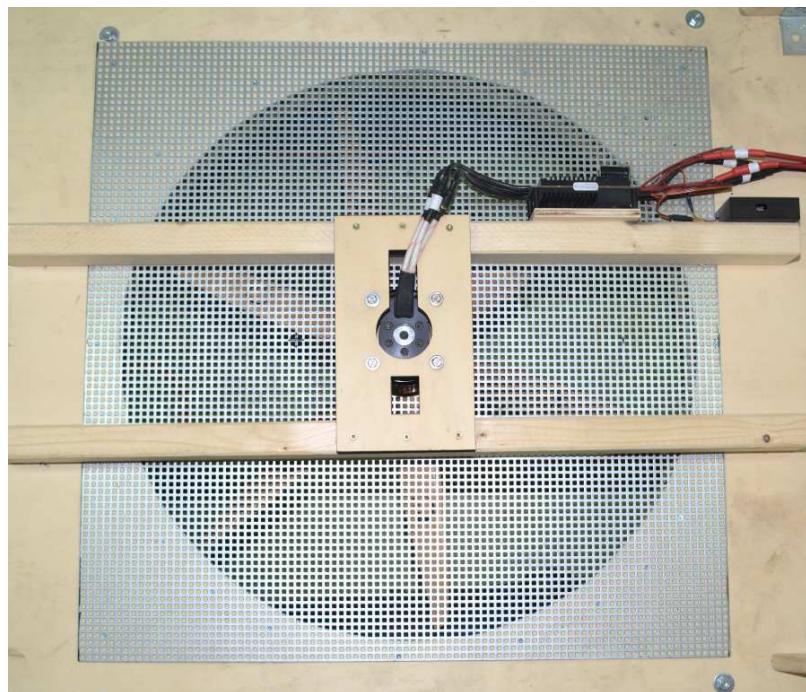


Abbildung 4.13: Foto Gitter

4.1.4 Zusammenbau

Die XPS-Platte wurde aus 4 einzelnen 100mm dicken XPS-Platten zusammengeklebt und mit einem Messer zugeschnitten. Die Schlitze für die Holzlatten wurden mit einer Oberflächenfräse eingefräst und die Holzlatten (Abbildung 4.8 & Abbildung 4.9) wurden mit Silikon eingeklebt.

Die Pappelsperrholzplatte wurde mit acht M8x120mm Schrauben mit der XPS-Platte verschraubt.

Danach wurde das Metallgitter montiert und die Staffeln zur Motorhalterung wurden darüber geleimt und geschraubt.

Um die anderen beiden Teile befestigen zu können, wurden einige Winkel an der Bodenplatte angeschraubt – siehe Abbildung 4.16.

Außerdem wurde in die XPS-Platte ein Kabelkanal für eine CAN-Bus-Leitung, eine Leitung für den Not-Aus sowie eine für den Precharge-Button eingefräst. Dieser Kabelkanal läuft von der Lenkung bis zum hinteren Aufbau.

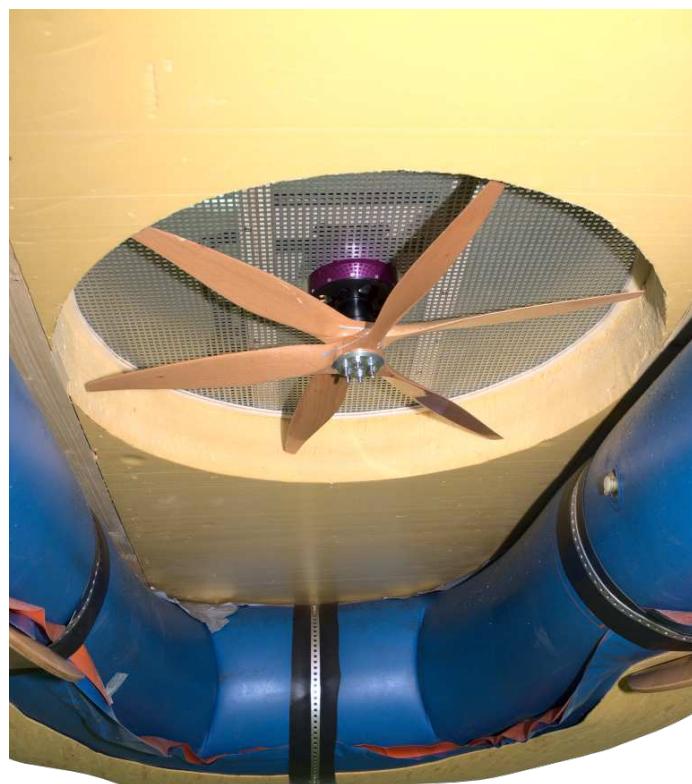


Abbildung 4.14: Zusammenbau Bodenplatte Sicht von unten



Abbildung 4.15: Zusammenbau Bodenplatte



Abbildung 4.16: Bodenplatte Winkelverbinder

4.1.5 Montage am Boot

Die Bodenplatte soll möglichst luftdicht mit dem Schlauchboot verbunden werden. Um dies zu erreichen, wurde eine Teichfolie mit doppelseitigem Klebeband am Boot angeklebt und an der Holzplatte mit einer Leiste angeschraubt.

Zum Schutz der Unterseite des Bootes beim Bremsen wurde eine Hartgummiplatte zugeschnitten und mit Pattex Kraftkleber an das Boot geklebt.

Um ein Verrutschen der Konstruktion zu verhindern, wurde die Bodenplatte zusätzlich mit einem Lochband am Boot befestigt – siehe Abbildung 4.17



Abbildung 4.17: Befestigung Lochband

4.2 Hinterer Aufbau

Der hintere Aufbau beinhaltet den Motor für den Vortrieb sowie die Fahnen zum Lenken. Die drei Fahnen werden von drei einzelnen Servos angesteuert.

Außerdem sind die Akkus und die Kupferschienen zur Stromverteilung im unteren Teil des Aufbaus versteckt. Um die Akkus zugänglich zu machen, sind die zwei äußeren Platten, dargestellt in Abbildung 4.28, aufklappbar.

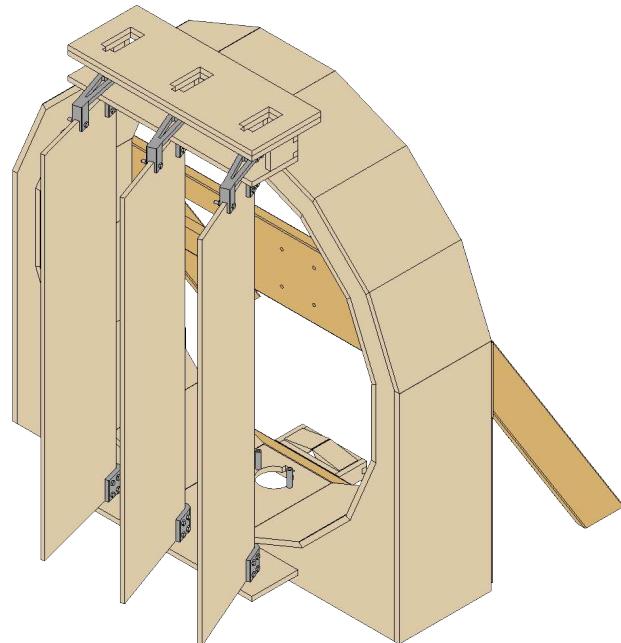


Abbildung 4.18: Hinterer Aufbau 3D-Modell

4.2.1 Stückliste

Stk	Name	Material	Herstellung	Abbildung
2	Hauptplatte	10mm Pappel	Stichsäge	4.21
1	Platte innen unten	10mm Pappel	Stichsäge	4.22
14	Platten innen 1 schräge Kante	10mm Pappel	Handkreissäge	4.23
1	Platte innen 2 schräge Kanten	10mm Pappel	Handkreissäge	4.24
1	Platte außen oben	10mm Pappel	Stichsäge	4.25
4	Platte außen 1 schräge Kante	10mm Pappel	Handkreissäge	4.26
2	Platte außen 2 schräge Kanten	10mm Pappel	Handkreissäge	4.27
2	Platte außen Seite	10mm Pappel	Handkreissäge	4.28
1	Verstärkung hinten Mitte	10mm Pappel	Stichsäge	4.29
2	Verstärkung hinten unten	10mm Pappel	Stichsäge	4.30
2	Verstärkung hinten schräg	10mm Pappel	Stichsäge	4.31
1	Verstärkung unten	10mm Pappel	Handkreissäge	4.32
2	Servo Halter seitlich	10mm Pappel	LaserCutter	4.33
2	Servo Halter seitlich 2	10mm Pappel	LaserCutter	4.34
1	Servo Halter hinten	10mm Pappel	LaserCutter	4.35
1	Servo Halter lang	10mm Pappel	LaserCutter	4.36
2	Servo Halter Hauptplatte	10mm Pappel	LaserCutter	4.37
2	Kabelbox Platte kurz	10mm Pappel	LaserCutter	4.38
2	Kabelbox Platte lang	10mm Pappel	LaserCutter	4.39
2	Kabelbox Deckel	10mm Pappel	LaserCutter	4.40
2	Fahnenhalter Hauptteil	PETG	3D-Drucker	4.41
2	Fahnenhalter Servo	PETG	3D-Drucker	4.42
2	Fahnenhalter Rechteck	PETG	3D-Drucker	4.43

Tabelle 4.2: Stückliste Aufbau hinten

4.2.2 Inventor

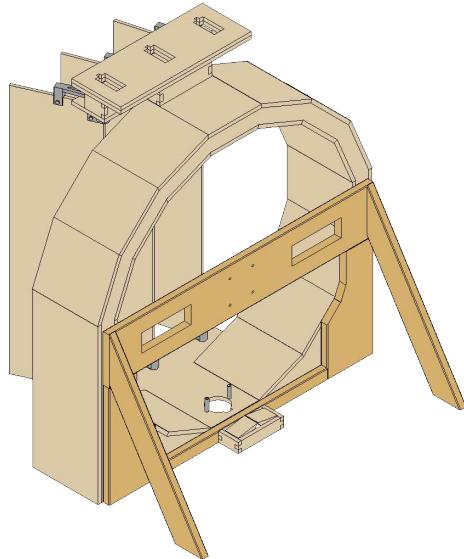


Abbildung 4.19: Hinterer Aufbau Ansicht schräg hinten

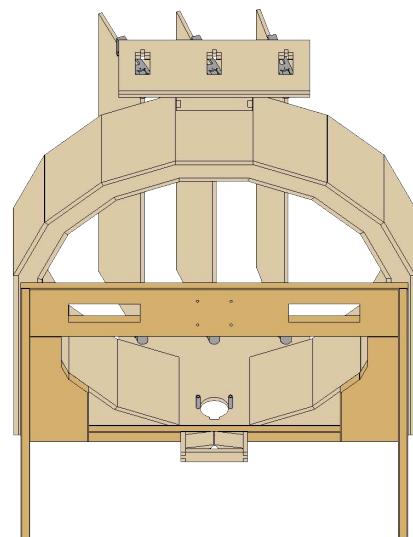


Abbildung 4.20: Hinterer Aufbau Ansicht hinten oben

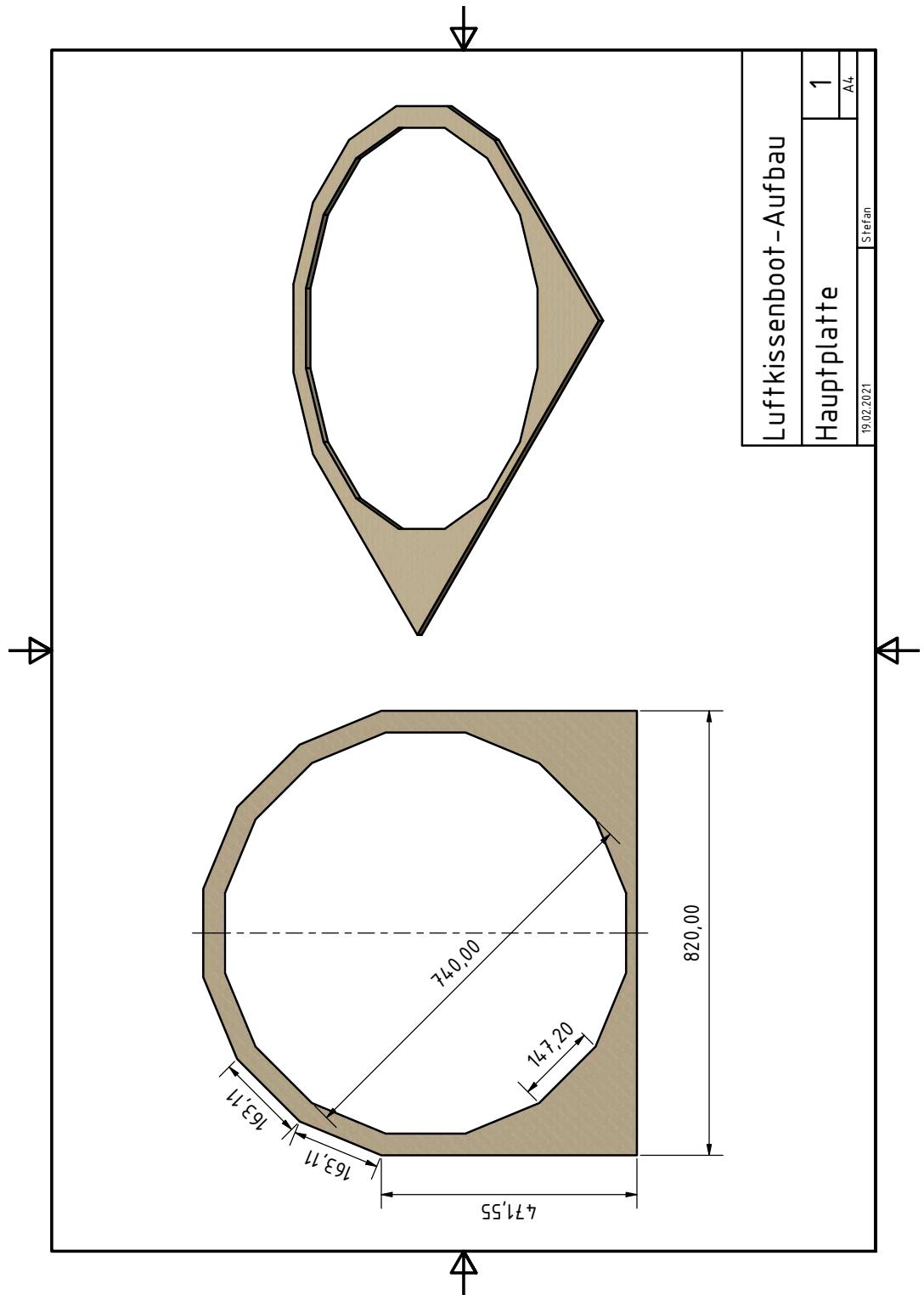


Abbildung 4.21: Zeichnung Hauptplatte

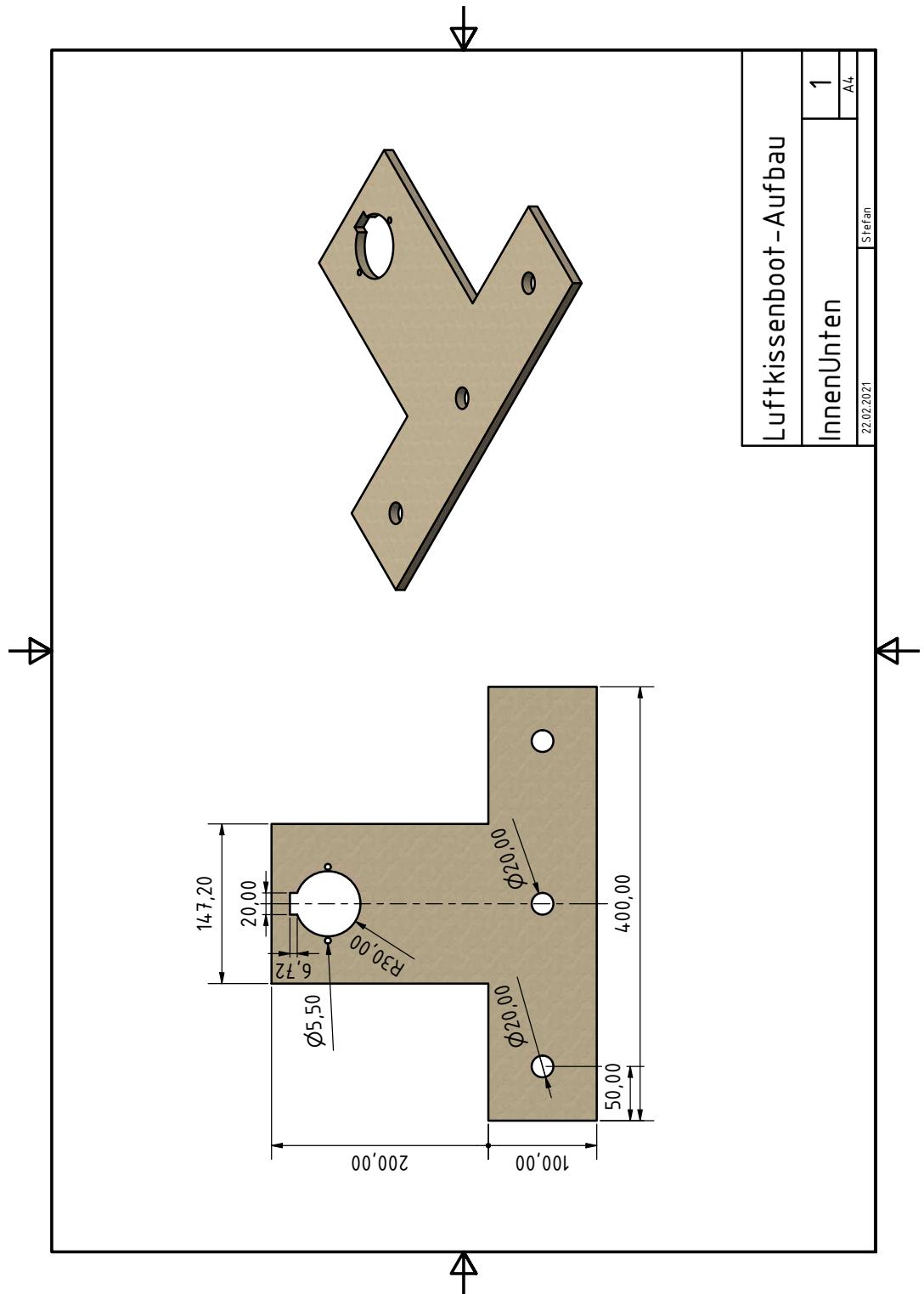


Abbildung 4.22: Zeichnung Platte innen unten

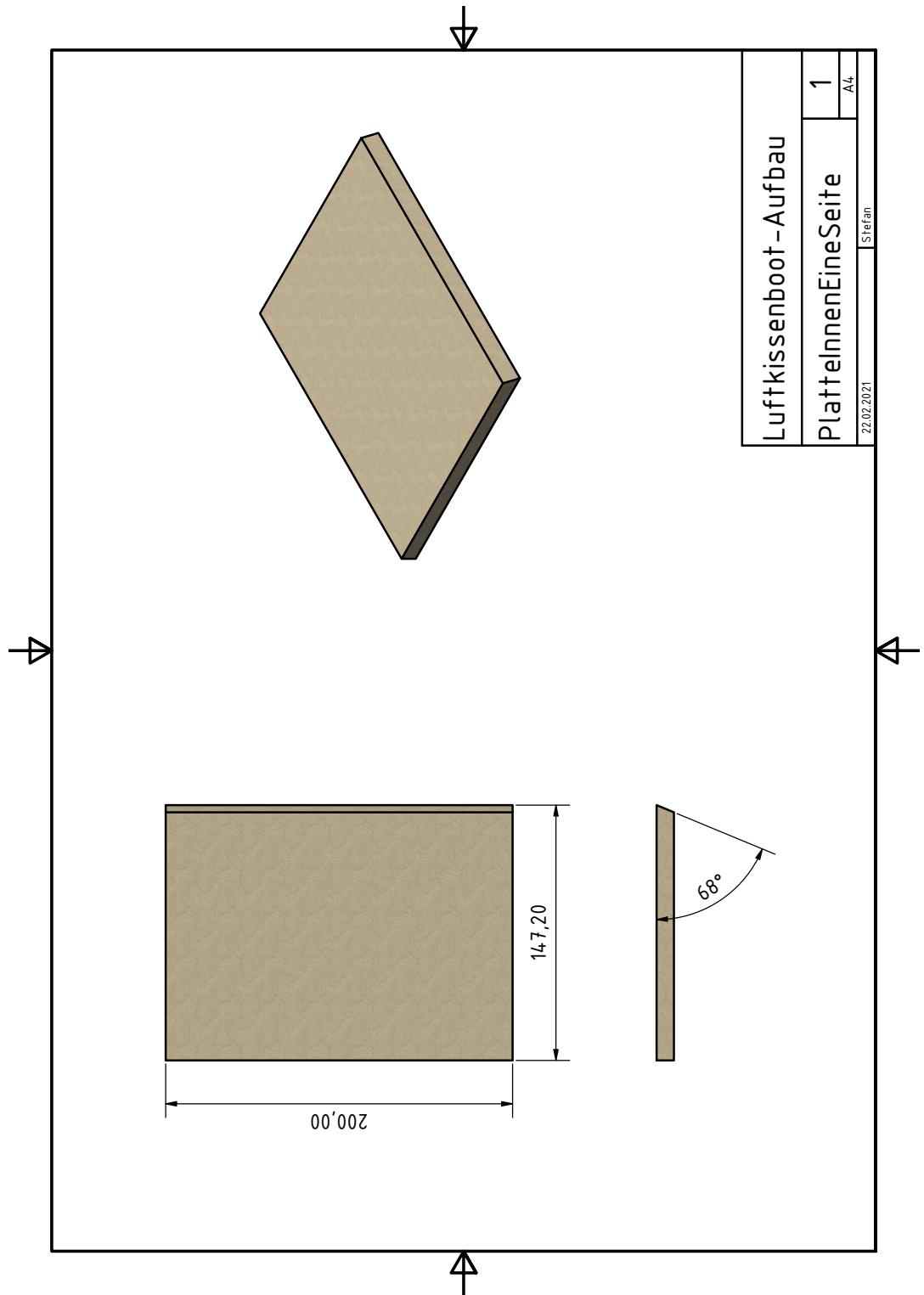


Abbildung 4.23: Zeichnung Platte innen 1 schräge Kante

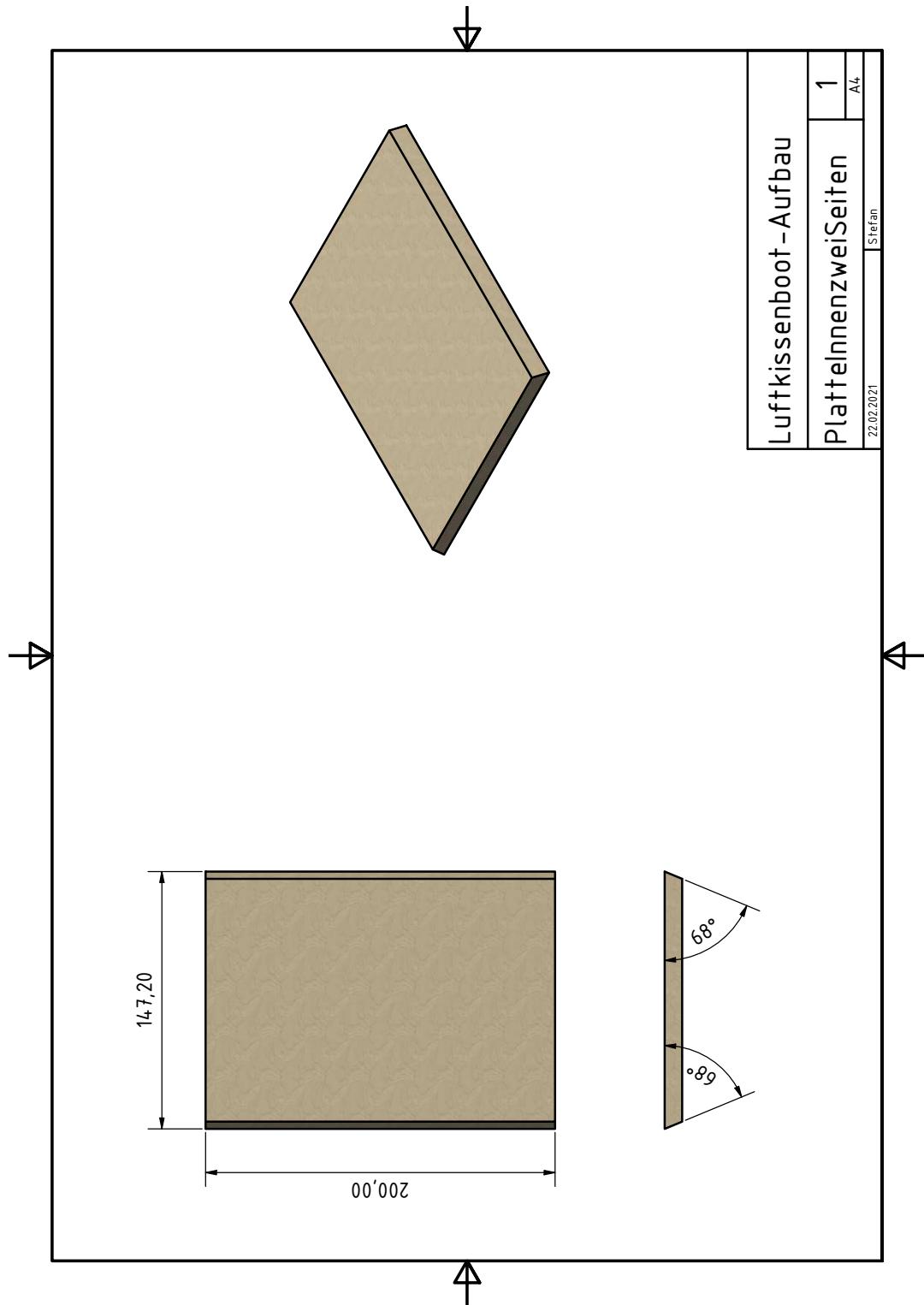


Abbildung 4.24: Zeichnung Platte innen 2 schräge Kanten

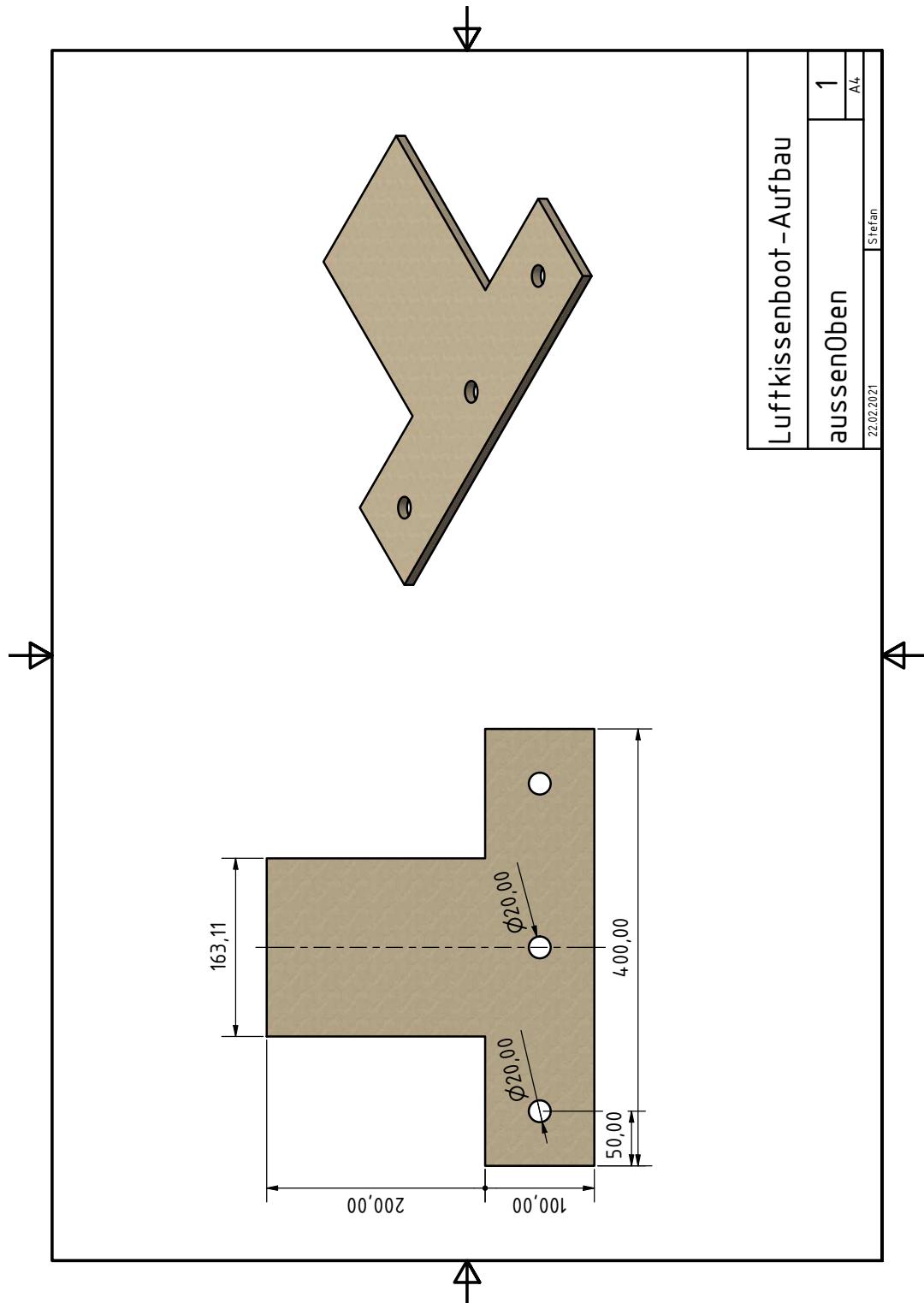


Abbildung 4.25: Zeichnung Platte außen oben

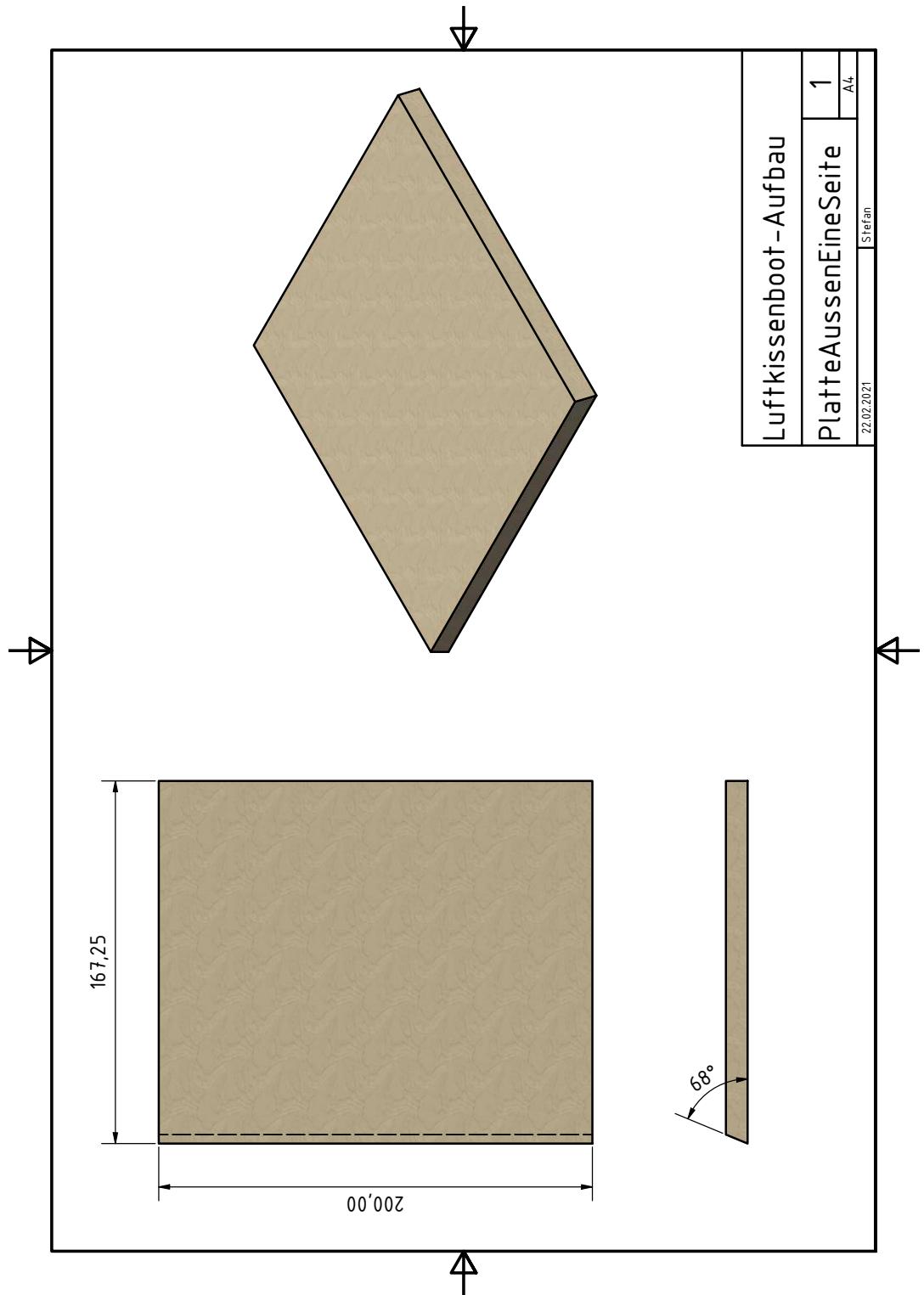


Abbildung 4.26: Zeichnung Platte außen 1 schräge Kante

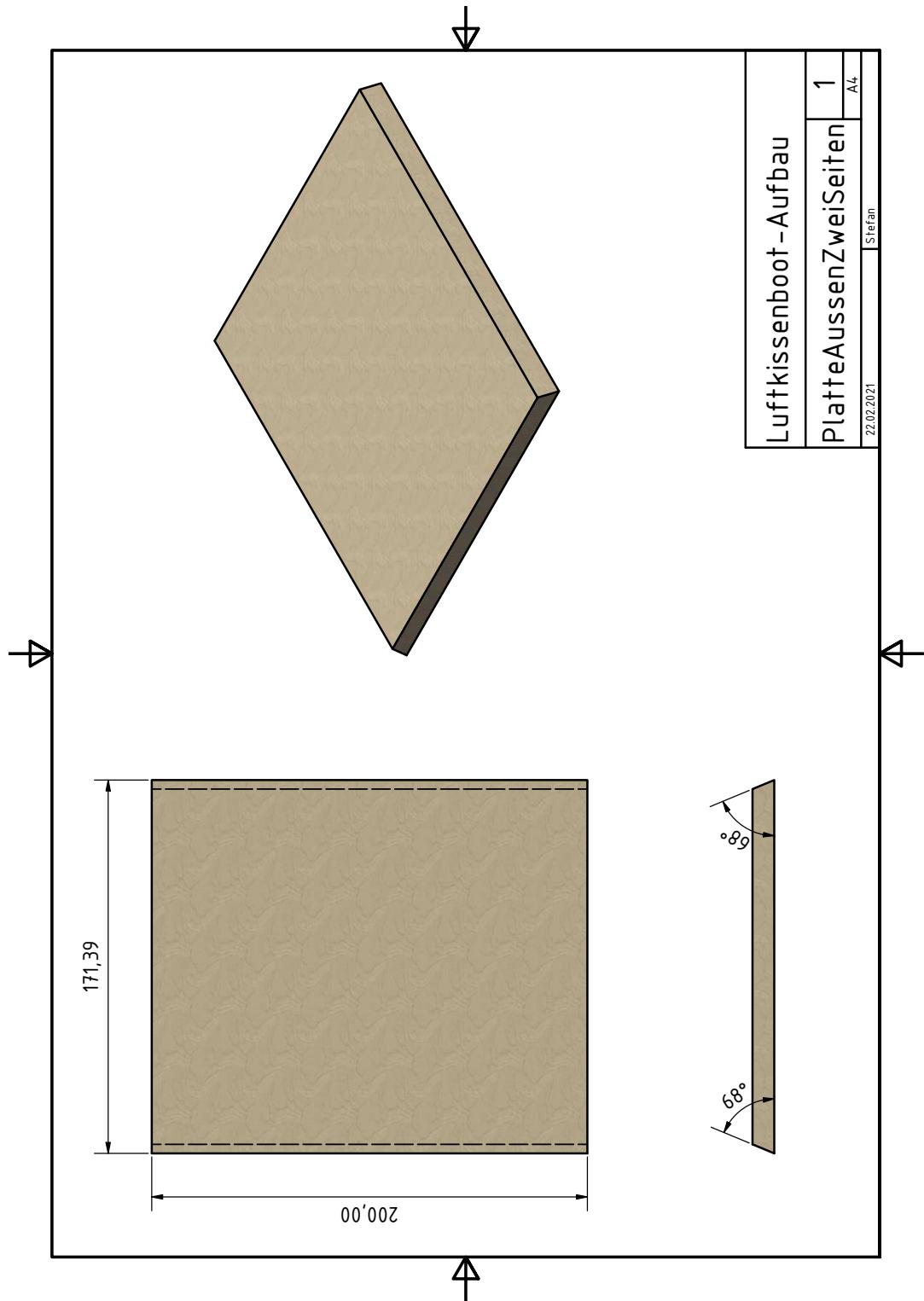


Abbildung 4.27: Zeichnung Platte außen 2 schräge Kanten

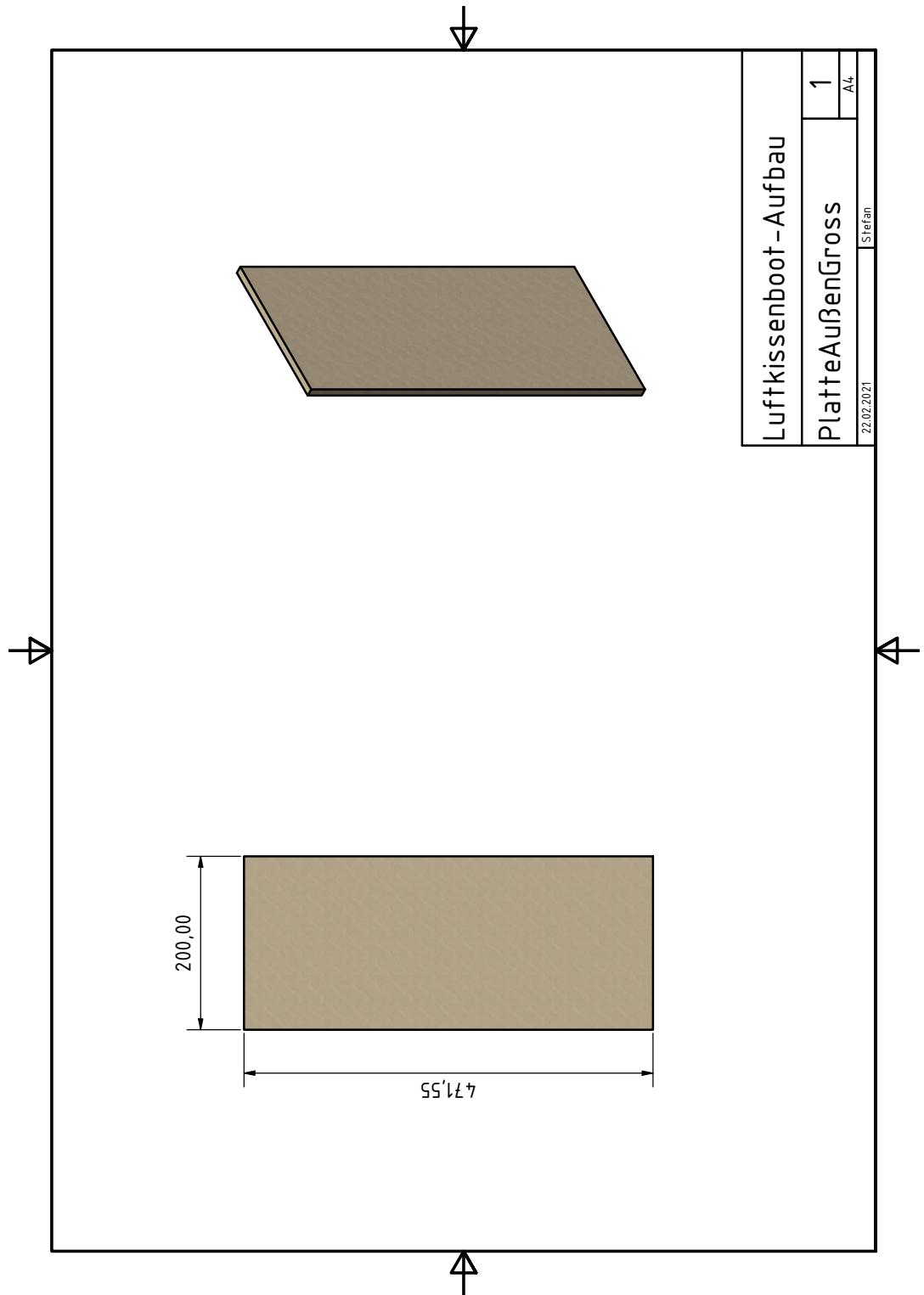


Abbildung 4.28: Zeichnung Platte außen Seite

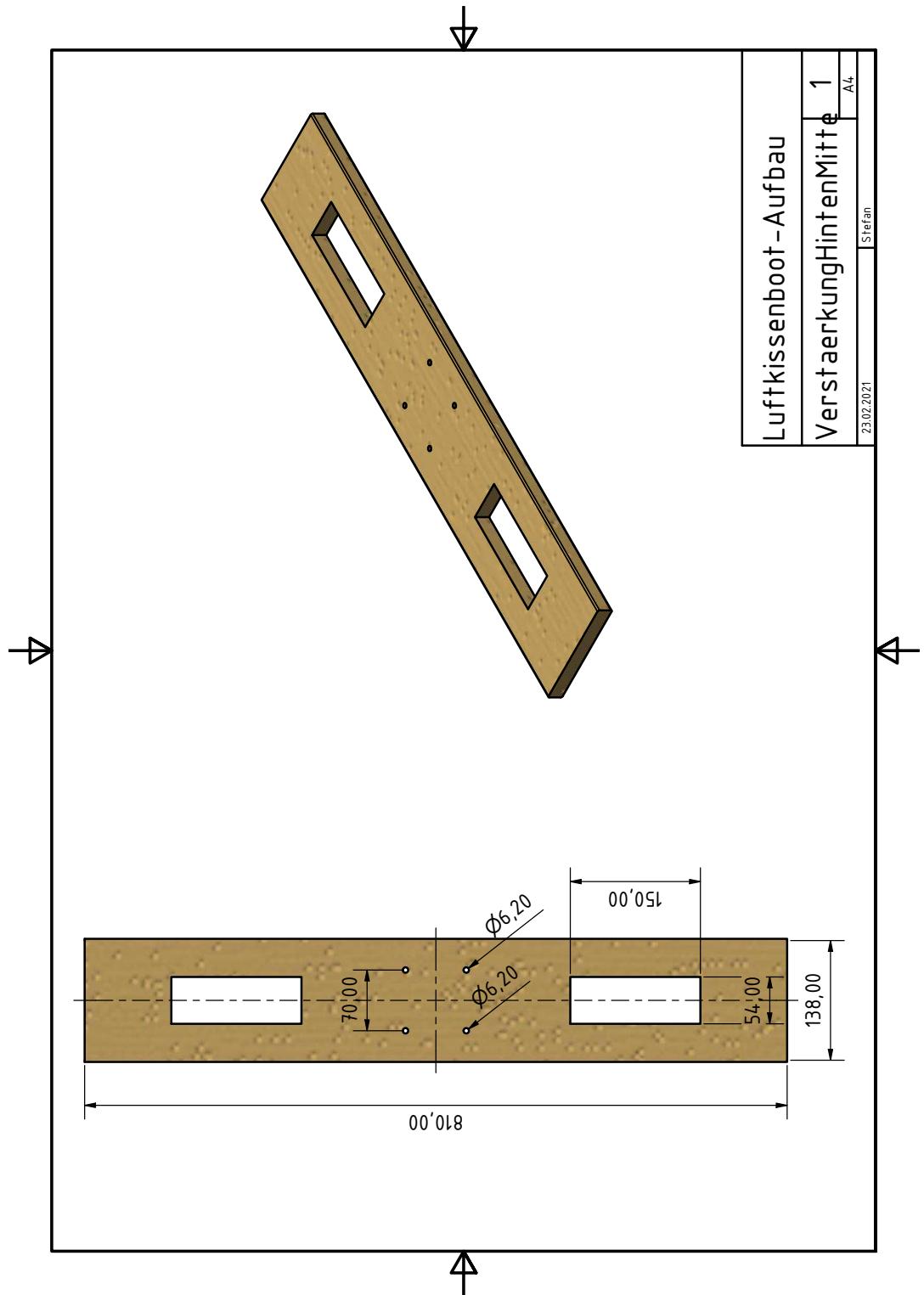


Abbildung 4.29: Zeichnung Verstärkung hinten Mitte

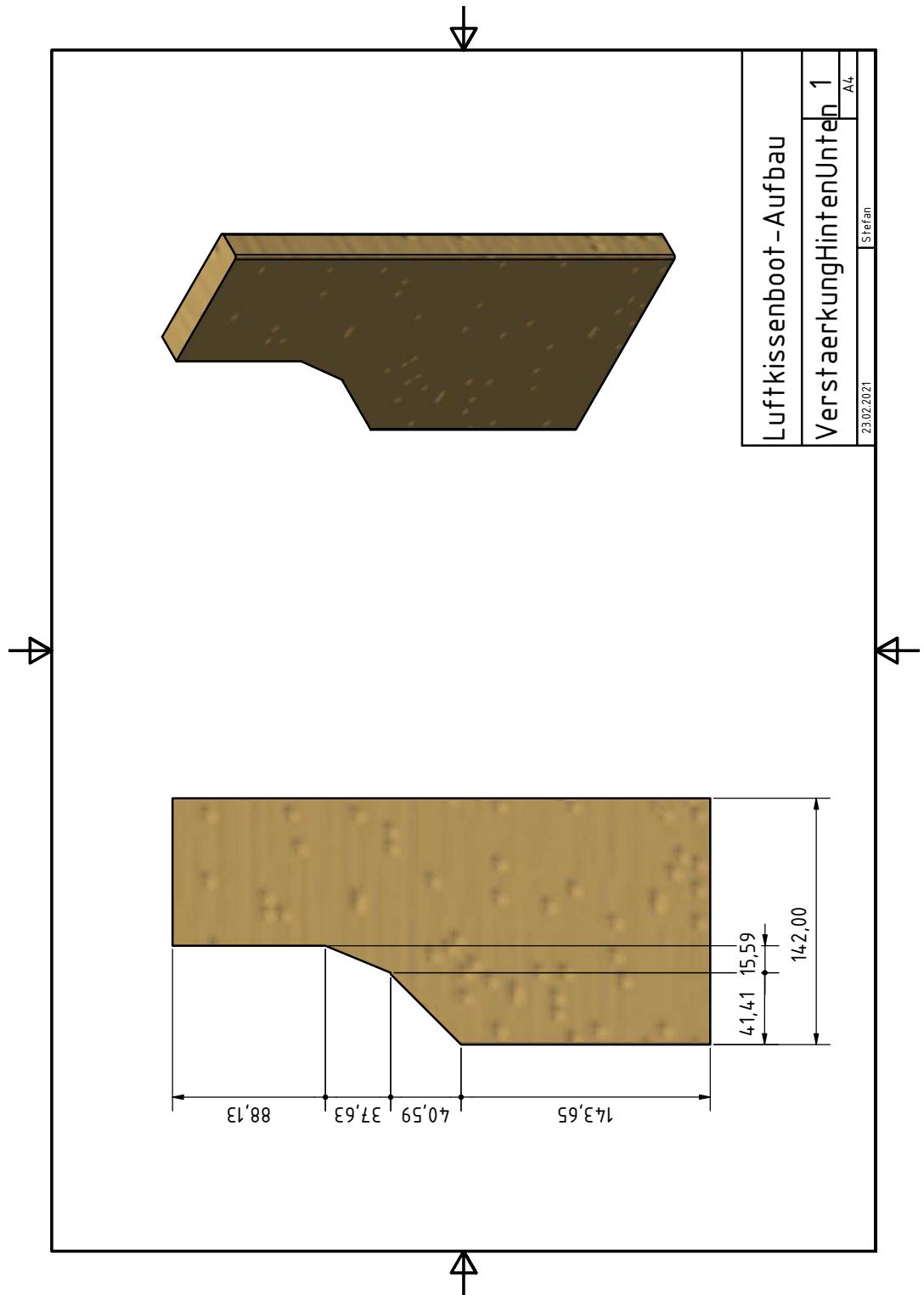


Abbildung 4.30: Zeichnung Verstärkung hinten unten

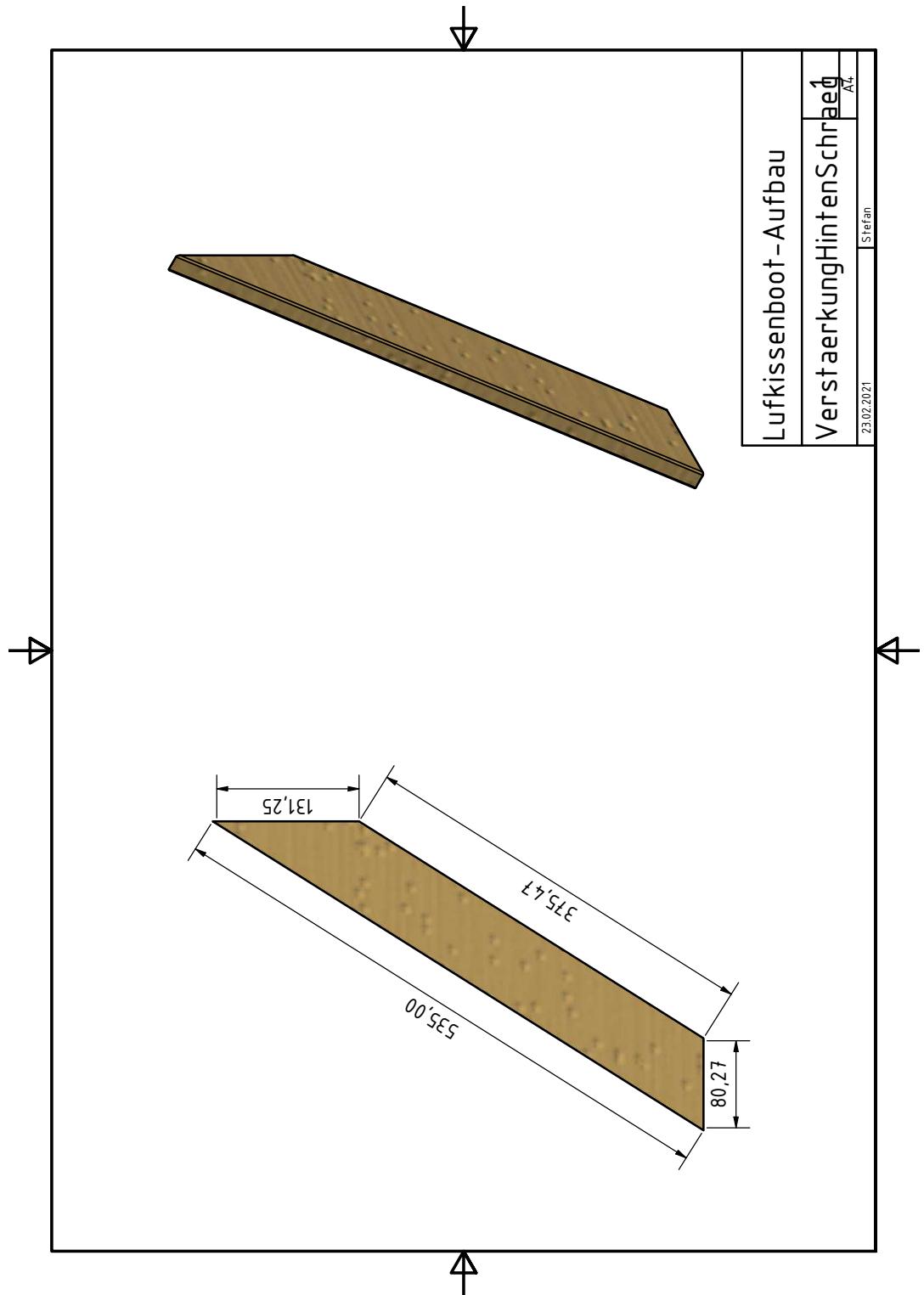


Abbildung 4.31: Zeichnung Verstärkung hinten schräg

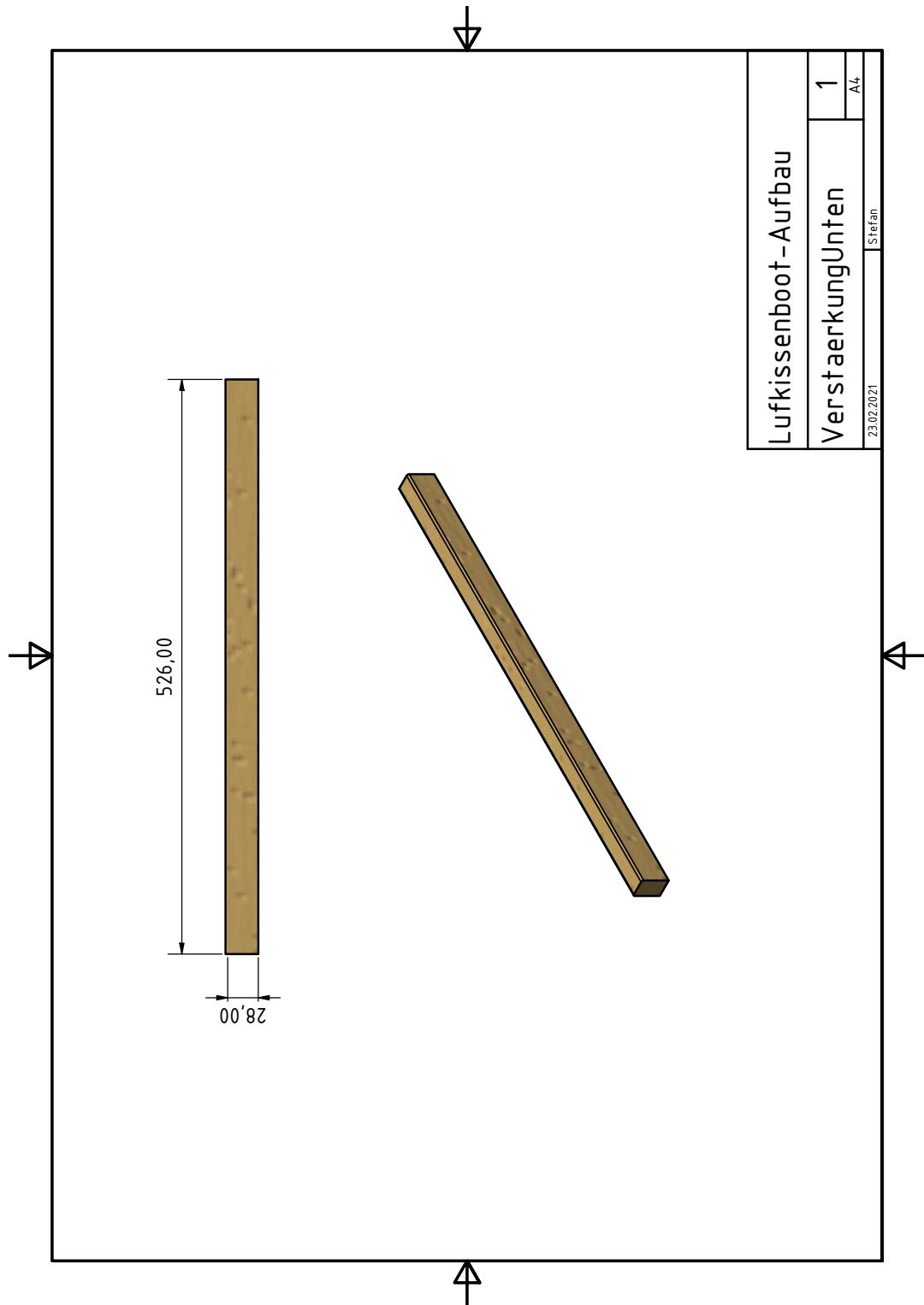


Abbildung 4.32: Zeichnung Verstärkung unten

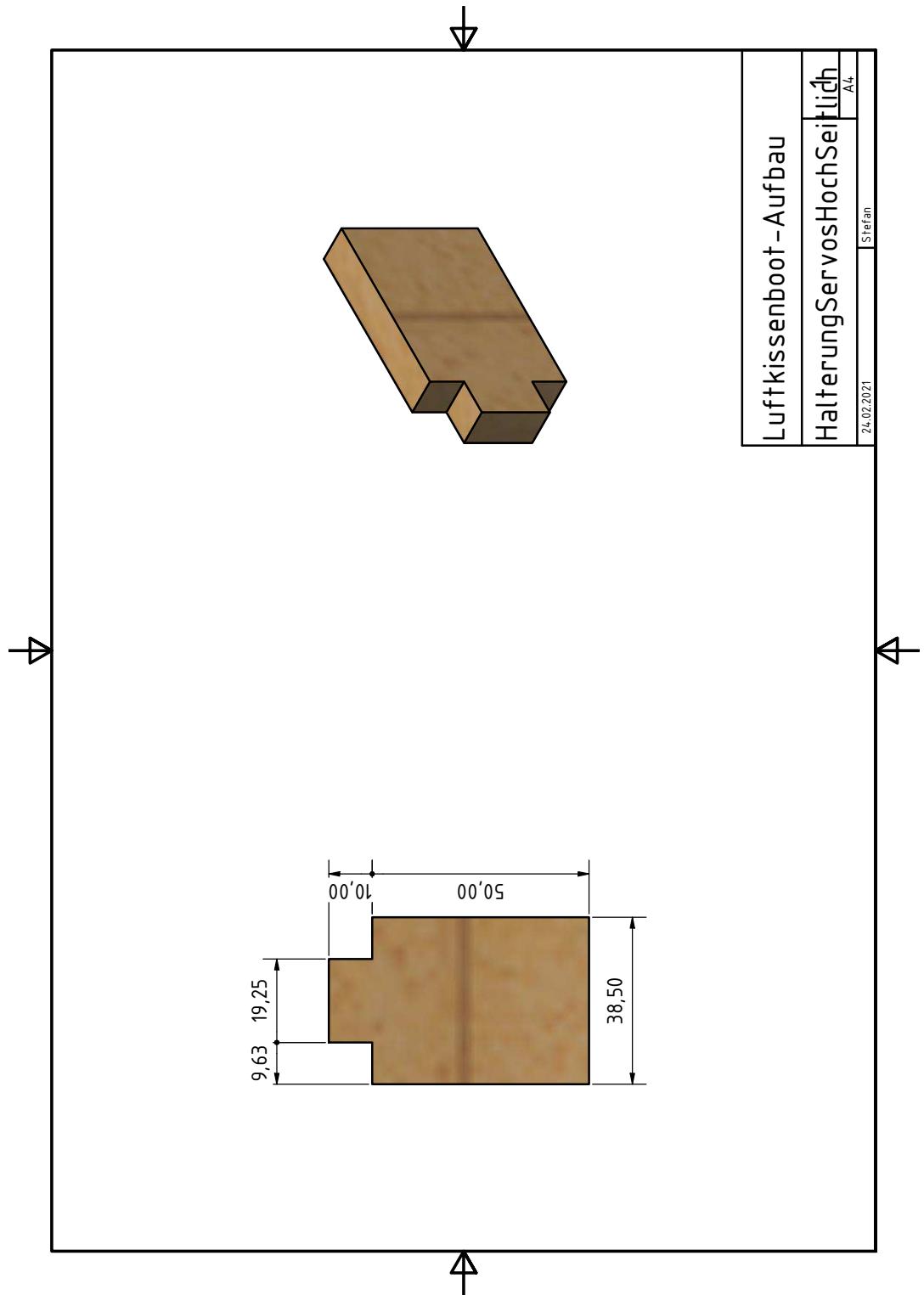


Abbildung 4.33: Zeichnung Servo Halter seitlich

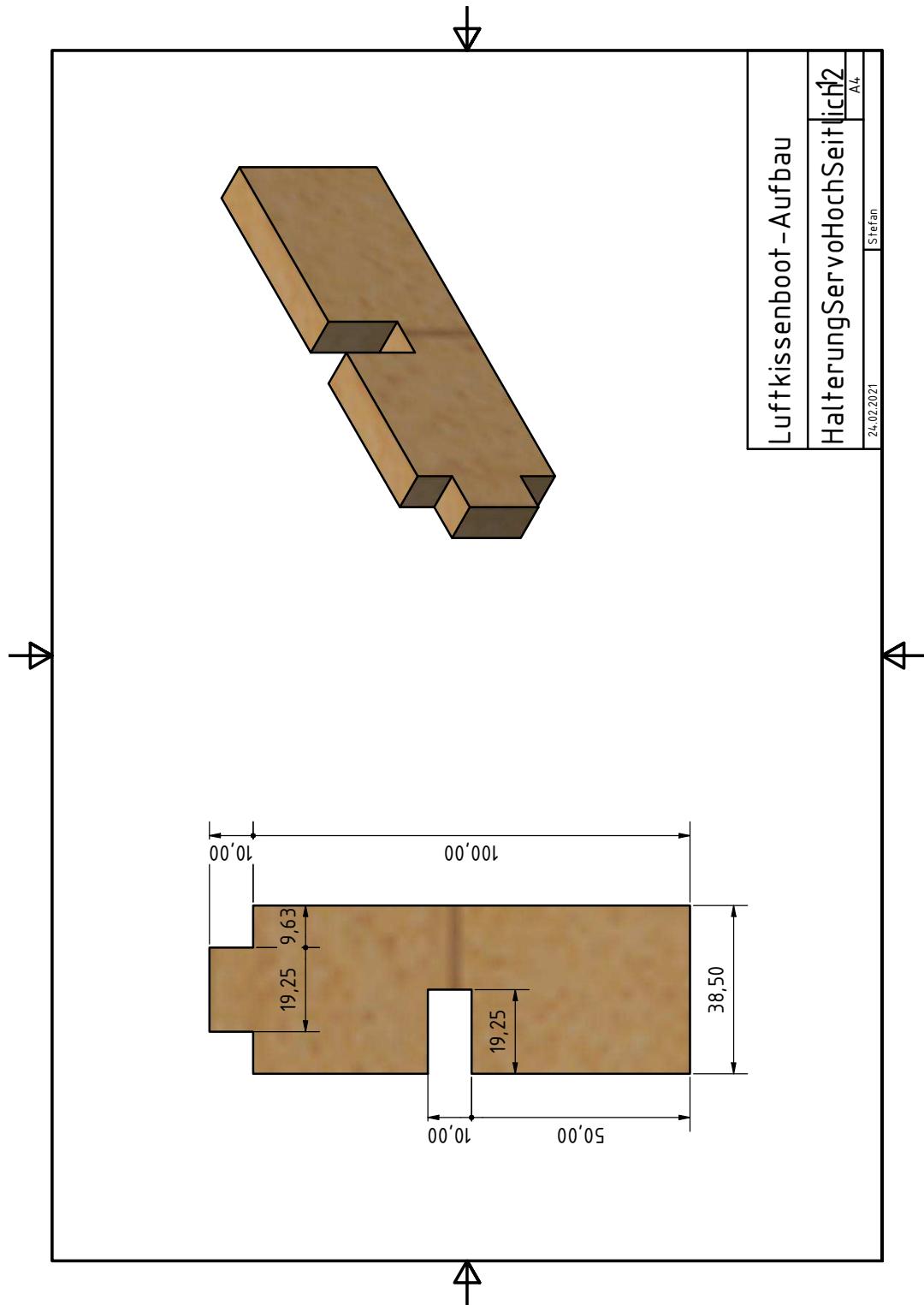


Abbildung 4.34: Zeichnung Servo Halter seitlich 2

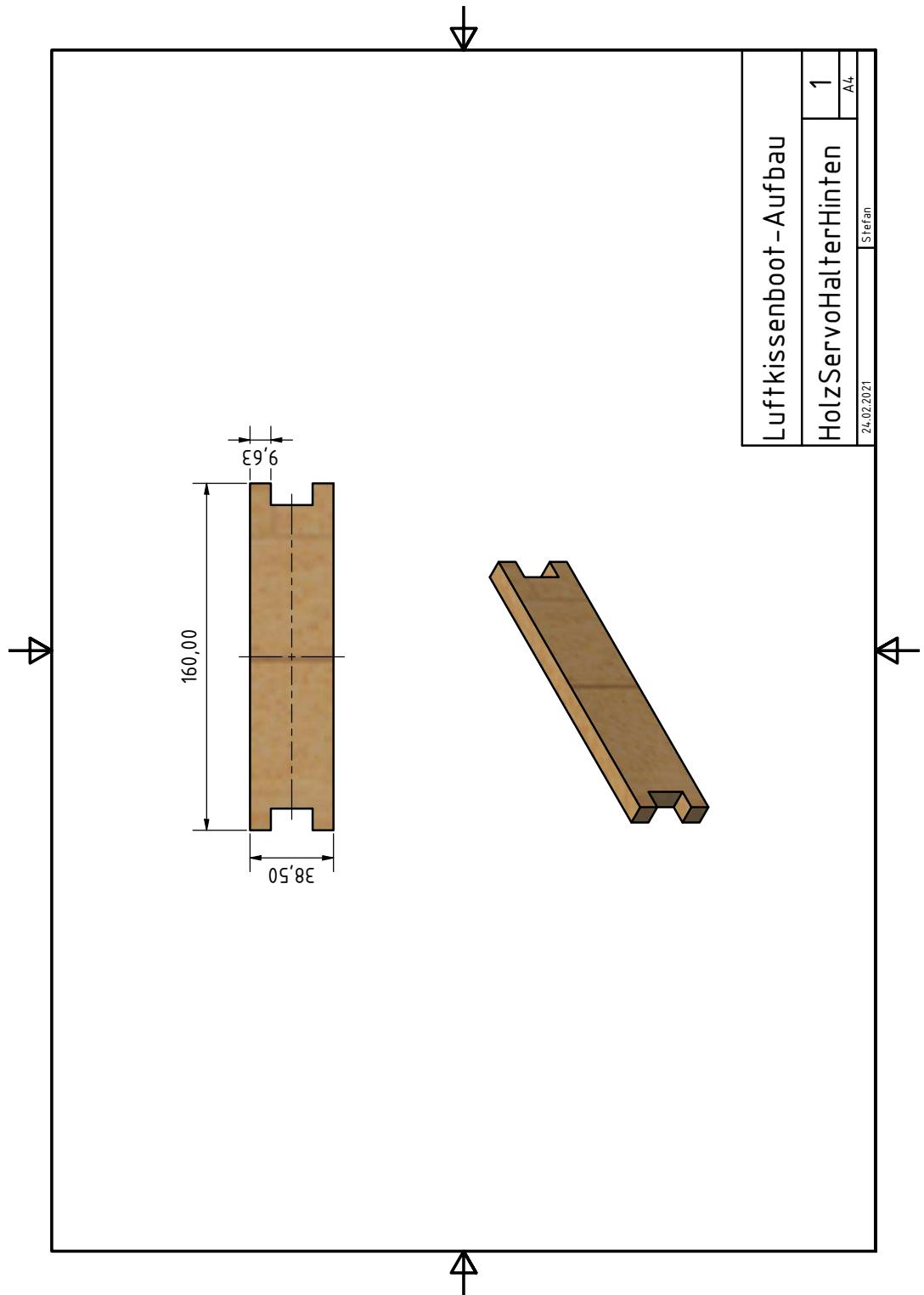


Abbildung 4.35: Zeichnung Servo Halter hinten

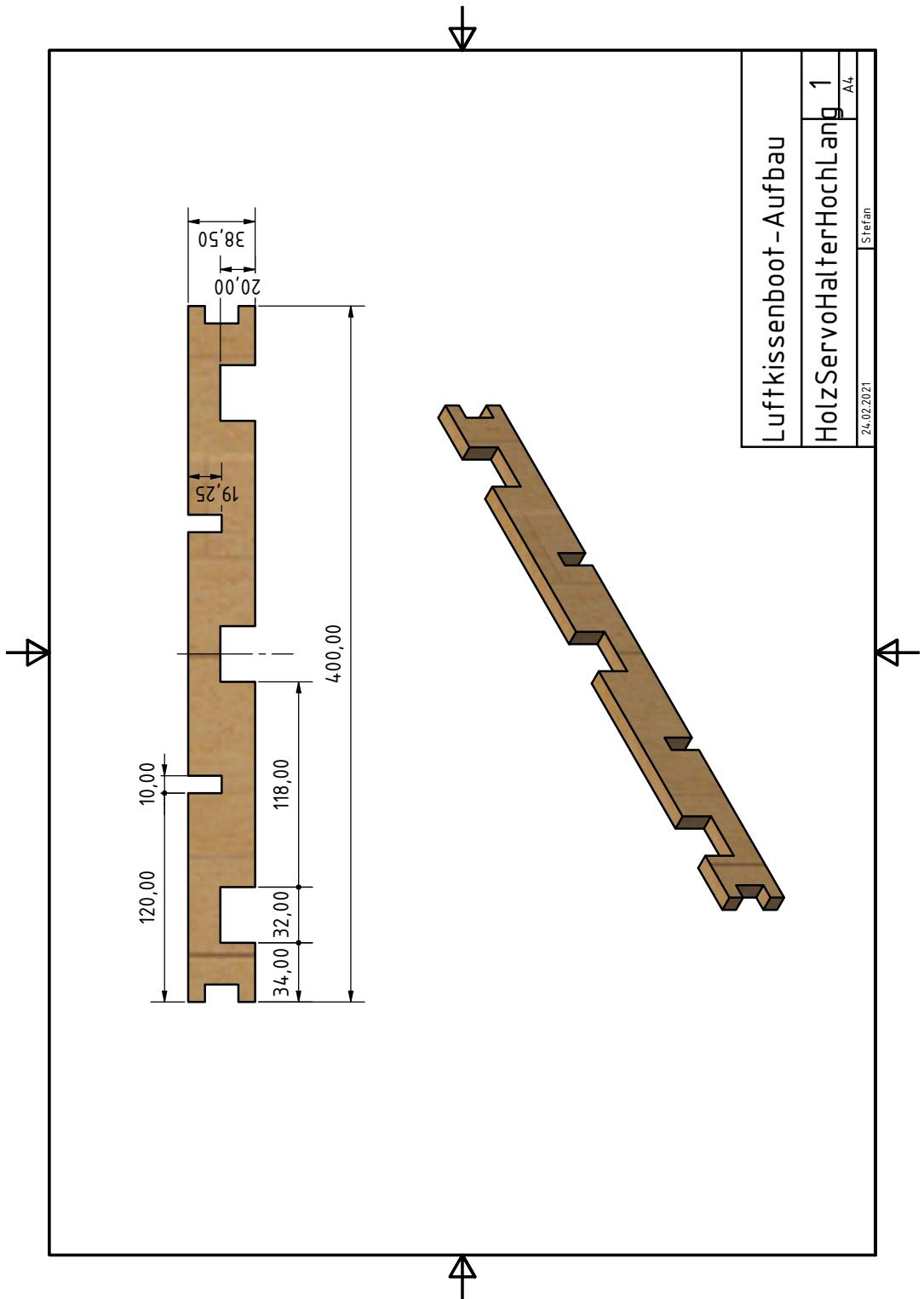


Abbildung 4.36: Zeichnung Servo Halter lang

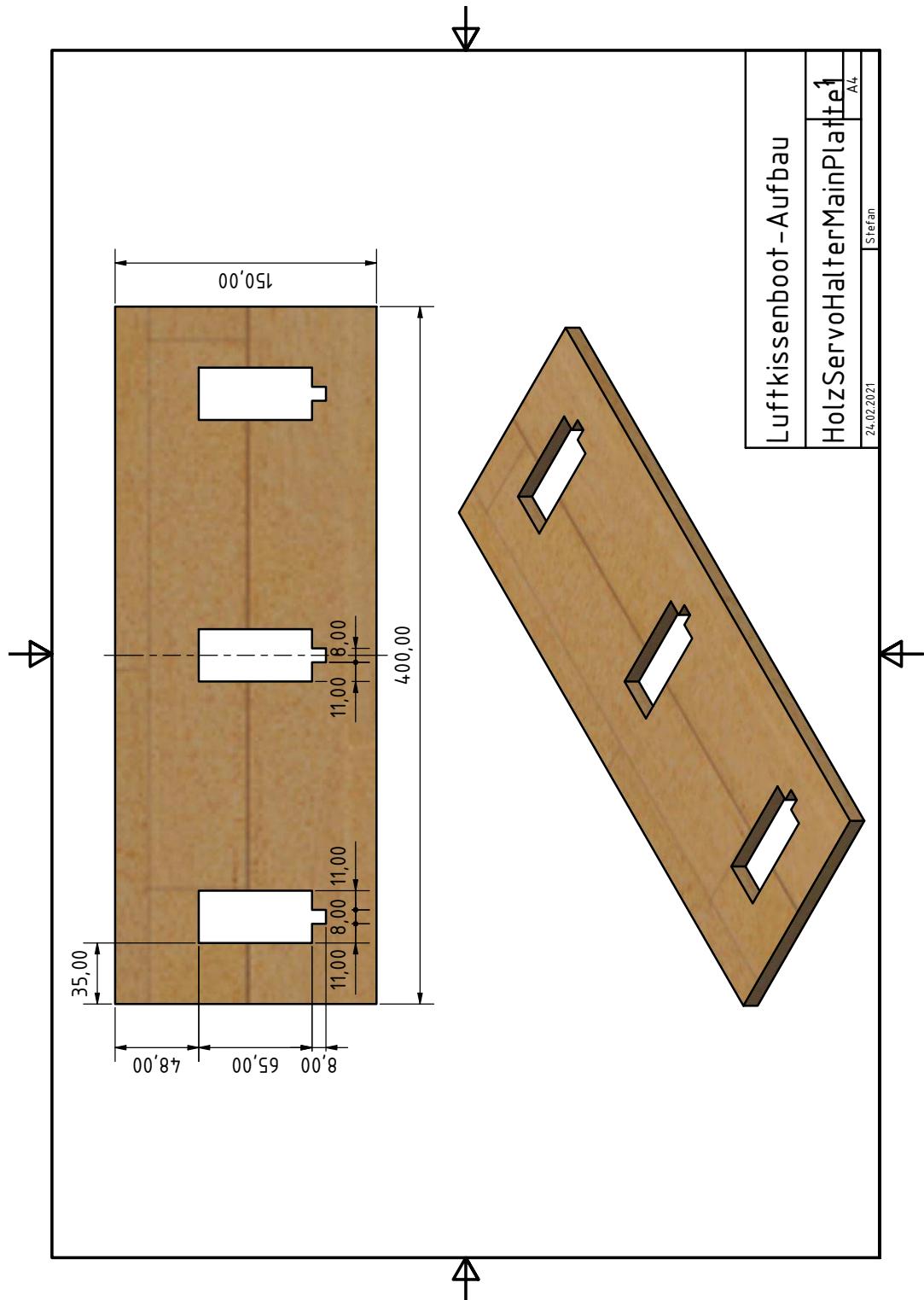


Abbildung 4.37: Zeichnung Servo Halter Hauptplatte

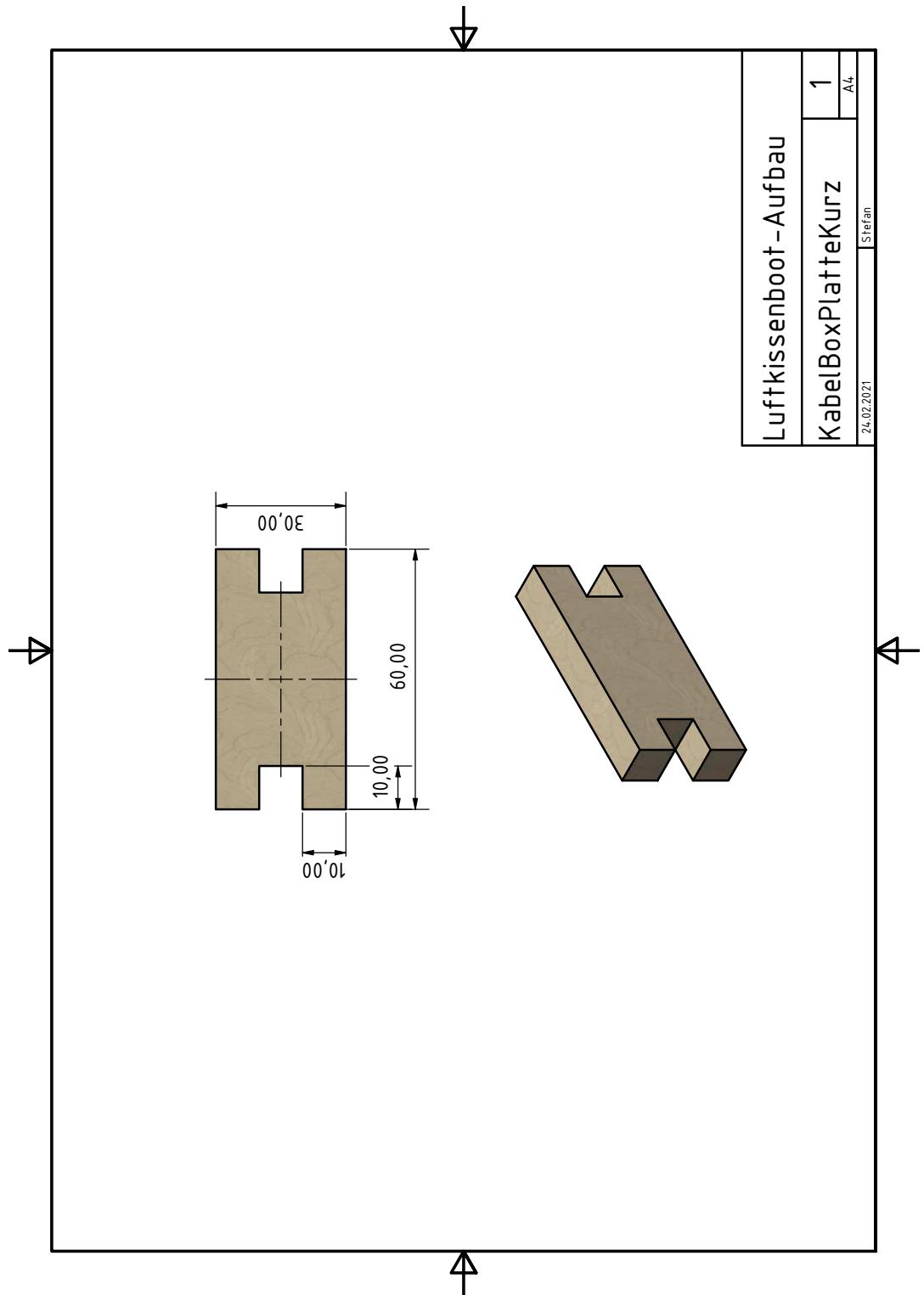


Abbildung 4.38: Zeichnung Kabelbox Platte kurz

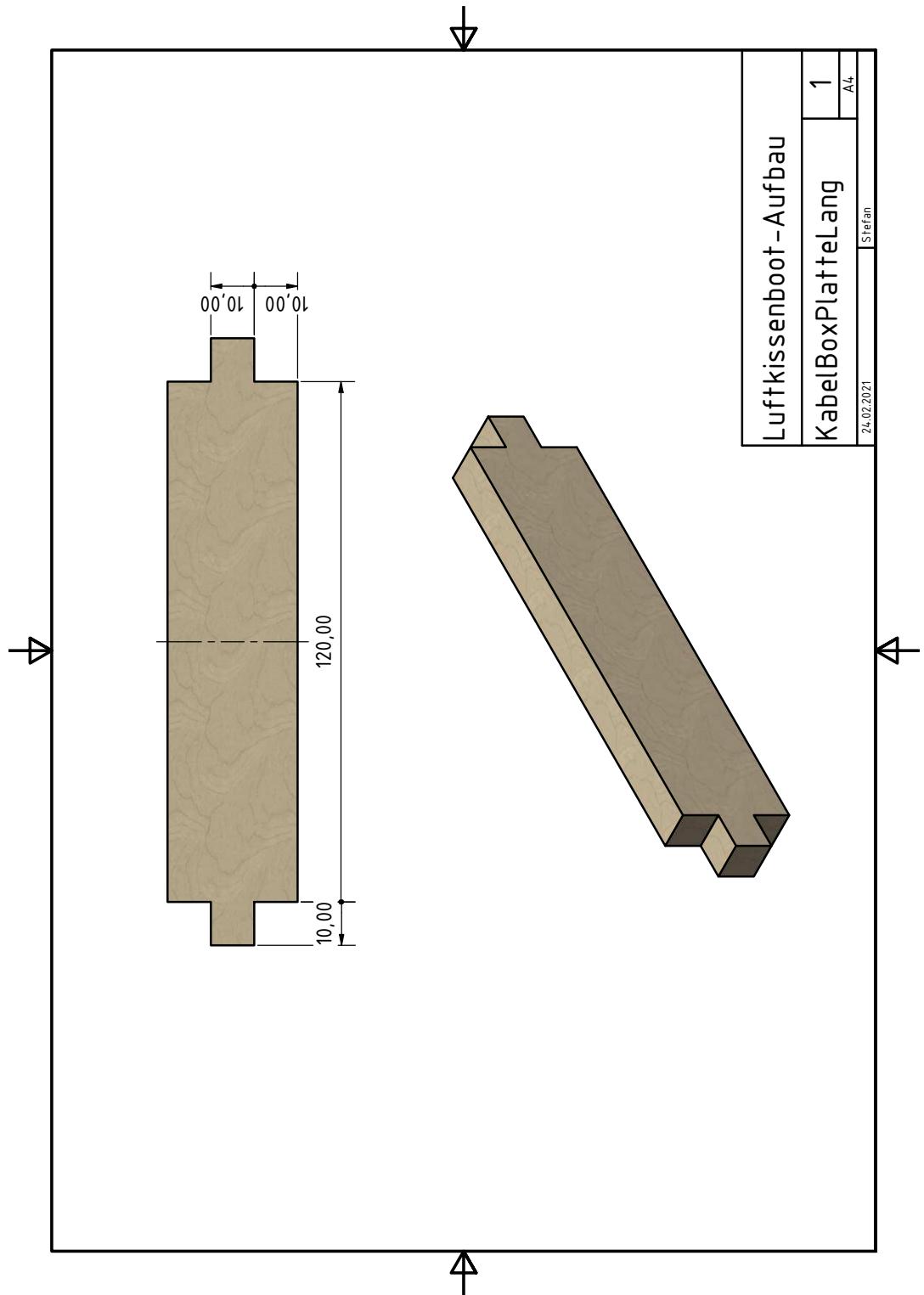


Abbildung 4.39: Zeichnung Kabelbox Platte lang

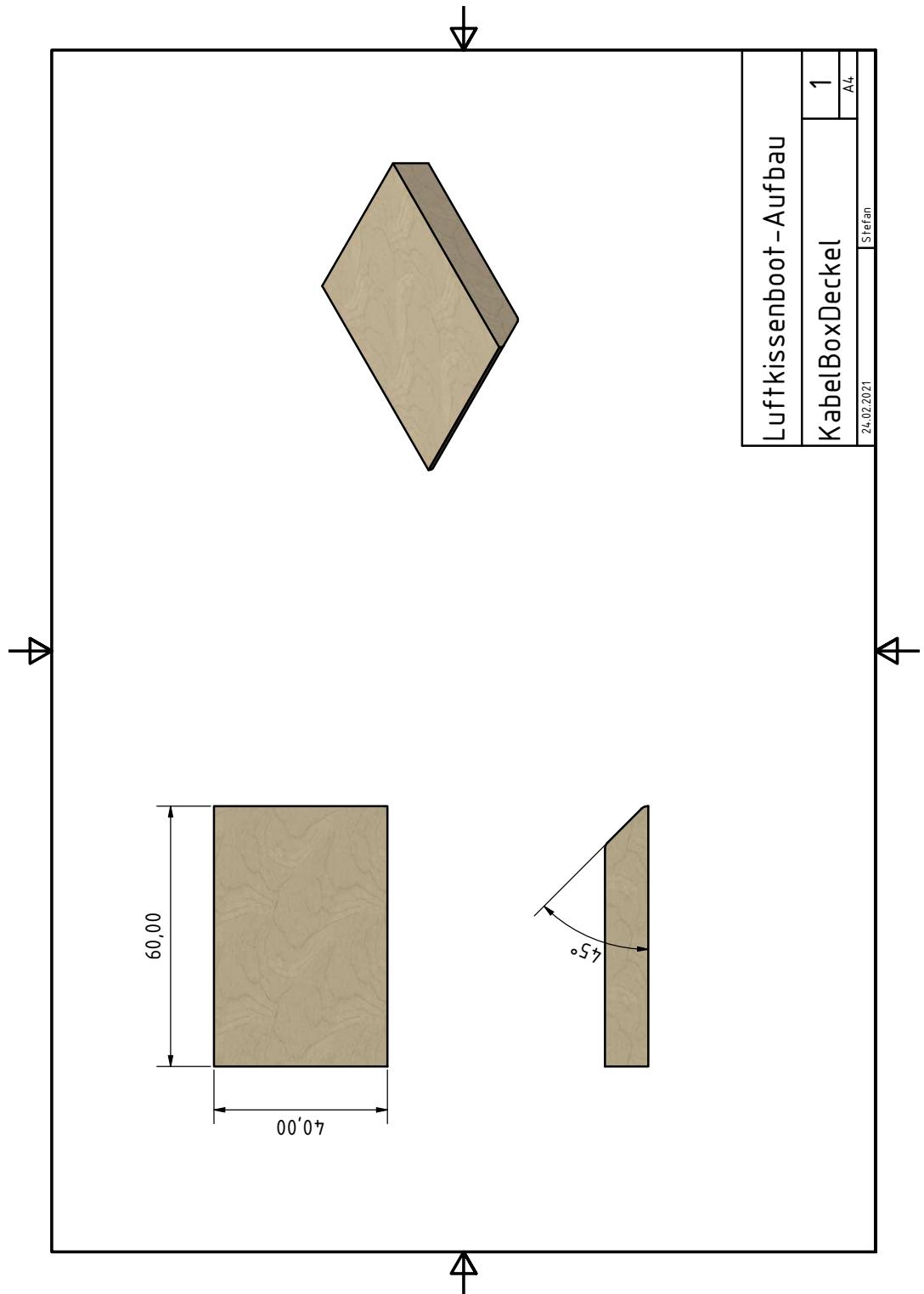


Abbildung 4.40: Zeichnung Kabelbox Deckel

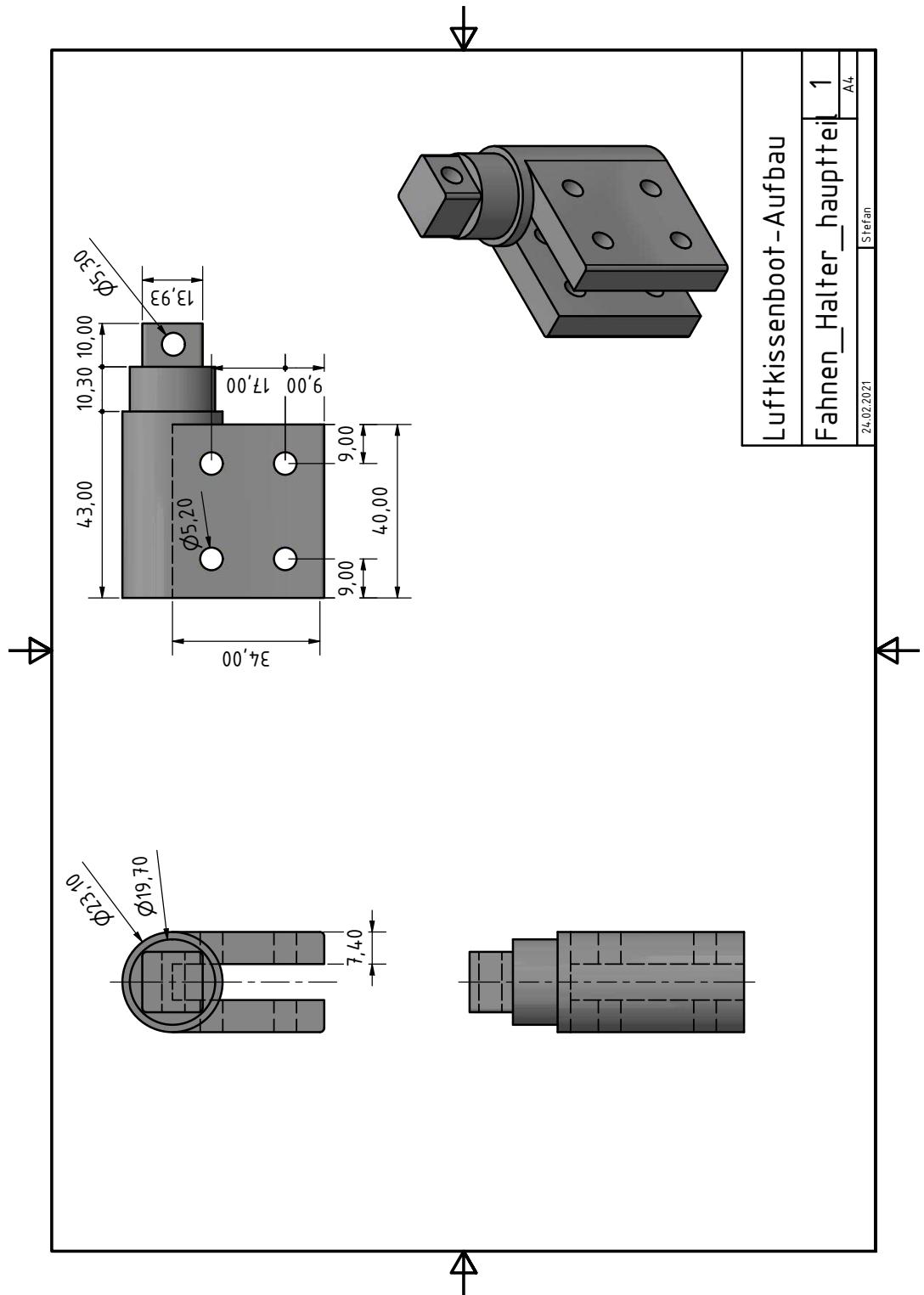


Abbildung 4.41: Zeichnung Fahnenhalter Hauptteil

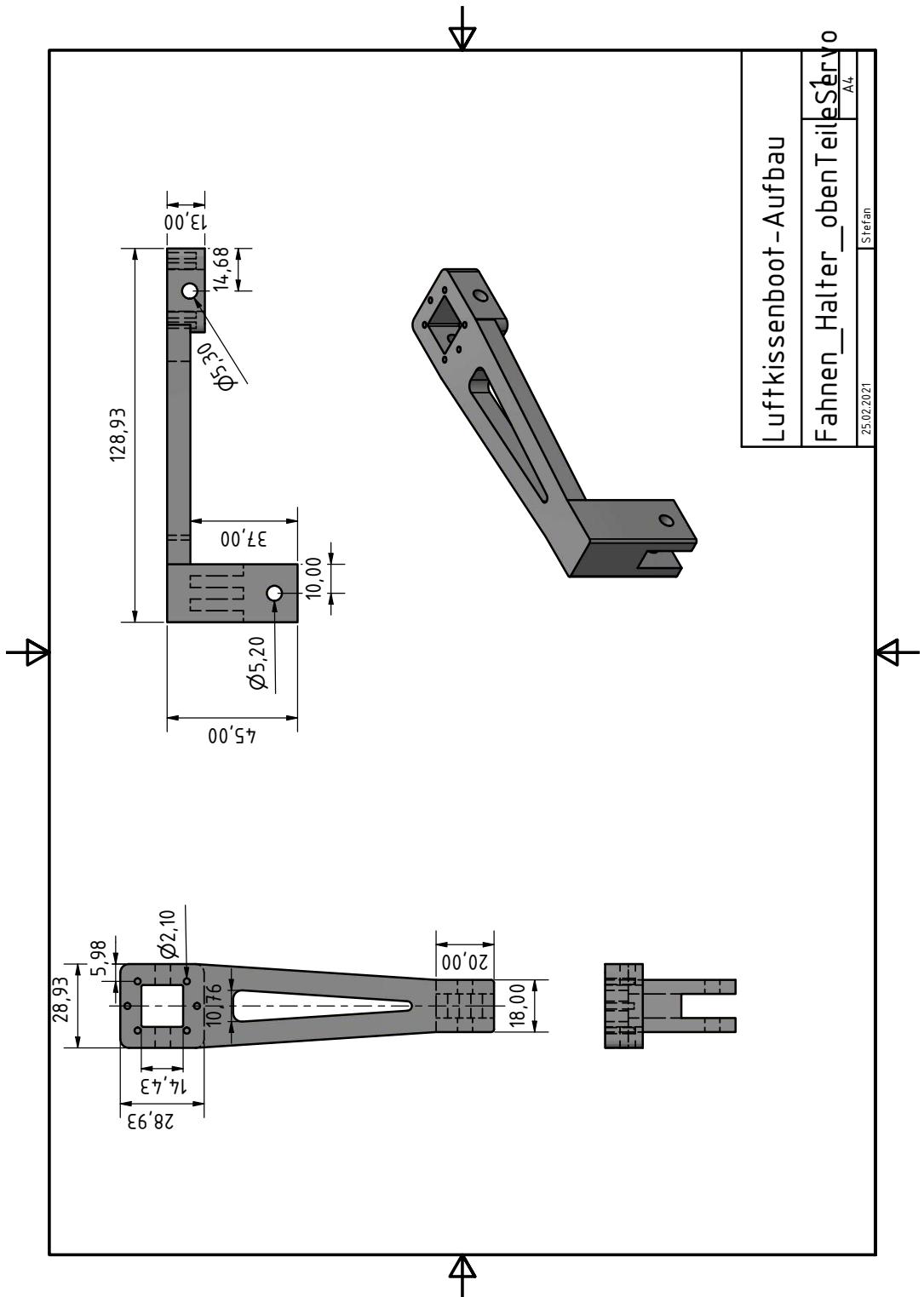


Abbildung 4.42: Zeichnung Fahnenhalter Servo

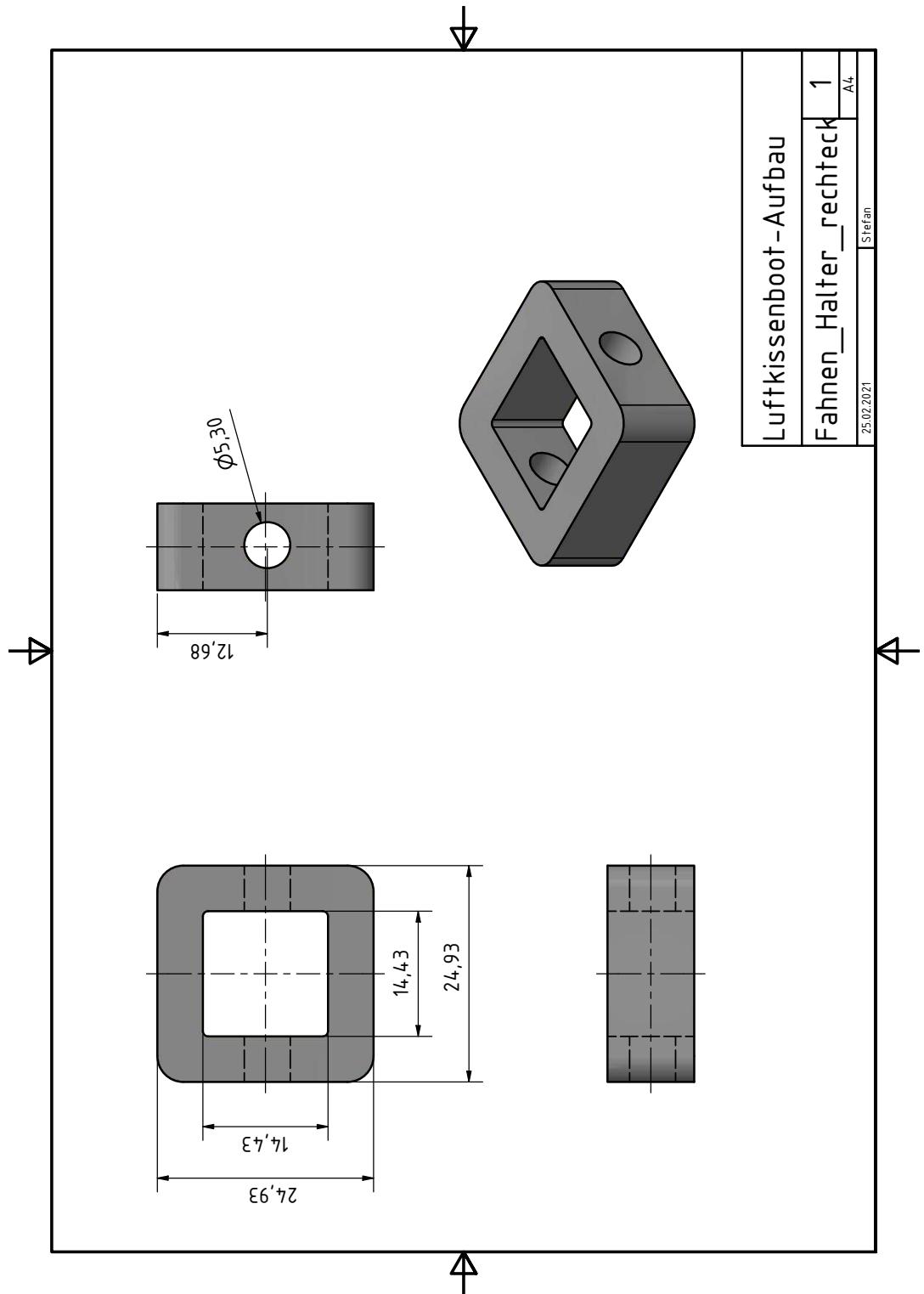


Abbildung 4.43: Zeichnung Fahnenhalter Rechteck

4.2.3 Zusammenbau

Der hintere Aufbau wurde, wie in Inventor geplant, aufgebaut und mit 16 Stück 70x70x55mm Winkelverbinder mit der Bodenplatte verbunden. Zusätzlich sind die schrägen Verstärkungen (Abbildung 4.31) mit je zwei Stück 50x50x35mm Winkelverbinder befestigt.

Die Winkelverbinder sind mit Holzschrauben an der Bodenplatte fix befestigt. Der hintere Aufbau wird mit zehn Stück M8 Schrauben an den Winkelverbündern festgeschraubt und ist somit relativ leicht abnehmbar. Die Löcher für die M8 Schrauben sind in den Zeichnungen nicht ersichtlich, da diese erst beim Zusammenbauen gebohrt wurden, um eine gute Passgenauigkeit zu garantieren.

Außerdem wurde ein Gitter (siehe Abbildung 4.44) montiert.



Abbildung 4.44: Zusammenbau hinterer Aufbau

4.3 Lenkung

Die Lenkung beinhaltet die komplette Steuerung des Hovercrafts.

Der Lenker selbst ist ein Fahrradlenker, welcher auf einer Metallstange befestigt wird. Diese Metallstange muss so stabil sein, damit sich der Fahrer an der Lenkung festhalten kann. Die Lenkerstellung wird mit einem Potenziometer, welches ebenfalls in der Box verbaut ist, ausgelesen.

An dem Fahrradlenker ist zusätzlich zu den beiden Daumengashebel noch ein Display montiert, um alle für den Fahrer wichtigen Informationen anzuzeigen.

Um den Lenker automatisch zu zentrieren und der Lenkung ein Gegenmoment zu geben, wurden Federn verwendet.

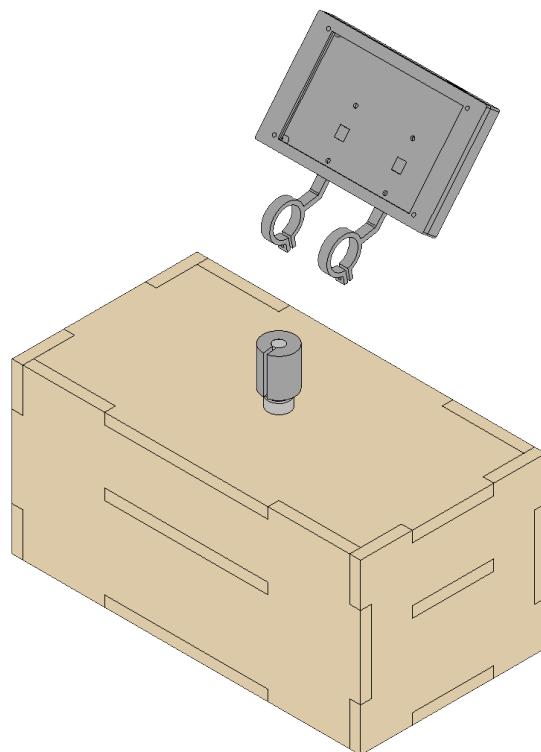


Abbildung 4.45: Lenkung 3D-Modell

4.3.1 Stückliste

Stk	Name	Material	Herstellung	Abbildung
1	Bodenplatte	10mm Pappel	LaserCutter	4.48
2	Platte groß mit Loch	10mm Pappel	LaserCutter	4.49
2	Platte Seite lang	10mm Pappel	LaserCutter	4.50
2	Platte Seite kurz	10mm Pappel	LaserCutter	4.51
1	Halterung Poti unten	PETG	3D-Drucker	4.51
1	Adapter Poti Teil 1	PETG	3D-Drucker	4.53
1	Adapter Poti Teil 2	PETG	3D-Drucker	4.54
1	Adapter Lenker	PETG	3D-Drucker	4.55
1	Stange Mitte	Ø10mm Alu-Stange	Einkauf	4.56
1	Gewindestange	M4 Gewindestange	Einkauf	4.57
2	Displayhalterung Halterung	PETG	3D-Drucker	4.58
1	Displayhalterung Hauptteil	PETG	3D-Drucker	4.59
1	Displayhalterung Deckel	PETG	3D-Drucker	4.60

Tabelle 4.3: Stückliste Aufbau hinten

4.3.2 Inventor

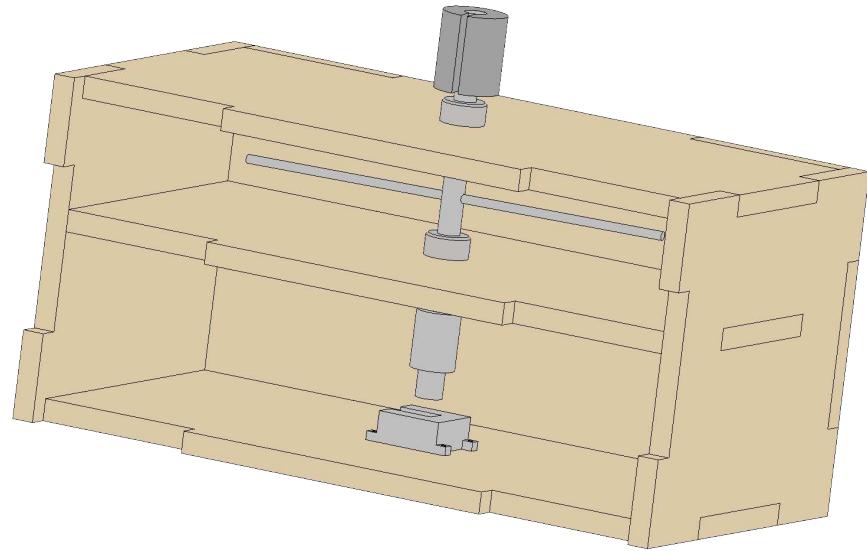


Abbildung 4.46: Lenkung Ansicht vorne, offen

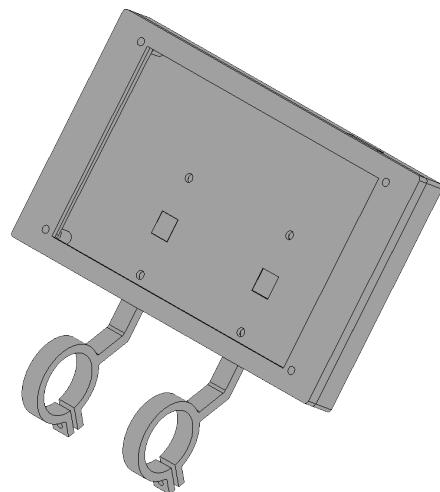


Abbildung 4.47: Lenkung Ansicht Display-Halterung

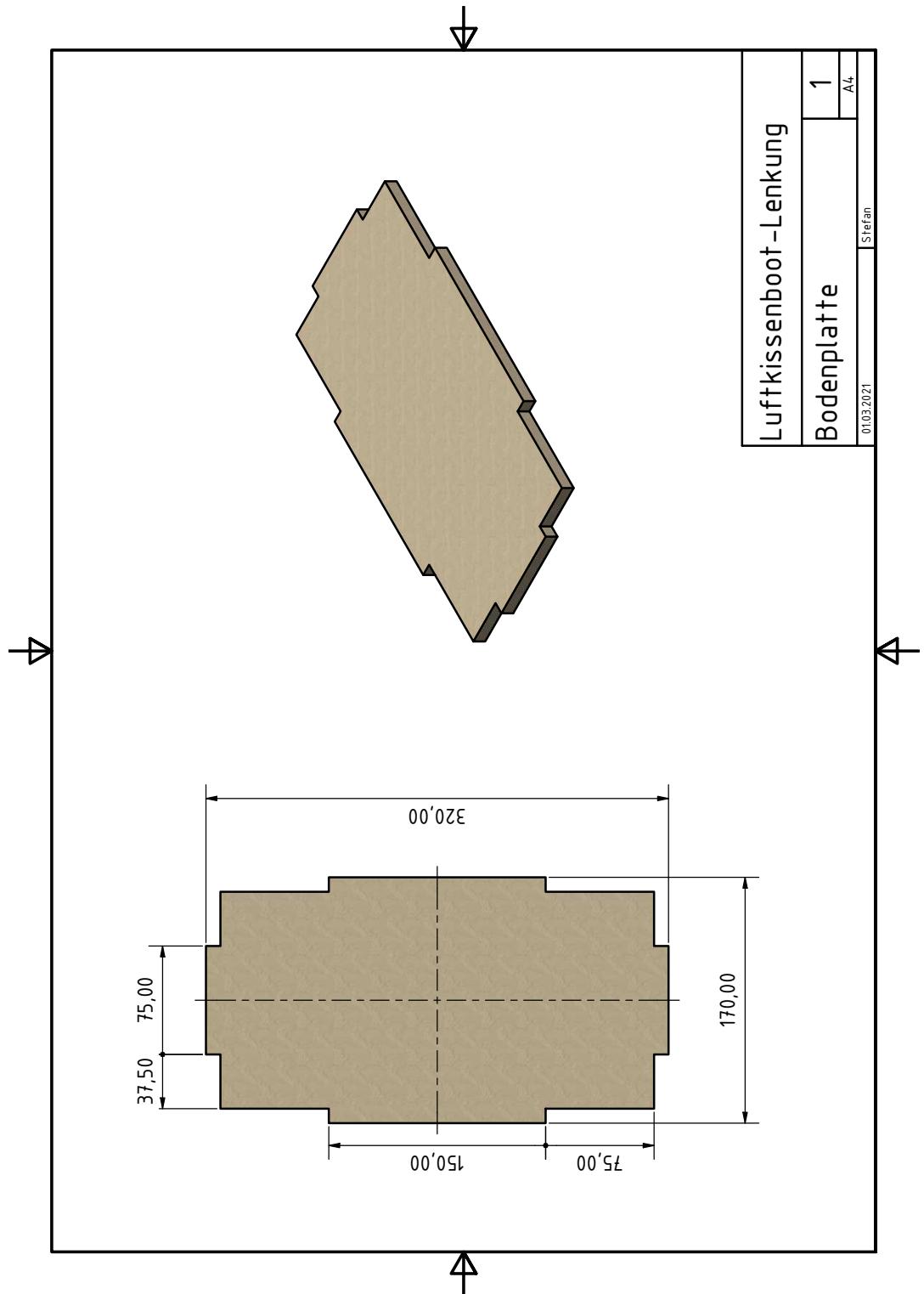


Abbildung 4.48: Zeichnung Bodenplatte

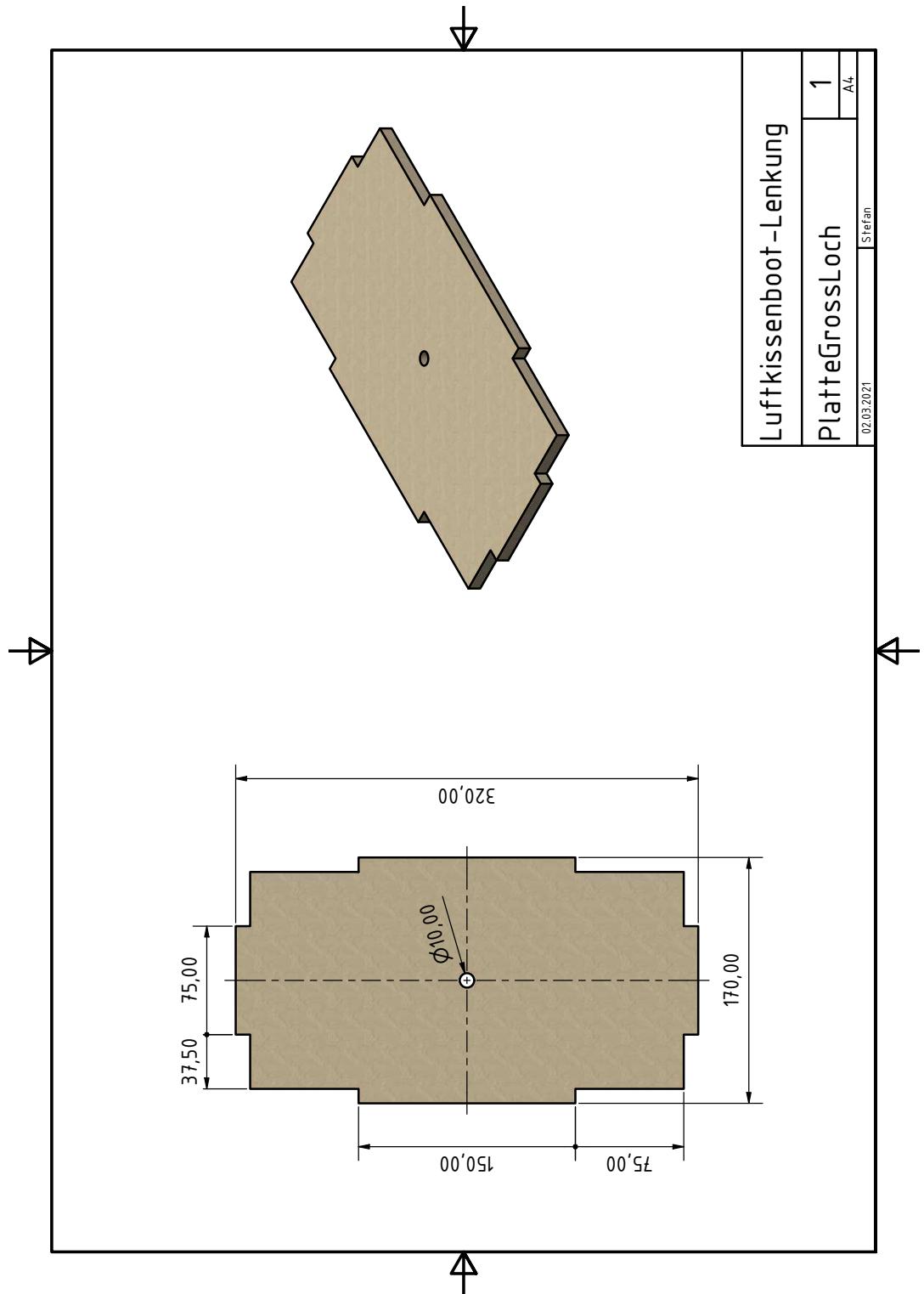


Abbildung 4.49: Zeichnung Platte groß mit Loch

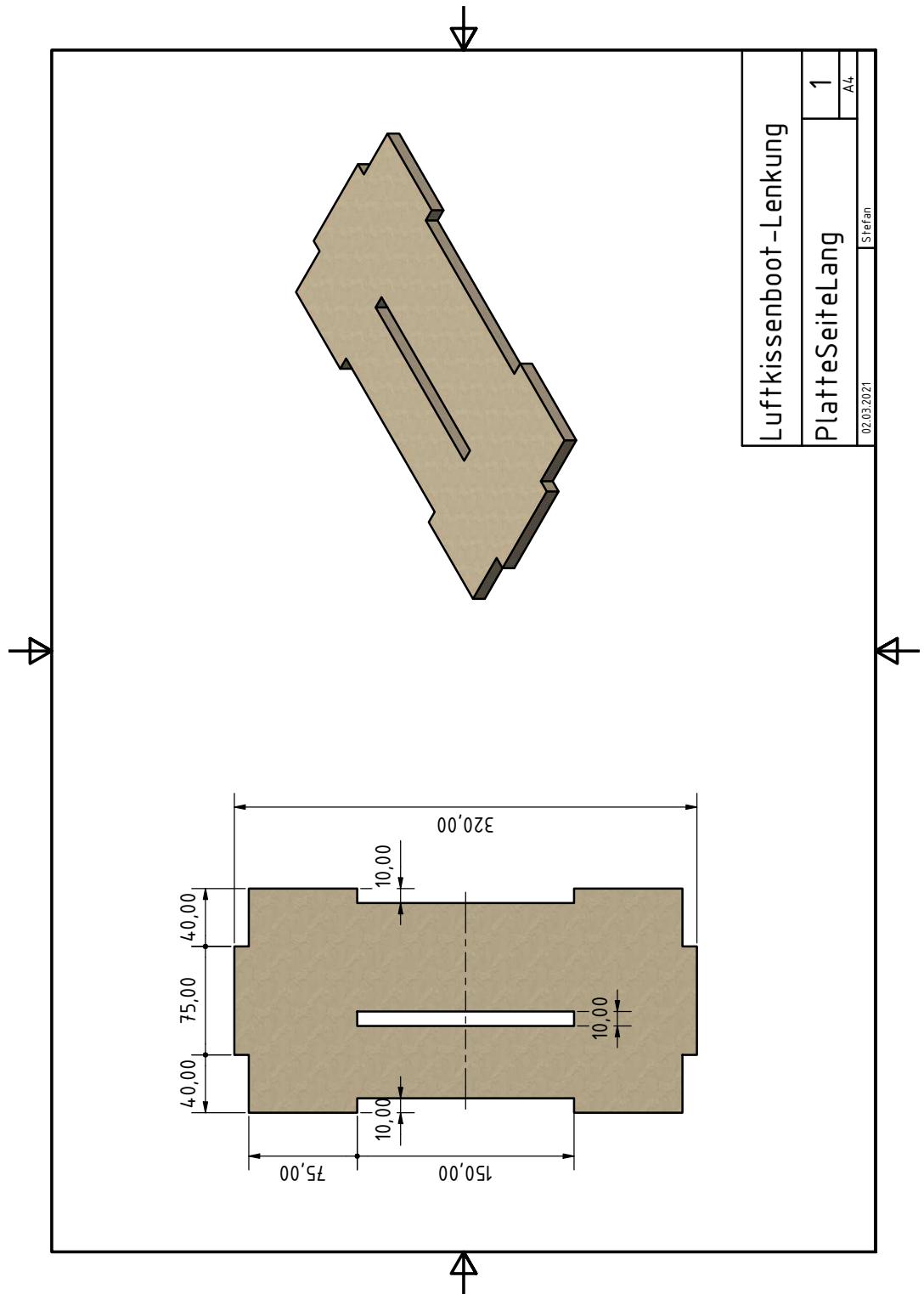


Abbildung 4.50: Zeichnung Platte Seite lang

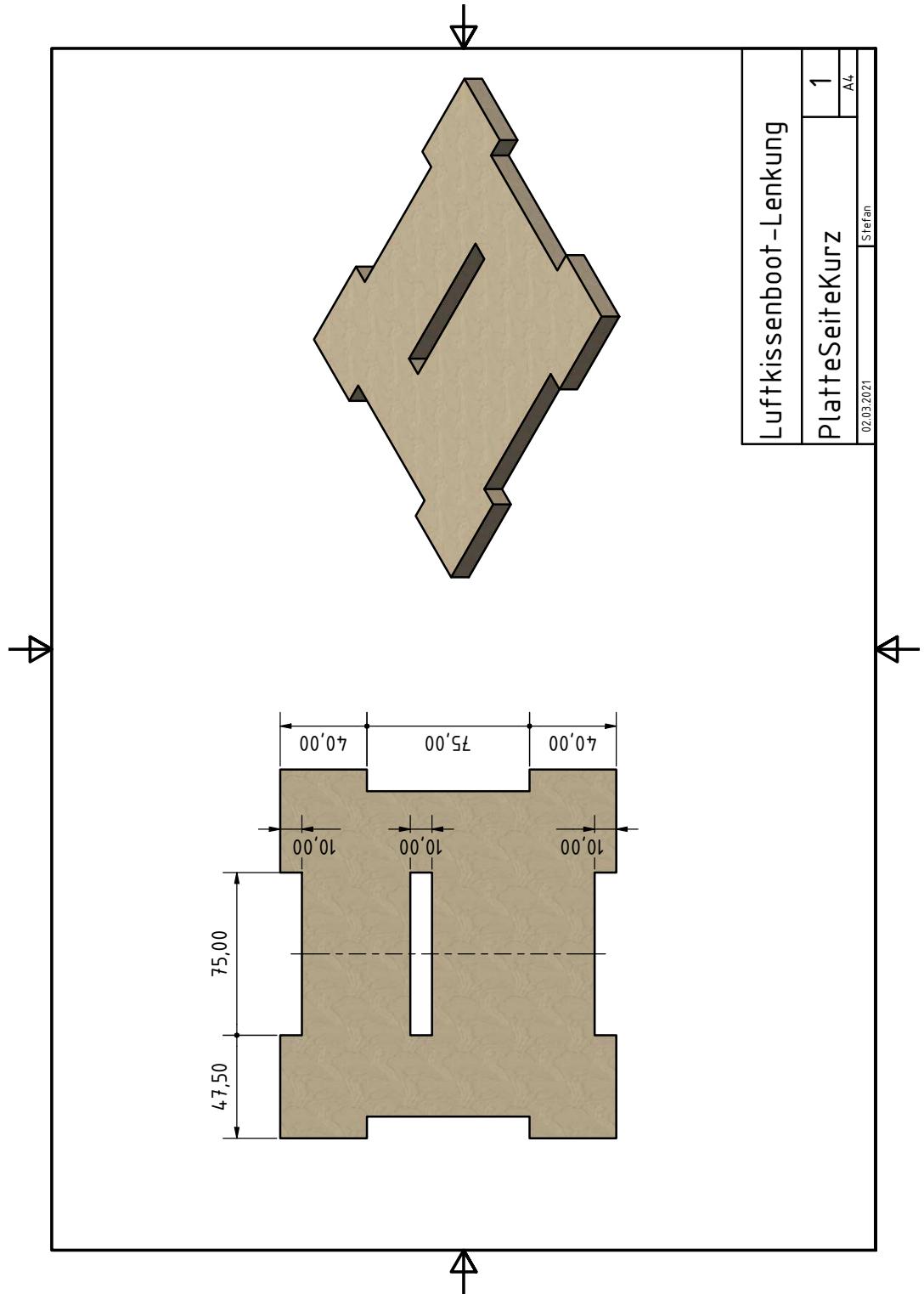


Abbildung 4.51: Zeichnung Platte Seite kurz

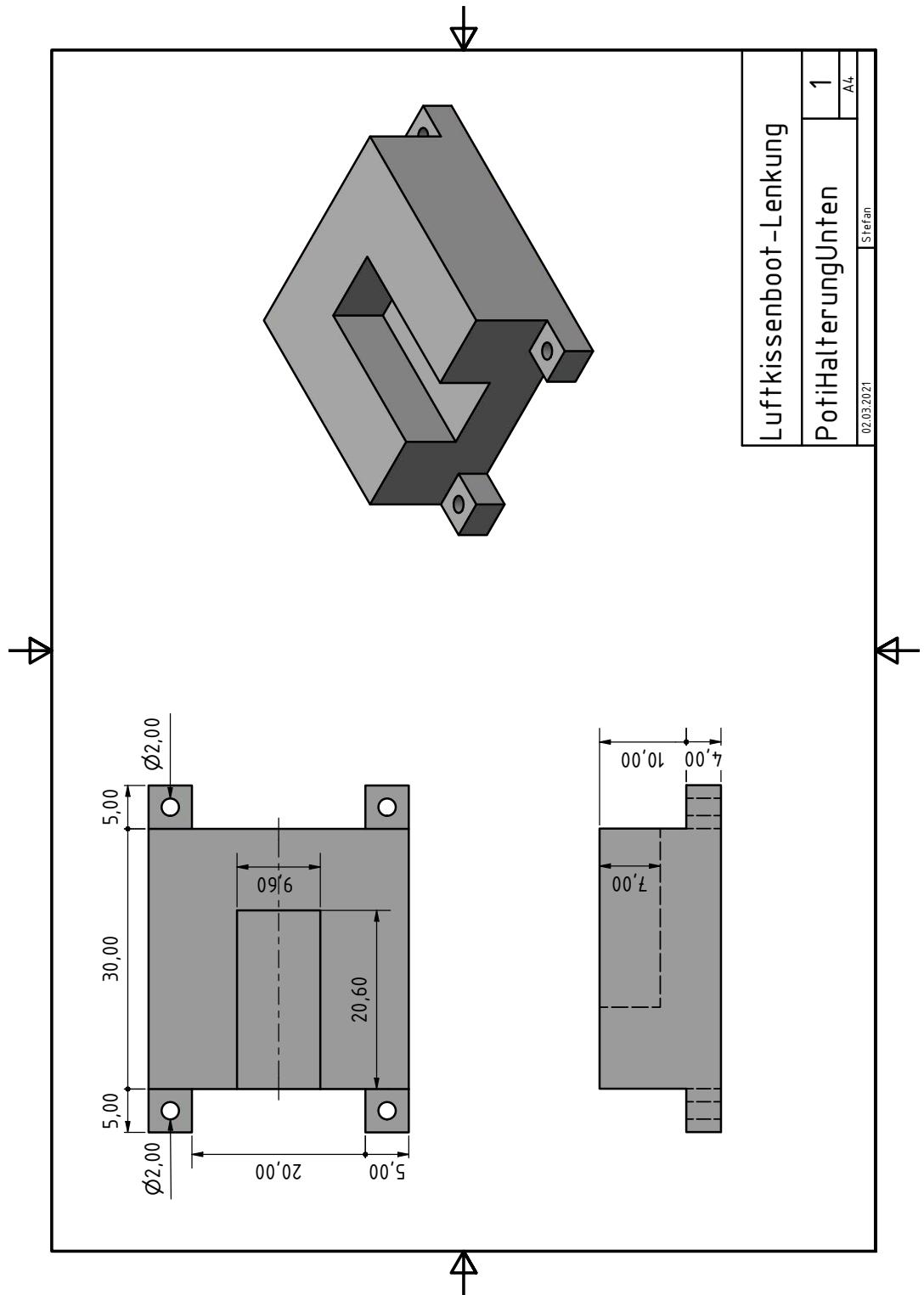


Abbildung 4.52: Zeichnung Poti Halterung

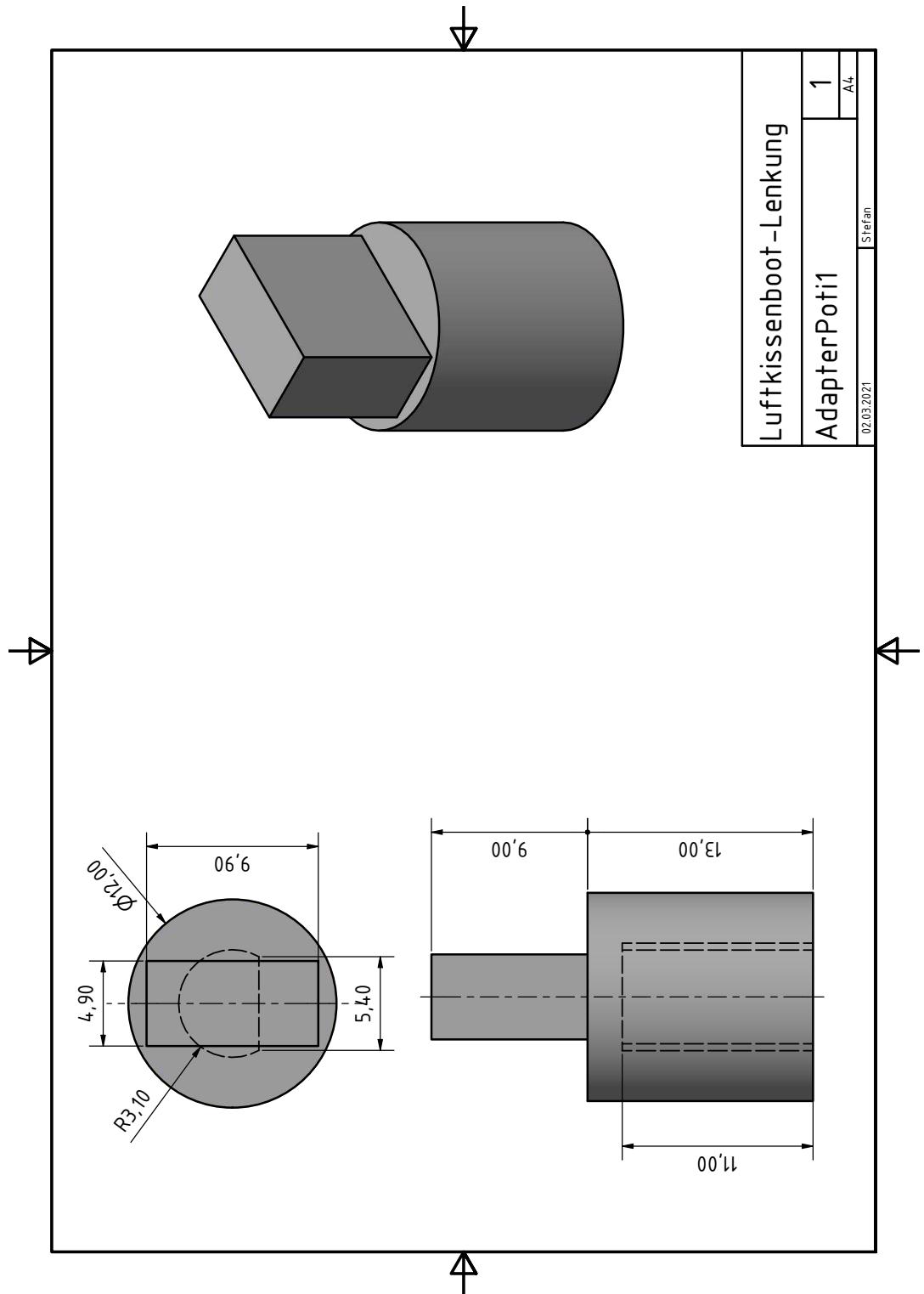


Abbildung 4.53: Zeichnung Adapter Poti Teil 1

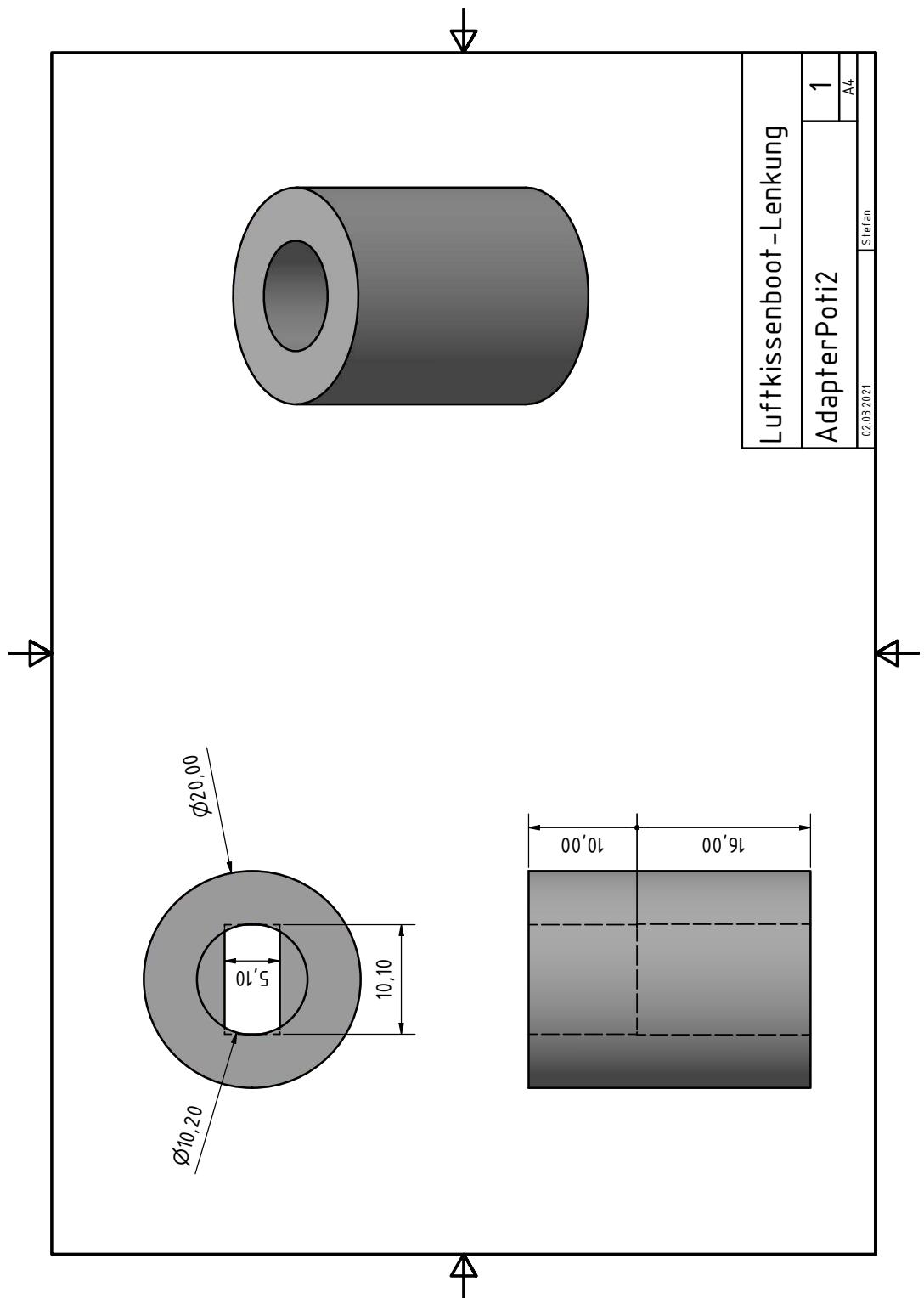


Abbildung 4.54: Zeichnung Adapter Poti Teil 2

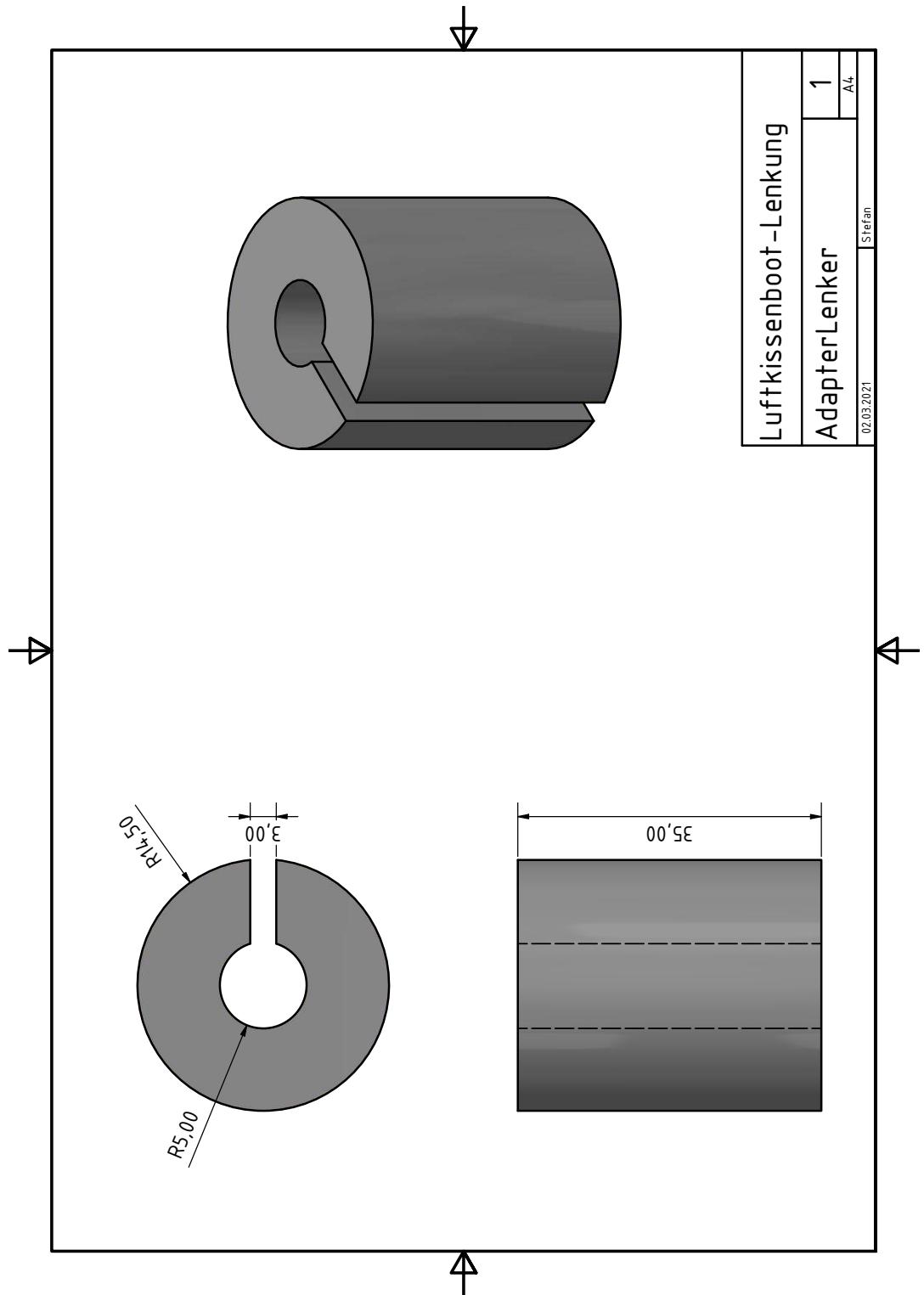


Abbildung 4.55: Zeichnung Adapter Lenker

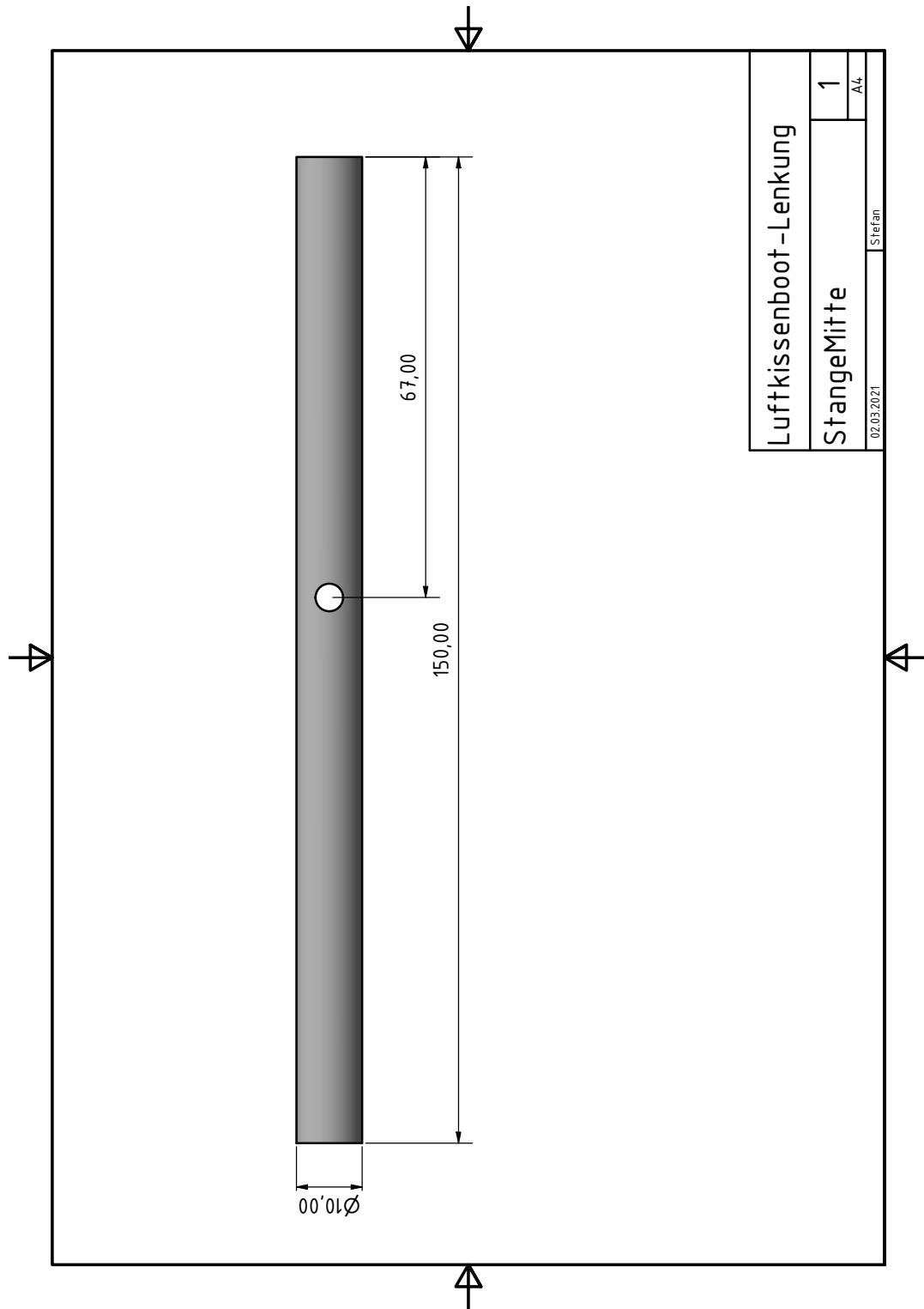


Abbildung 4.56: Zeichnung Stange Mitte

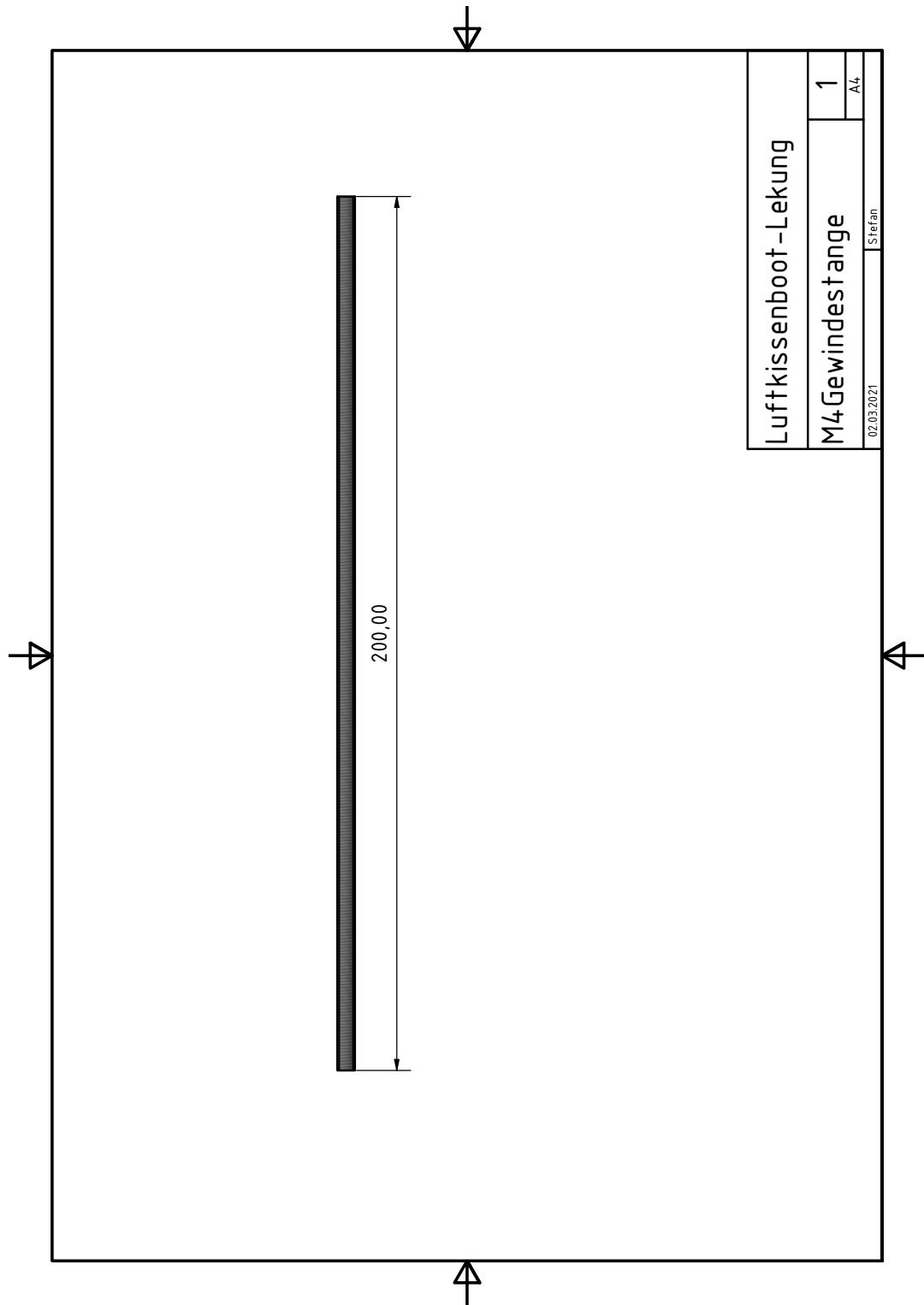


Abbildung 4.57: Zeichnung Gewindestange

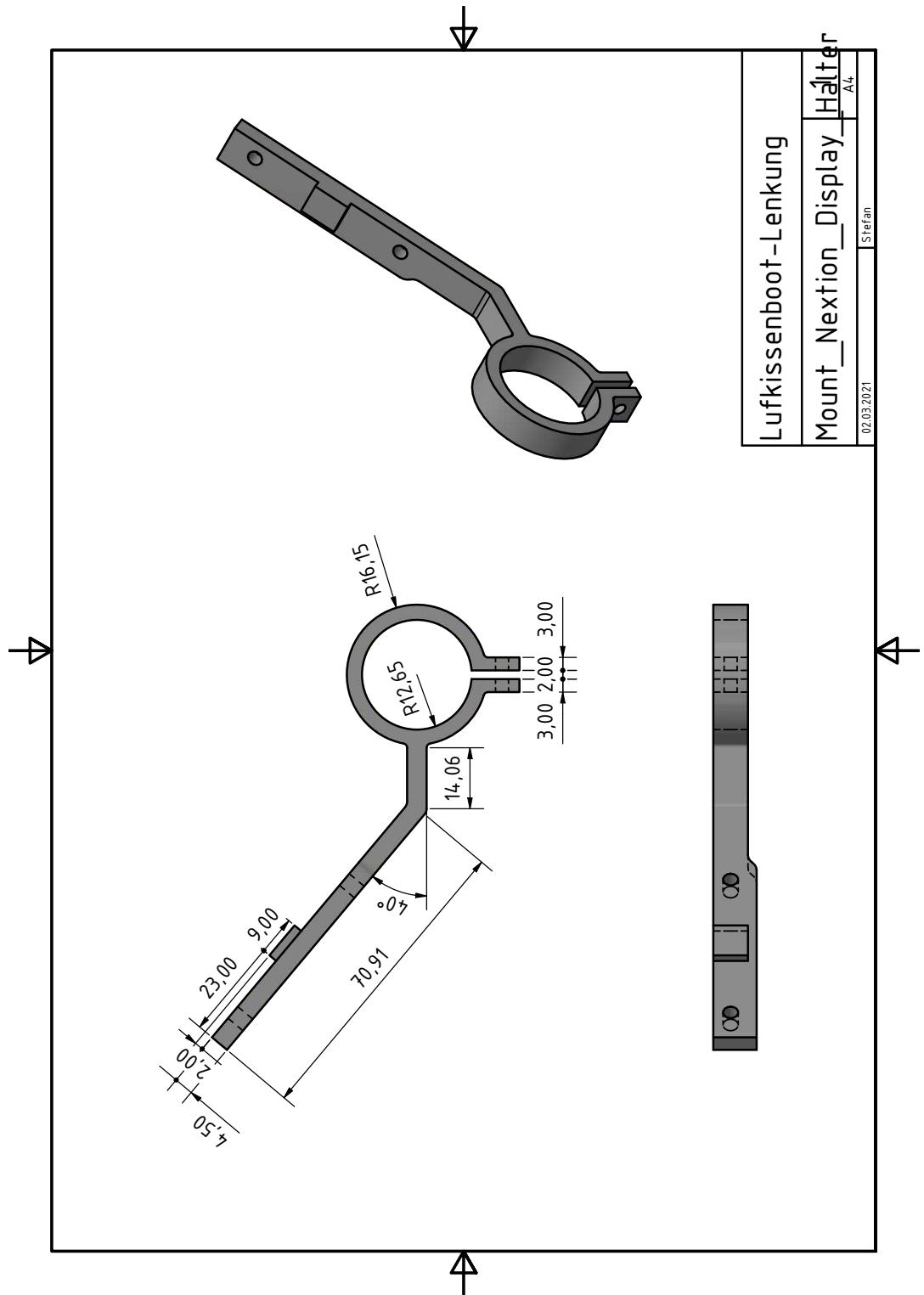


Abbildung 4.58: Zeichnung Displayhalterung Halterung

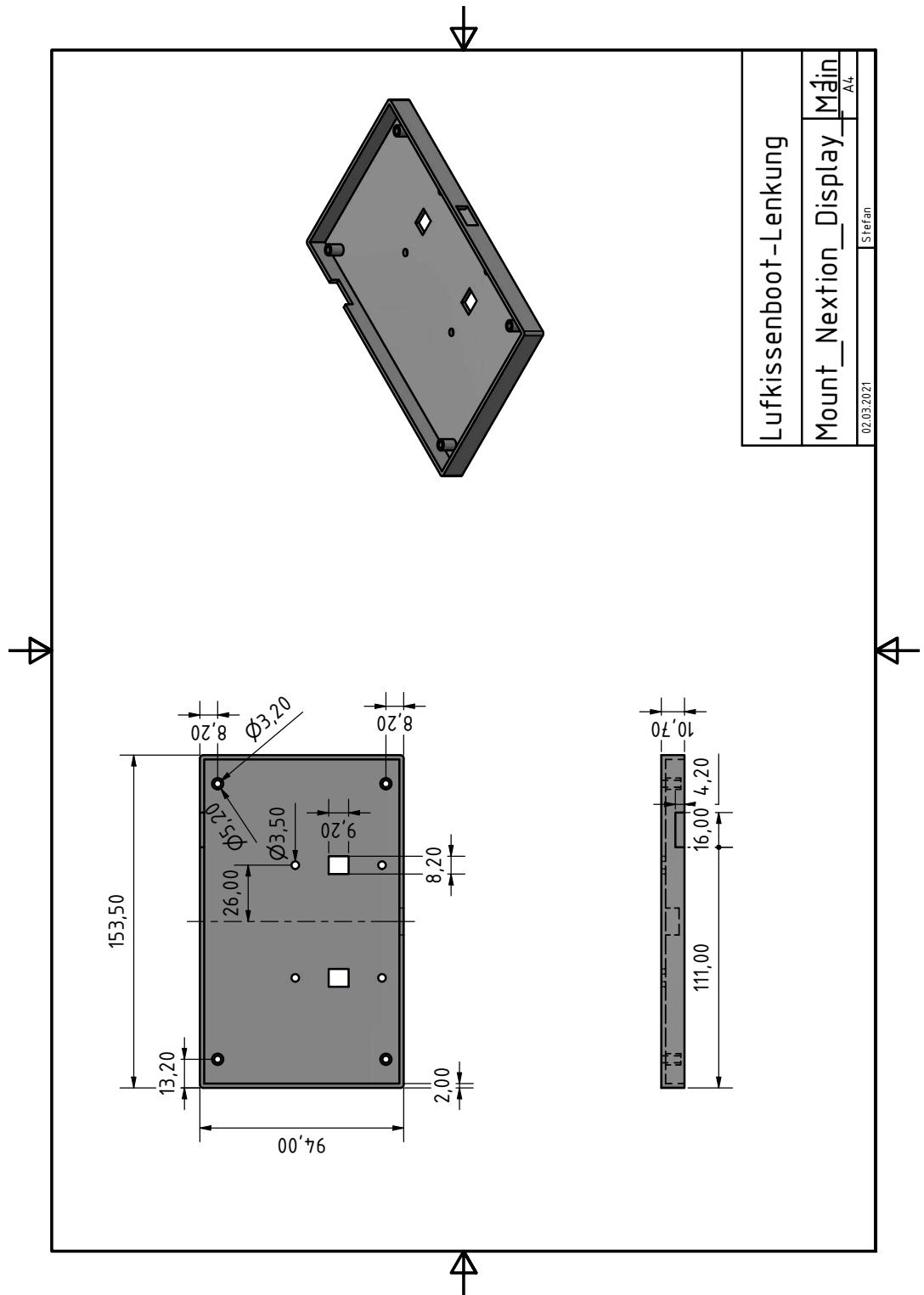


Abbildung 4.59: Zeichnung Displayhalterung Hauptteil

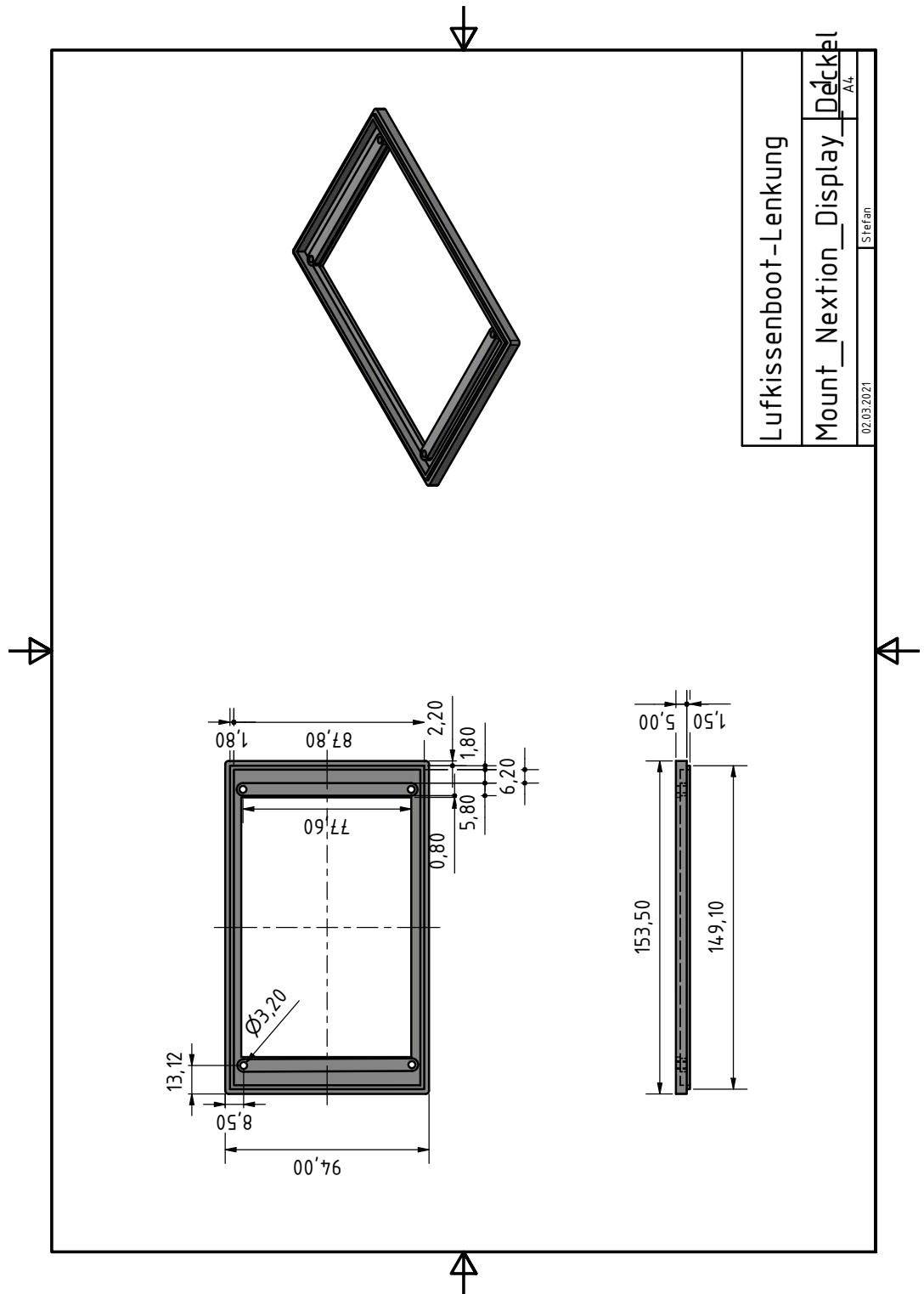


Abbildung 4.60: Zeichnung Displayhalterung Deckel

4.3.3 Zusammenbau

Die Lenkung wurde, wie in Inventor geplant, zusammengebaut. Die Löcher für den Not-Aus, den Precharge-Taster und die Kabeln des Displays und der Daumengashebel wurden gebohrt.

Um die 4 Federn, welchen den Lenker zentrieren, befestigen und spannen zu können, wurden M6 Schrauben verwendet – siehe Abbildung 4.61.

Die Lenkung ist wie der hintere Aufbau ebenfalls mit vier Stück 50x50x35mm Winkelverbinder an der Bodenplatte befestigt.



Abbildung 4.61: Foto Federsystem Lenkung



Abbildung 4.62: Zusammenbau Lenkung

4.4 Bestellliste

Name	Stück	Händler	Artikel Nr.
Fahrradlenker	1	Amazon	Link
Fahrradlenker-Griffe	1	Amazon	Link
Fahrradlenker Verlängerung	1	Amazon	Link
Winkelverbinder 70x70x55mm	16	Obi	9967308
Winkelverbinder 50x50x35mm	12	Obi	9967035
Federn	4	Obi	7624166
M6 Einschlagmuttern	4	Obi	7606379
Pattex Kraftkleber 650g	1	Hornbach	2560329

Tabelle 4.4: Bestellliste Konstruktion

Anmerkung:

Die elektrischen Teile des Hovercrafts, wie zum Beispiel das Display und der Not-Aus-Schalter, sind in der Bestelliste unter Tabelle 6.2 auf Seite 117 zu finden.

4.5 Gesamter Zusammenbau

Die Konstruktion wurde, wie in den Unterkapiteln bereits ausführlich beschrieben, umgesetzt.

Zusätzlich wurden Sticker mit den Logos der Sponsoren angebracht.



Abbildung 4.63: Gesamter Zusammenbau

5 Leistungselektronik

5.1 Antrieb

5.1.1 Motoren

Angetrieben wird das Luftkissenboot von zwei **Hacker A200-6** Elektromotoren mit einer Dauerleistung von jeweils 10 kW.



Abbildung 5.1: Hacker A200 Elektromotor

Konkret weisen die Motoren folgende Leistungsdaten auf:

Parameter	Wert
Leistung	10 kW (15 kW max 15 sec.)
Leerlaufstrom (8.4 V)	5.1 A
Innenwiderstand	0.011 Ω
Polzahl	20
empf. Timing	22°
Leerlaufdrehzahl	151 U/min ⁻¹ /V

Tabelle 5.1: Hacker A200 Leistungsdaten

5.1.2 Propeller

Zur Erzeugung des Vorschubes

Hierfür kommt ein zwei blättriger XOAR Elektro Flugzeug-Propeller 28 x 12 Zoll zum Einsatz.

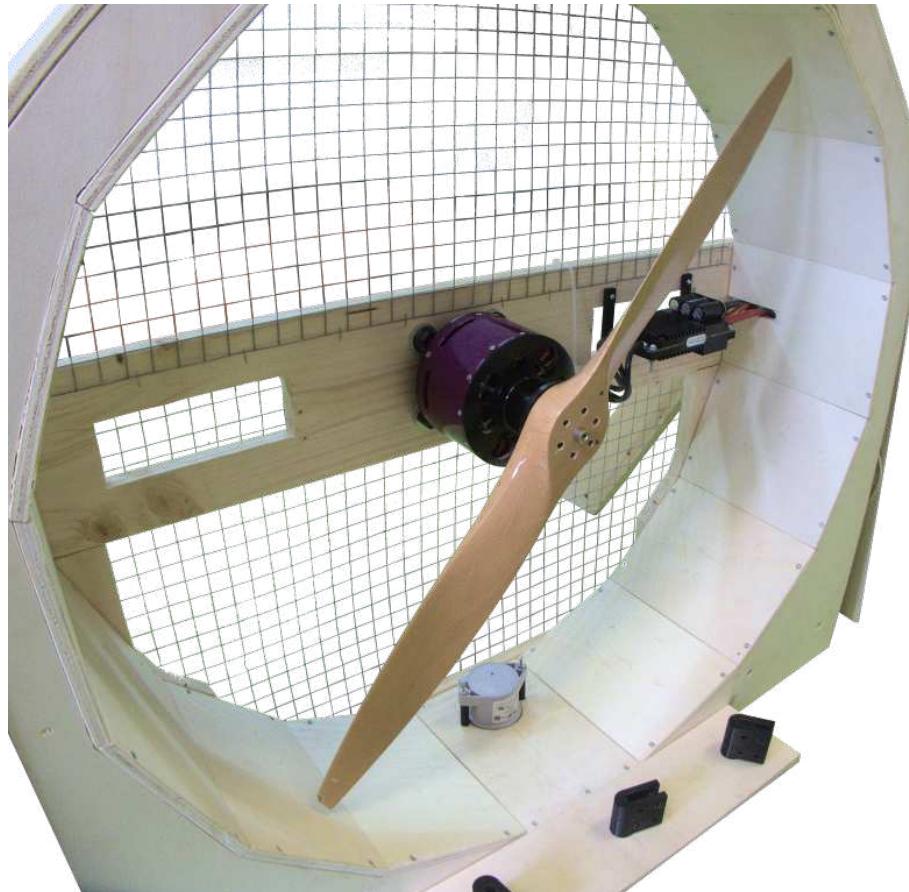


Abbildung 5.2: Hinterer Propeller

Zur Erzeugung des Auftriebes

Um das Luftkissen, auf dem das Boot zu schweben beginnt, erzeugen zu können, wurden zwei, drei blättrige, XOAR Elektro Flugzeug-Propeller 28 x 10 Zoll verwendet.



Abbildung 5.3: Unterer Propeller

Diese sind 60° zueinander versetzt, wodurch die effektive Blattzahl auf sechs erhöht wird, um das Entweichen der Luft zurück nach oben hin zu verhindern.

5.1.3 Motorregler

Die Ansteuerung der Motoren übernimmt der dafür empfohlene **MasterSpin 220 Pro OPTO**. Hierbei handelt es sich um einen Brushless-DC Regler, welcher einen maximalen Dauerstrom von 220 A aufweist.



Abbildung 5.4: MasterSPIN 220 Pro OPTO

Parametrierung durch JetiBox

Um den Regler auf unsere Bedürfnisse parametrieren zu können, wurde eine sogenannte "JetiBox" verwendet.



Abbildung 5.5: Jetibox

Um in das Konfigurationsmenü des Reglers zu gelangen, muss die JetiBox am kürzeren der beiden 3-poligen Kabeln des MasterSpin 220 PRO OPTO (jenes mit dem roten Stecker) angeschlossen werden, während das PWM-Kabel (länger & schwarzer Stecker) nicht verbunden ist.

Konkret wurden folgende Einstellungen getroffen:

Menüpunkt	getroffene Einstellung
Temp. Protection	90 °C
Brake	Hard 70/100/0,5s
Operation Mode	Fast Response
Autorotation	OFF
Timing	22°
Frequency	8 kHz
Acceleration	0-100% 5,0 s
Accumulator Type	Li-Ion/Pol/Fe
Number of Cells	Li-XX 14
Li-XX Cut OFF	V Per Cell 3.2
Off Voltage Set	44,75 V
Cut OFF	Slow Down
Initial Point	AUTO
END Point	2.05 ms / 2.15 ms ¹
AutoInc.END Point	OFF(FIX)
Throttle Curve	Linear
Rotation	Left ²
Start-up Power	Auto
Setting th. R/C	OFF

¹ 2.15 ms zur Begrenzung der Motorleistung auf
≈ 90%

² Abhängig von der Verkabelung des Motors

Tabelle 5.2: Durch JetiBox am Regler konfigurierte Parameter

Eine genauere Beschreibung der einzelnen Parameter findet sich in der Anleitung des MasterSpin 220 Pro OPTO's im Unterabschnitt 12.2.2.

5.2 Servos

Die Steuerung der, für die Umleitung des zur Fortbewegung erzeugten Luftstromes, verwendeten Fahnen, übernimmt je ein **DS5160 Digital Servo**.



Abbildung 5.6: Servos zur Fahnensteuerung

5.3 Schaltplan

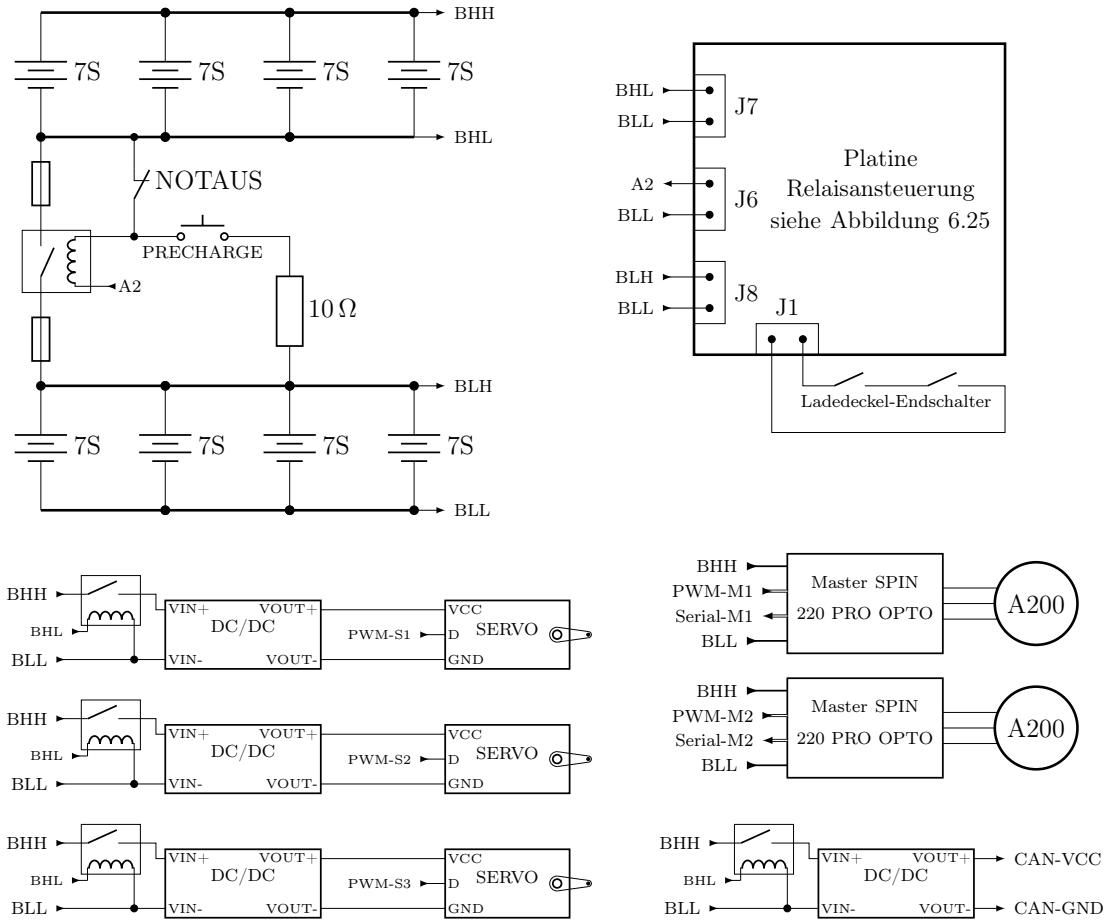


Abbildung 5.7: Schaltplan Leistungselektronik

Für die Verbindung zwischen Board- und Leistungselektronik siehe Abbildung 6.5. Um die Lesbarkeit des Planes möglichst gut zu erhalten, wurde sowohl bei den PWM- als auch bei den Serial-Leitungen der beiden Motorregler auf das Einzeichnen separater +5V sowie GND Leitungen verzichtet und stattdessen mithilfe von doppelten Linien darauf aufmerksam gemacht.

5.4 Verkabelung

Da wir mit beiden Motoren einen Strom von 400A haben, verwenden wir Kupferschienen zur Verteilung dieses Stromes.

Die Akkus gehen jeweils mit beiden Anschlüssen über einen 10mm²-Draht auf eine Kupferschiene. Die Motorregler sind ebenfalls mit jeweils zwei 10mm²-Drähten angelassen, um bei diesen hohen Strömen möglichst wenig Spannungsabfall zu haben. Das Hauptrelais ist über die Schmelzsicherungen direkt mit den Kupferschienen verschraubt.

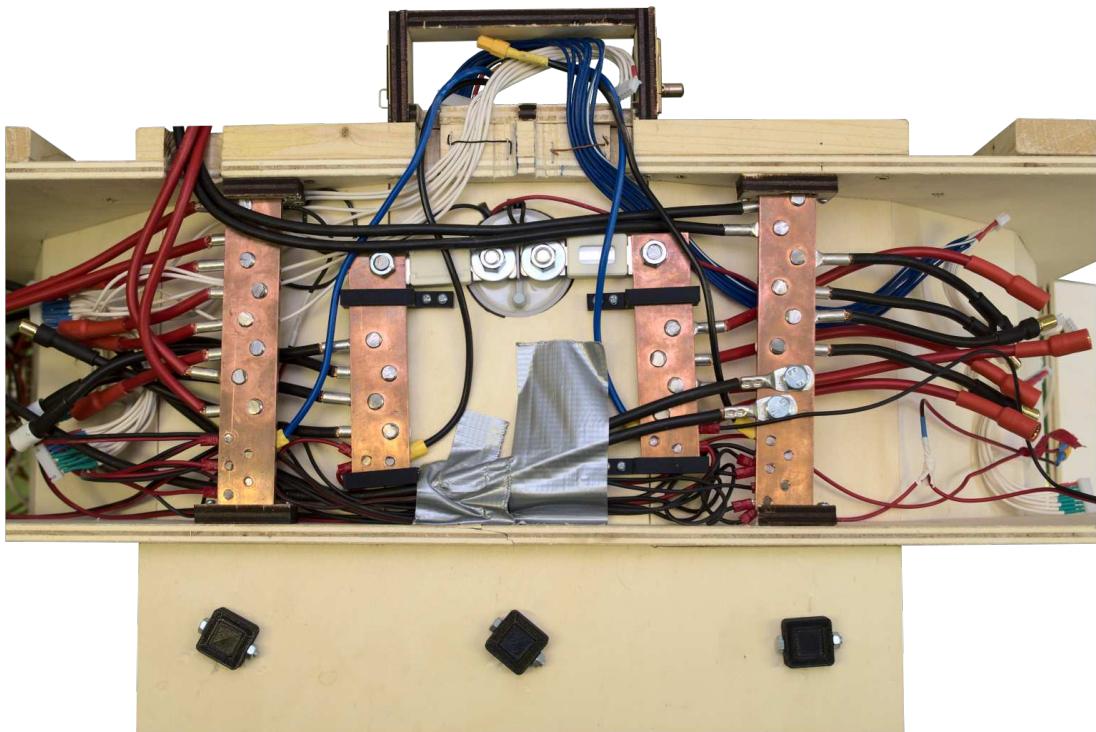


Abbildung 5.8: Verkabelung Kupferschienen

5.5 Hauptrelais

Mit diesem Relais wird die Spannungsversorgung des gesamten Luftkissenbootes kontrolliert. Bedingt durch die Tatsache, dass das gewählte Ladegerät nur maximal 8S-Lipo Akkus laden kann und wir durch die Serienschaltung 2er 7S Akkus auf insgesamt 14S sind, wurde es zwischen den beiden Parallelschaltungen von jeweils 4 Akkus platziert.



Abbildung 5.9: TE Connectivity EV200 Kfz-Relais

5.6 Schmelzsicherungen

Um im Falle eines Fehlers einen dauerhaften Kurzschluss aller Akkus zu verhindern, wurde sowohl vor als auch nach dem Hauptrelais eine 500 A Kfz-DC Sicherung verbaut.

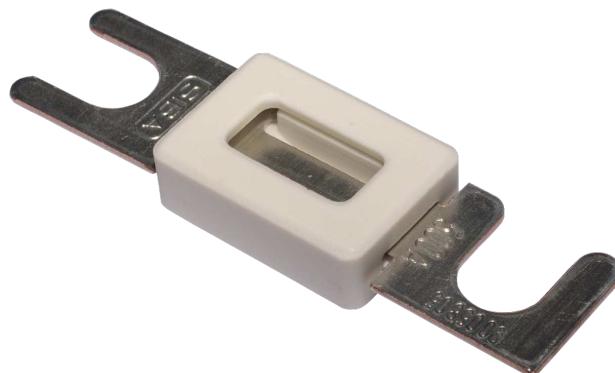


Abbildung 5.10: Automotive Kfz Sicherung

5.7 Buck-Converter

Die Versorgung der Bootselektronik und der Servos übernehmen Buck-Converter, von welchen jeweils einer für ein Servo und einer für die gesamte restliche Elektronik verbaut sind.



Abbildung 5.11: Buck Converter

Wichtig war bei der Auswahl dieser, auf die maximale Eingangsspannung von mindestens 60 V zu achten. Sehr viele der angebotenen Buck-Converter sind auf 40 V begrenzt und können daher nicht verwendet werden. Auch ein Mindeststrom von 6 A sekundärseitig ist aufgrund des hohen Leistungsbedarfes der Servos vonnöten.

5.8 Schaltrelais für Buck-Converter

Da die negativen Pole beider Akkustränge während des Ladevorganges verbunden werden, müssen die Buck-Converter über eigene Relais abgetrennt werden. Um die Schaltströme hierfür möglichst klein zu halten, wird jeder der Converter über ein eigenes Relais geschalten.



Abbildung 5.12: Relais

Hierbei handelt es sich um spezielle DC-Relais, welche für einen Nennstrom von 10 A ausgelegt sind. Diese stellen sicher, dass die Buck-Converter während des Ladevorganges von den Akkus getrennt sind.

5.9 Akkus

Nach gründlicher Überlegung, welche Form von Energiespeicher für unser Boot die beste wäre, fiel die Wahl auf die TopFuel LiPo 35C Power-X 7500mAh 7S. Ursprüngliche Pläne, den Akku selbst aus 18650-Zellen zusammenzubauen, wurden verworfen, da er dadurch bei gleicher speicherbarer Energiemenge deutlich schwerer und auch größer ausgefallen wäre.



Abbildung 5.13: TopFuel Power-X 7500mAh 7S

Auf jeder Seite des Bootes sind, versteckt im hinteren Aufbau, jeweils vier von diesen Akkus verbaut. Mit einer Kapazität von 7500 mAh und einer Nominalspannung von 25.9 V pro Akku ergibt sich somit eine Gesamtkapazität von zirka 1.55 kWh.

5.10 Ladeelektronik

Nach längerer Überlegung wurde entschieden, kein Batteriemanagement selbst zu entwerfen, sondern ein Ladegerät mit zugehörigem Schaltnetzteil zuzukaufen.

Eine Anleitung für den Ladevorgang findet man im Abschnitt 9.2.

5.10.1 Ladegerät

Verwendet wird das Ladegerät „iSDT SMART Duo Ladegerät P30 - 1500W, 30A“. Das Ladegerät ist ein 2-Kanalladegerät, das heißtt, dass beide Akku-Stränge gleichzeitig geladen werden können. Das P30 kann mit dem Programm „Dual Task“ beide Kanäle mit demselben Programm laufen lassen. Somit gibt man die Ladeparameter nur einmal ein.

Der maximale Ladestrom beträgt 30A pro Strang.

Weitere Infos sind unter [Link zur Website](#) zu finden.



Abbildung 5.14: Ladegerät

5.10.2 Schaltnetzteil

Die Wahl fiel auf das zum Ladegerät empfohlene „iSDT SP3060 SMART POWER Schaltnetzteil“. Das Schaltnetzteil ist gegen Überlast, Überspannung, Überhitzung und Kurzschluss geschützt.



Abbildung 5.15: Schaltnetzteil

5.11 Stückliste

Name	Stück	Händler	Bestell Nr.
Hacker A200-6	2	Conrad	255083 - AW
Jeti MasterSPIN 220 Pro OPTO	2	Conrad	275457 - AW
Jeti Programmierbox	1	Conrad	208809 - AW
XOAR Elektro Flugzeug-Propeller	1	Conrad	321841 - AW
XOAR Elektro Flugzeug-Propeller ¹	2	Conrad	322012 - AW
TopFuel LiPo 35C Power-X 7500mAh 7S	8	Hacker	97500761
ISDT P30 Smart Charger	1	modell- hubschrauber.at	34549
ISDT SP3060 Smart Power Supply	1		32324
Omron G2R Monostabiles Relais	4	RS-Components	794-8139
Sockel Omron P2R-057P	4	RS-Components	260-4446
TE Connectivity Kfz-Relais	1	RS-Components	690-0814
Automotive Kfz Sicherung	2	RS-Components	161-1829
Kupfer-Sammelschiene	1	Schrack	IS505069
Presskabelschuh	40	Schrack	XCZ102R8
Servo	4	Amazon	Link
Buck Converter ¹	4	Amazon	Link

¹ zum Zeitpunkt des Schreibens der Dokumentation nicht mehr verfügbar

Tabelle 5.3: Stückliste der Leistungselektronik

6 Boardelektronik

Zusammen mit der für den jeweiligen Einsatzzweck benötigten Peripherie bilden sowohl ein Arduino Nano als auch ein CAN-Bus Modul den Grundstein für die zur Steuerung des Bootes benötigte Elektronik.

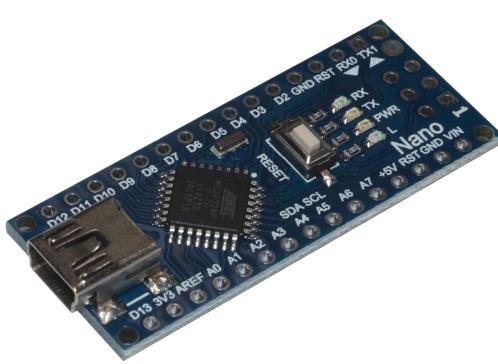


Abbildung 6.1: Arduino Nano

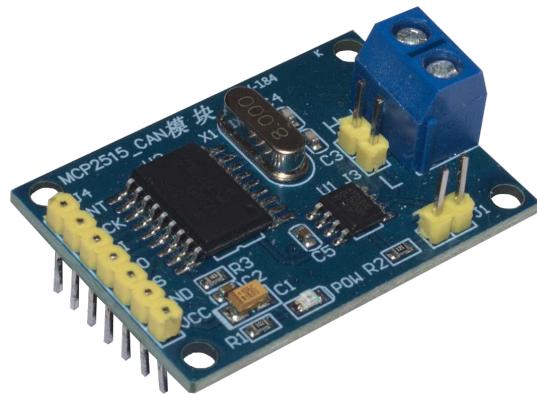


Abbildung 6.2: CAN-Bus Modul

6.1 Übertragungsmedium CAN-Bus

6.1.1 Gründe für die Verwendung

Um eine zuverlässige Übertragung der Regelparameter für unsere Motoren sowie Servos garantieren zu können, wurde für deren Übertragung ein aus der Automobilbranche bekanntes Übertragungsmedium verwendet - der CAN-Bus. Neben seiner Robustheit trotz relativ hoher Übertragungsgeschwindigkeit war es vor allem sein dezentraler Aufbau, der ihn für unseren Anwendungszweck optimal gemacht hat.

6.1.2 Verwendete Adressen

Folgende Tabelle 6.1 zeigt die zur Kommunikation verwendeten Adressen (CAN-IDs), sowie eine kurze Information über deren Funktion und Inhalt.

Name	CAN-ID	Info	Länge ¹
CONTROL_MOTORS_SERVOS	0xC0	Gas-& Lenkerstellung	3
INFOS_LOWER_CONTROLLER	0xC1	Telemetriedaten Motor unten	6
INFOS_BACK_CONTROLLER	0xC3	Telemetriedaten Motor hinten	6
BATTERY_TEMPS	0xE0	Akku-Temperaturen	8

¹ Länge der Nutzdaten (nicht der gesamten Nachricht) in Byte

Tabelle 6.1: Verwendete CAN-Bus Adressen

6.2 Zur Programmierung verwendete Software

Die Programmierung der Mikrocontroller wurde in *PlatformIO*^[3] durchgeführt. Hierbei handelt es sich um eine Programmierumgebung, in welcher sich, ähnlich wie in der *Arduino IDE*, eine Vielzahl von unterschiedlichen Prozessoren programmieren lassen.

Sie bietet nicht nur den Vorteil, über eine gut funktionierende Autovervollständigung zu verfügen, sondern bietet auch (anders als die Verwendung eines reinen WINAVR Compilers, wie beispielsweise in der Eclipse) die Möglichkeit, *Arduino.h* zu verwenden, um den Programmcode nicht unnötig komplex ausführen zu müssen.

6.2.1 Visual Studio Code

Wenngleich *PlatformIO* in einer Reihe von Texteditoren zur Verfügung steht, wurde *Visual Studio Code* von *Microsoft* aufgrund seiner umfassenden Erweiterungsmöglichkeiten verwendet.

Visual Studio Code Webseite

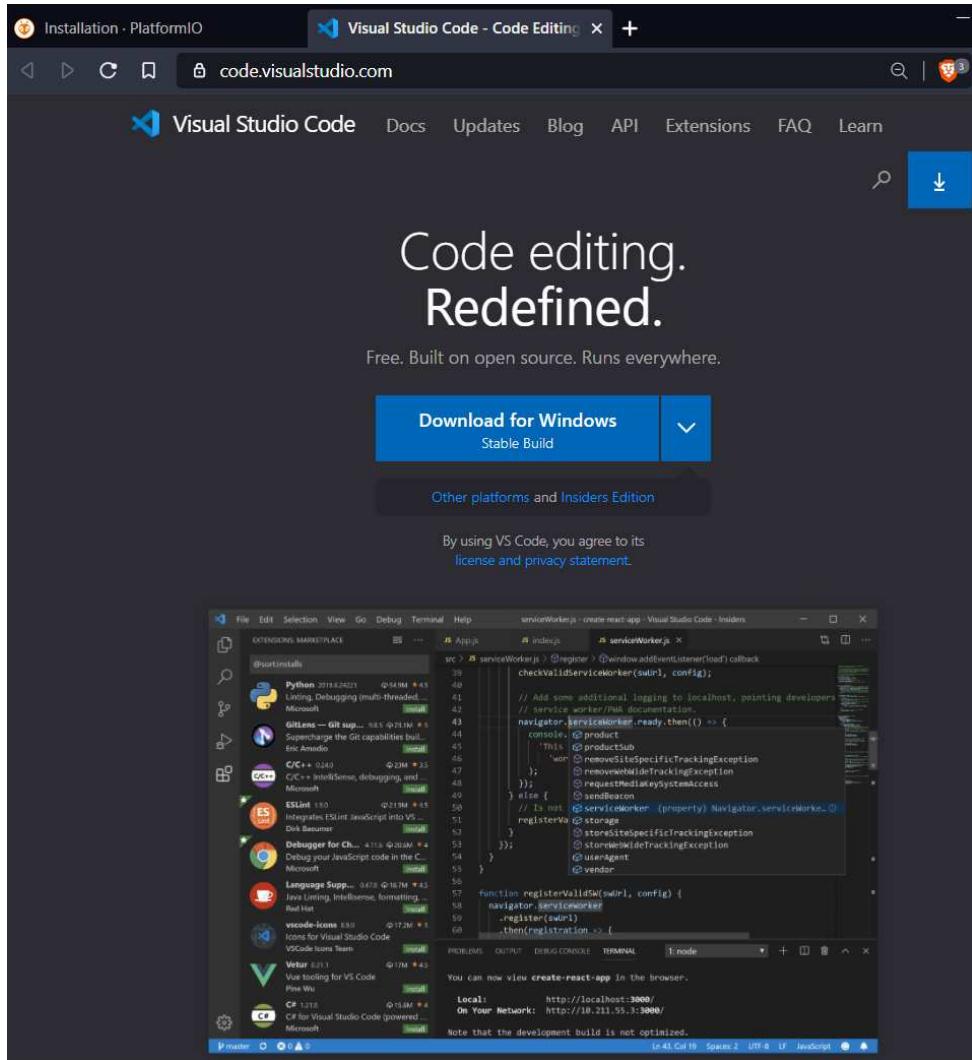


Abbildung 6.3: Visual Studio Code Webseite

Visual Studio Code wird anschließend mit seiner Standard-Auswahl bei allen Abfragen installiert, weswegen hier auf eine genauere Dokumentation verzichtet wurde.

6.2.2 PlatformIO IDE

Die Installation der Programmierumgebung *PlatformIO IDE* erfolgt direkt über den in *Visual Studio Code* integrierten Erweiterungs-Manager.

[PlatformIO Download Webseite](#)

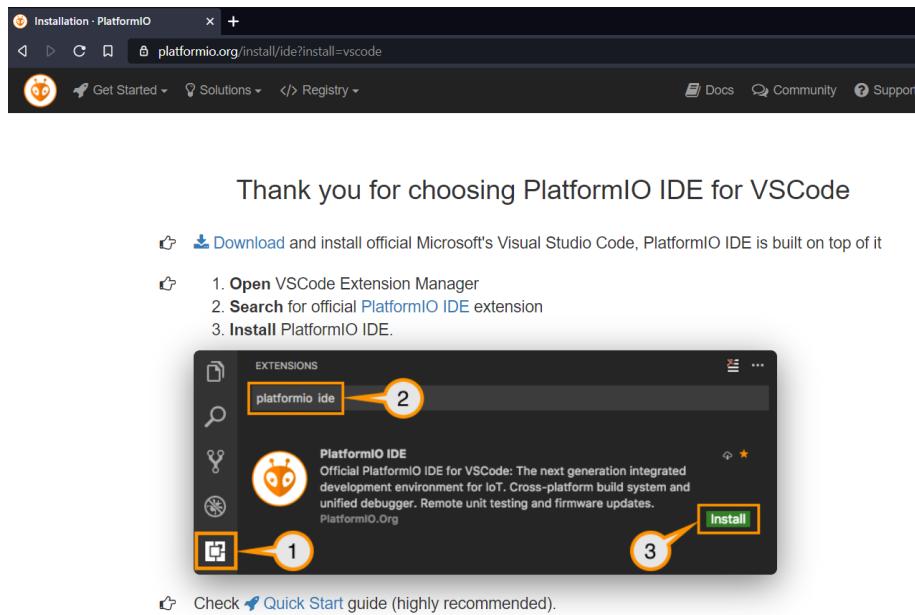


Abbildung 6.4: PlatformIO Webseite

Anmerkung:

In neueren Versionen von Visual Studio Code ist das Icon des *Extension Managers* nicht mehr das in obiger Abbildung gezeigte, sondern folgendes:



Dieser Vorgang kann einige Minuten in Anspruch nehmen und sollte am Ende zu einem *Reload Window* auffordern. Danach ist die für die Diplomarbeit benötigte Software vollständig installiert und einsatzbereit.

Video um den Einstieg in PlatformIO zu erleichtern: [Youtube-Tutorial zu PlatformIO](#)

6.3 Steuerungsplatinen

6.3.1 Schaltplan – Übersicht

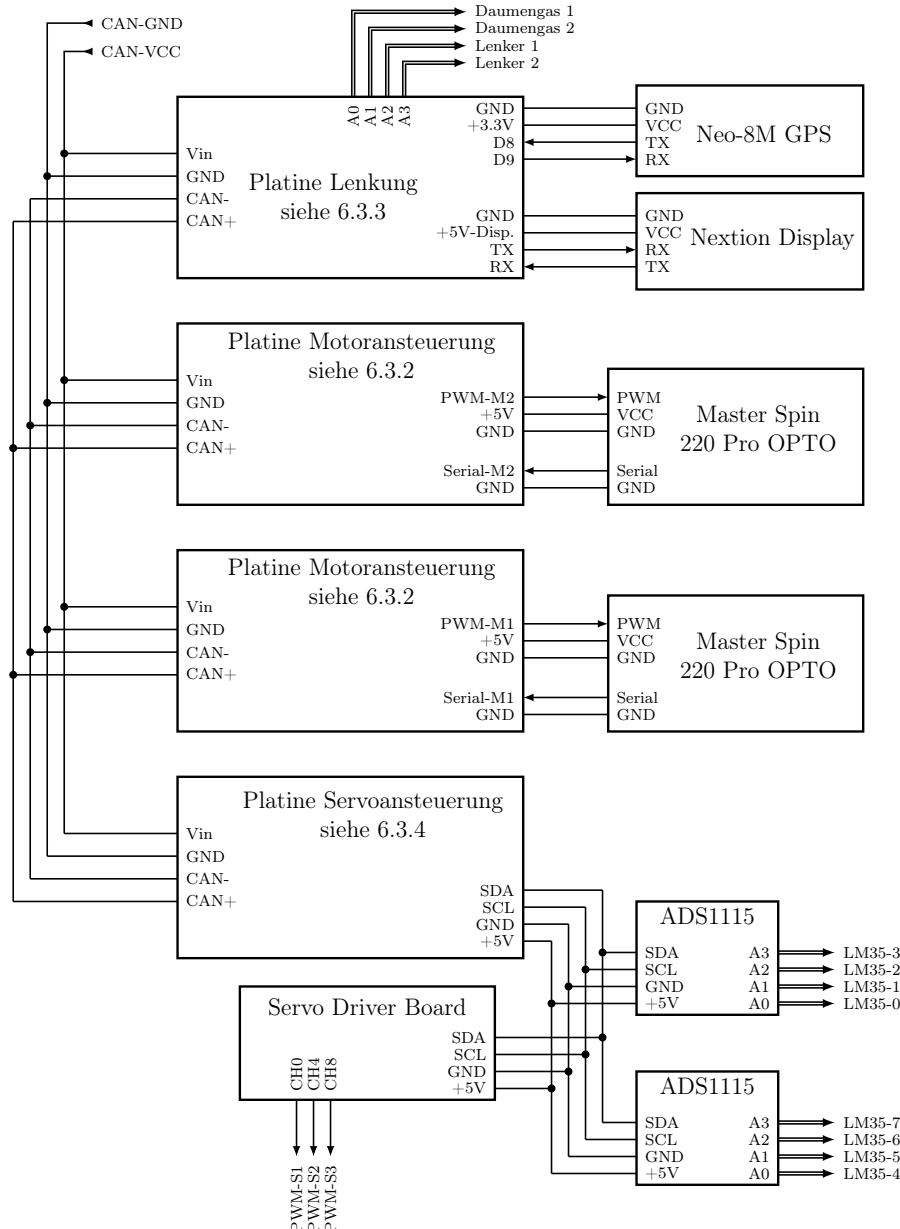


Abbildung 6.5: Schaltplan Boardelektronik

Auf die Versorgung der LM35-Temperatursensoren, sowie der beiden Daumengase und dem Lenkpotentiometer mit +5V und GND wurde aus Platzgründen im obigen Plan verzichtet und stattdessen mit doppelten Linien aufmerksam gemacht. Siehe zudem Abbildung 5.7 für die Verbindung des Planes zur Leistungselektronik.

6.3.2 Motorregleransteuerung

Aufgaben & verwendete Bauteile

Diese Platinen wandeln die entsprechenden vom CAN-Bus übertragenen Daten mit Hilfe ihres 16-bit Timers (Timer1 des ATMEGA328p) in ein für unseren Motorregler verständliches PWM-Muster um. Gleichzeitig übermitteln sie die von den Motorreglern übertragenen Telemetriedaten an den Controller des Lenkers.

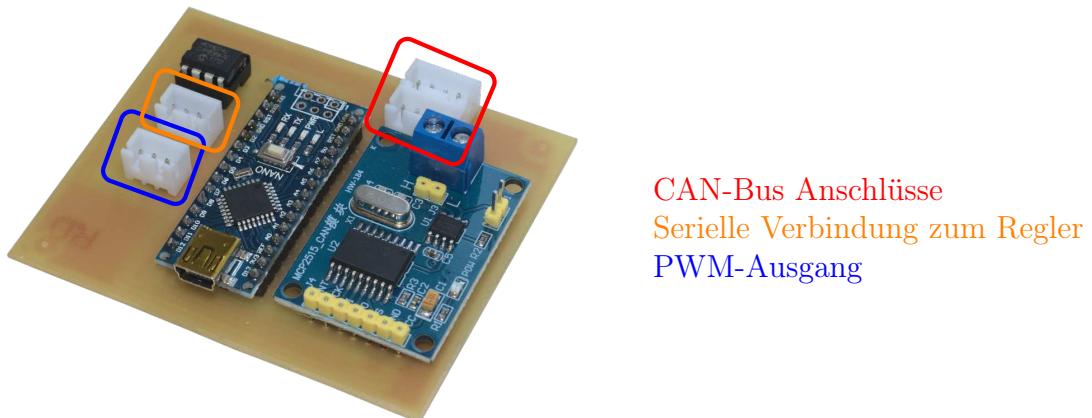


Abbildung 6.6: Platine zur Regleransteuerung

Anmerkung:

Durch die Verwendung der seriellen Schnittstelle des Arduinos muss zur Programmierung der Operationsverstärker entfernt werden.

Schaltplan

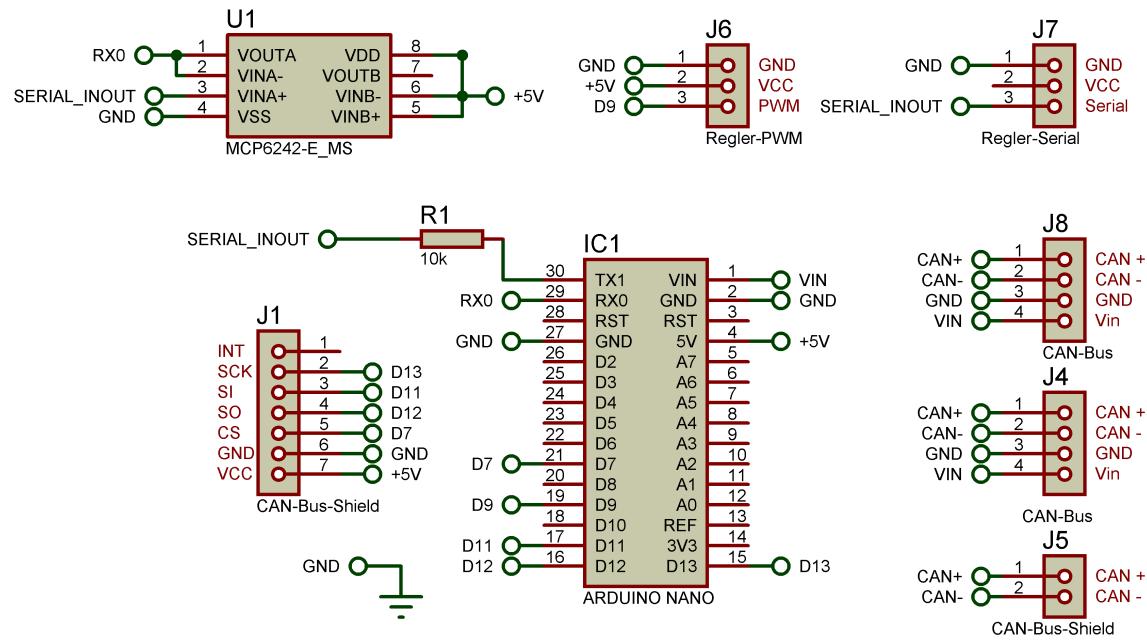


Abbildung 6.7: Schaltplan der Regler-Platine

PCB-Layout

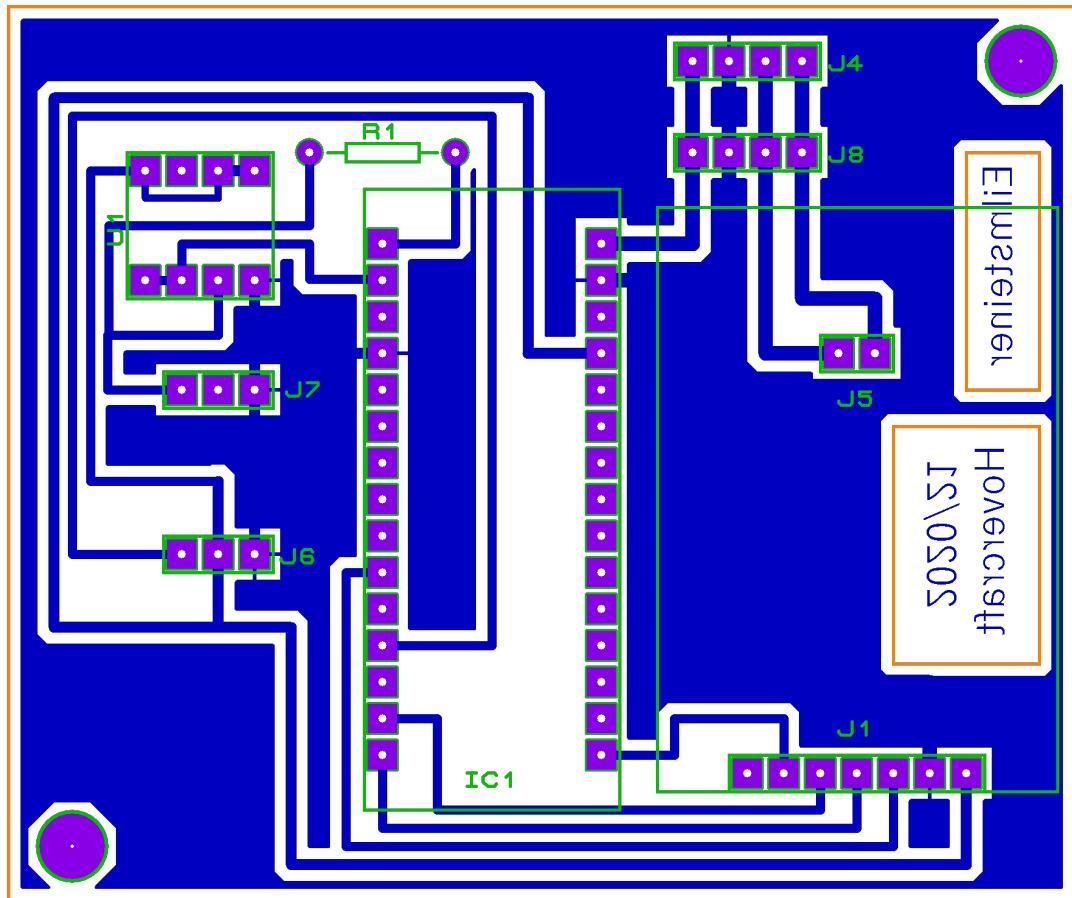


Abbildung 6.8: PCB-Layout der Regler-Platine

Programmcode

```
1 #include <Arduino.h>
2 #include <SPI.h>
3 #include <mcp2515.h>
4
5 //----- Do not change these values! -----
6 #define CAN_ID_CONTROL_MOTORS_SERVOS 0xC0
7 #define CAN_ID_INFOS_LOWER_CONTROLLER 0xC1
8 #define CAN_ID_INFOS_BACK_CONTROLLER 0xC3
9 //-----
10 #define MIN_PWM_VALUE 30
11
12 #define CONTROLLER_ID 1 //either 0 for bottom or 1 for back controller
13
14 #define INFO_SEND_INTERVAL 1000000 //in micro-seconds
15 #define MAX_TIME_WITHOUT_UPDATE 4000000 //in micro-seconds
16
17 #if not(CONTROLLER_ID == 0 or CONTROLLER_ID == 1)
18 #error wrong controller ID defined!
19 #endif
20
21 #include <JetiModes.hpp>
22 #define JETI_MODE ARDUINO_UNO
23 #include <Jeti.hpp>
24
25 JetiBase *jeti = new Jeti_UNO();
26 MCP2515 mcp2515(7);
27
28 //CAN-Bus-Data-Frames
29 struct can_frame canMsg;
30 struct can_frame canMsg_send;
31
32 unsigned long last_data_sent = 0;
33 unsigned long last_data_received = 0;
34
35 void setup()
36 {
37
38   DDRB = DDRB | OB010; //Setting Port D9 as an output
39   PORTB = PORTB | Ob110;
40
41   //initialize Timer1 to a Fast-PWM-Mode
42   TCCR1A = Ob10 << COM1A0 | Ob10 << WGM10; //Clear OC1A/OC1B on Compare Match + Fast PWM (TOP = ICR1)
43   TCCR1B = Ob11 << WGM12 | Ob11 << CS10; //Prescaler 64 + Fast PWM
44   //TIMSK1=Ob110; //Enabeling the overflow interrupts leads to inexplicable problems with the SPI-
45   //→ interface
46
47   //Top-value of Timer1 for a frequency of approximately 60Hz
48   ICR1H = OB10000;
49   ICR1L = OBO;
50
51   sei(); //Enabeling the interrupts
52
53   OCR1AH = Ob01;
54   OCR1AL = MIN_PWM_VALUE;
55
56   //initialize canbus-shield
57   mcp2515.reset();
58   mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
59   mcp2515.setNormalMode();
60   #if CONTROLLER_ID == 0
61     canMsg_send.can_id = CAN_ID_INFOS_LOWER_CONTROLLER;
```

```

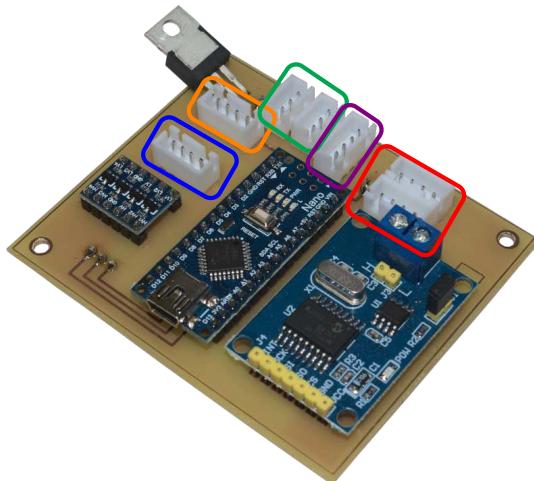
61 #endif
62 #if CONTROLLER_ID == 1
63   canMsg_send.can_id = CAN_ID_INFOS_BACK_CONTROLLER;
64 #endif
65   canMsg_send.can_dlc = 6; //set canbus-package-length
66
67   jeti->init(); //initialize controller-decoder
68 }
69
70 void loop()
71 {
72   if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)
73   { //check for new control data
74     if ((canMsg.can_id == CAN_ID_CONTROL_MOTORS_SERVOS) && (canMsg.can_dlc == 3))
75     {
76       //check if correct message was received and if its length is
77       //correct
78       last_data_received = micros(); //store timestamp
79 #if CONTROLLER_ID == 0
80       int value = canMsg.data[0];
81 #endif
82 #if CONTROLLER_ID == 1
83       int value = canMsg.data[1];
84 #endif
85       OCR1AH = 0b01;
86       OCR1AL = max(value, MIN_PWM_VALUE); //set PWM duty cycle according to received value
87     }
88
89   if (micros() - last_data_received >= MAX_TIME_WITHOUT_UPDATE)
90   { //stop the motor if no data from CAN-Bus was received for 400 milliseconds
91     OCR1AH = 0b01;
92     OCR1AL = MIN_PWM_VALUE;
93   }
94
95   jeti->loop();
96   if ((micros() - last_data_sent) >= INFO_SEND_INTERVAL && jeti->isNewMsg())
97   {
98     jetiTelemetry_t tel = JetiBase::getTelemetry(jeti->getMsg()); //decode latest message from
99     //controller
100    double voltage_100 = tel.voltage * 100.0; //multiply voltage by 100 to avoid floating point
101    //numbers
102    int voltage = (int)voltage_100; //typecasting voltage*100.0 to integer
103    canMsg_send.data[0] = voltage >> 8; //Voltage High-Byte
104    canMsg_send.data[1] = voltage & 0xFF; //Voltage Low-Byte
105    canMsg_send.data[2] = tel.temperature & 0xFF; //Temperature
106    canMsg_send.data[3] = tel.percent & 0xFF; //Power [%]
107    canMsg_send.data[4] = tel.rpm >> 8; // Rotations High-Byte
108    canMsg_send.data[5] = tel.rpm & 0xFF; //Rotations Low-Byte
109    mcp2515.sendMessage(&canMsg_send); //send new controller info to the canbus
110    last_data_sent = micros(); //store timestamp of sending controller telemetry
111 }

```

6.3.3 Lenker

Aufgaben & verwendete Bauteile

Neben seiner Hauptaufgabe, nämlich dem Einlesen und der Übertragung der beiden Daumengas- sowie Lenkerstellungen, ist dieser Mikrocontroller weiters für die Darstellung der von den verschiedenen Komponenten erhaltenen Telemetriedaten auf dem Bildschirm verantwortlich. Dazu zählen neben den Informationen über die beiden Motoren vor allem auch die Temperaturen aller verbauten Akkus sowie die per GPS-Modul ermittelte derzeitige Geschwindigkeit.



CAN-Bus Anschlüsse

Serielle Verbindung zum GPS-Modul

Serielle Verbindung zum Bildschirm

Daumengas Anschlüsse

Anschluss Lenkerpotenziometer

Abbildung 6.9: Platine des Lenkers



Abbildung 6.10: NEO-8M GPS-Modul



Abbildung 6.11: Daumengas

Schaltplan

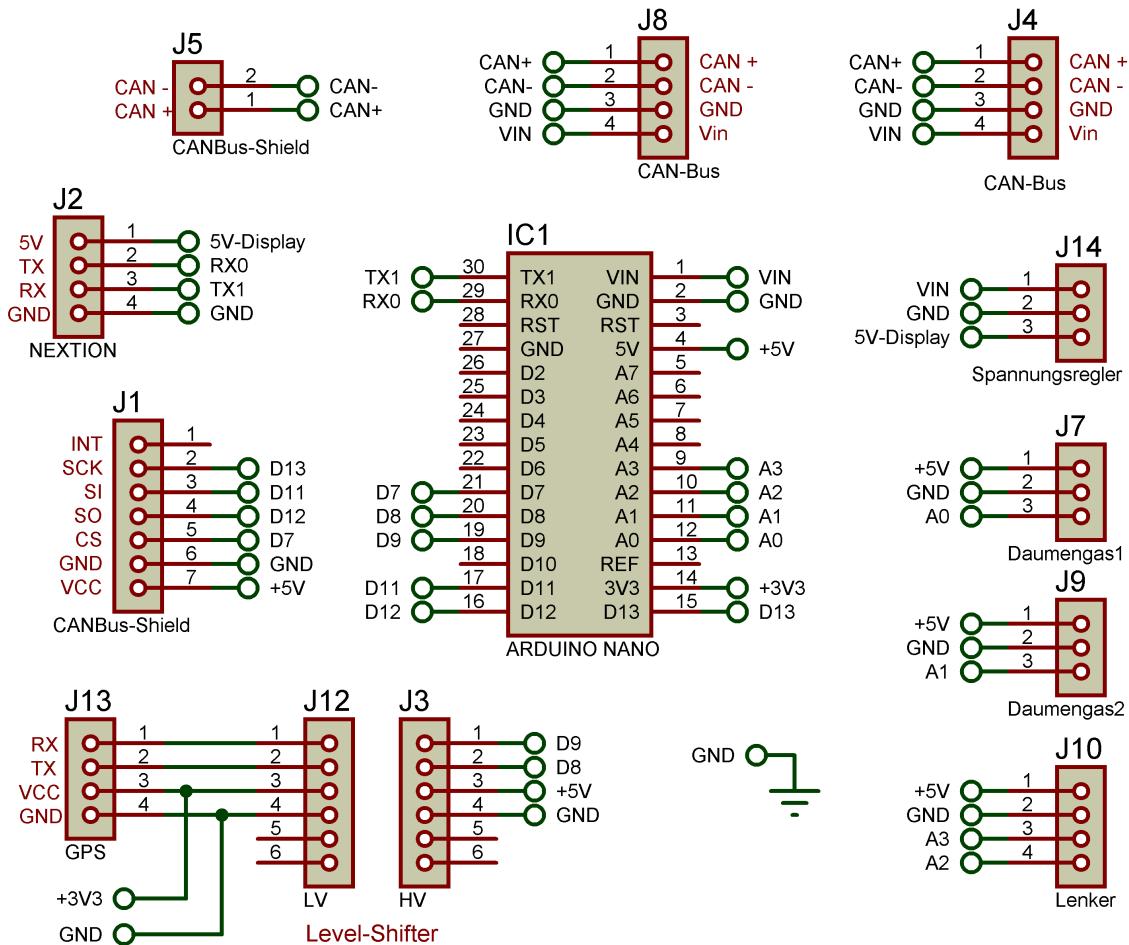


Abbildung 6.12: Schaltplan der Lenker-Platine

PCB-Layout

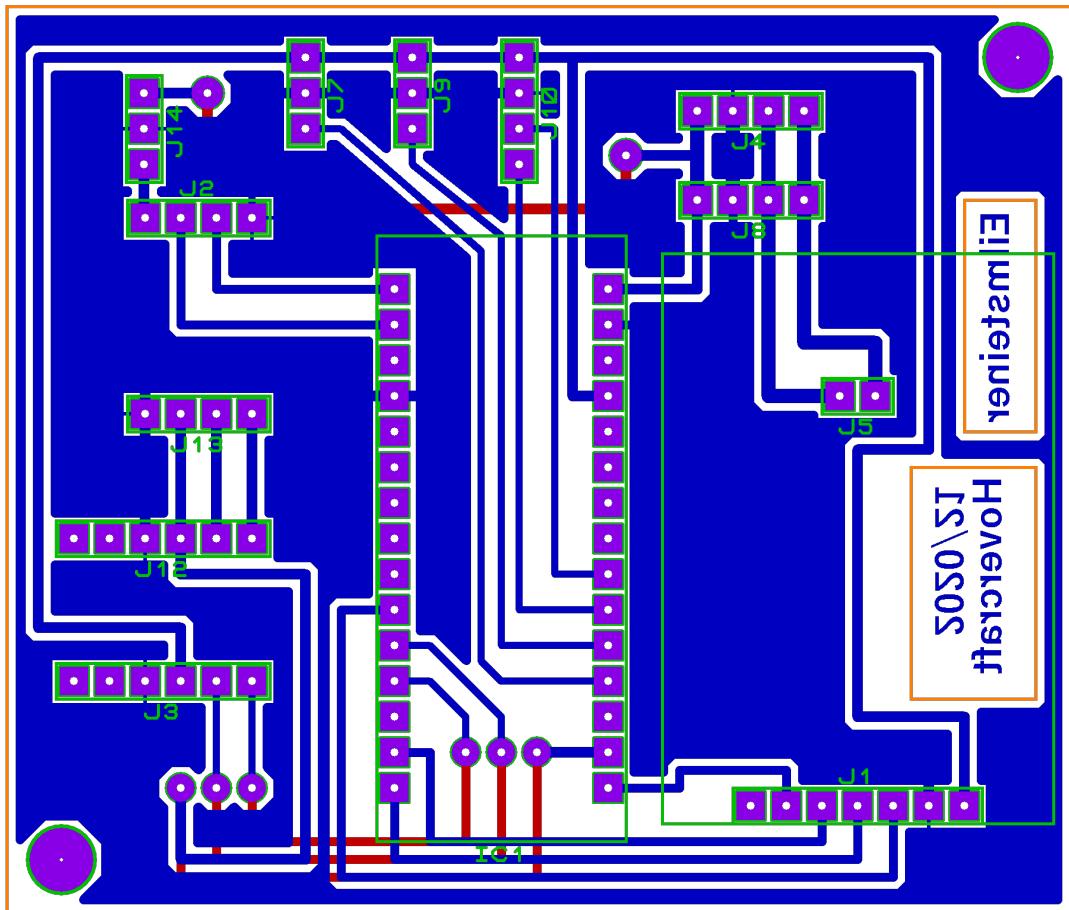


Abbildung 6.13: PCB-Layout der Lenker-Platine

Programmcode

```
1 #include <SPI.h>
2 #include <mcp2515.h>
3 #include "Nexzion.h"
4 #include "my_TinyGPS++.h"
5 #include <SoftwareSerial.h>
6
7 #define CAN_ID_CONTROL_MOTORS_SERVOS 0xC0
8 #define CAN_ID_INFOS_LOWER_CONTROLLER 0xC1
9 #define CAN_ID_INFOS_BACK_CONTROLLER 0xC3
10 #define CAN_ID_BATTERY_TEMPS 0xE0
11
12 static const int RXPin = 8, TXPin = 9;
13 static const uint32_t GPSBaud = 9600;
14
15 uint32_t max_power_lower = 255;
16 uint32_t max_power_back = 255;
17 uint32_t steering_offset = 0;
18
19 #define MIN_VALUE_A0 190 //Analogue value of thumb throttle connected to A0 when not pressed
20 #define MAX_VALUE_A0 862 //Analogue value of thumb throttle connected to A0 when fully pressed
21 #define MIN_VALUE_A1 185 //Analogue value of thumb throttle connected to A1 when not pressed
22 #define MAX_VALUE_A1 865 //Analogue value of thumb throttle connected to A1 when fully pressed
23
24 TinyGPSPlus gps; //TinyGPS++ object
25
26 SoftwareSerial ss(RXPin, TXPin); //serial connection the GPS device
27
28 //This custom version of delay() ensures that the gps object
29 //is being "fed".
30 static void smartDelay(unsigned long ms)
31 {
32     unsigned long start = millis();
33     do
34     {
35         while (ss.available())
36             gps.encode(ss.read());
37     } while (millis() - start < ms);
38 }
39
40 #define NEXTION_ENABLED
41
42 #ifdef NEXTION_ENABLED
43
44 int nextion_page = 0; //screen number currently displayed on nextion
45
46 // -----nextion objects for main page -----
47 NexText disp_gps_speed = NexText(0, 15, "gpsspeed");
48
49 NexText disp_drz_u = NexText(0, 3, "drzunt");
50 NexText disp_temp_u = NexText(0, 2, "tempunt");
51 NexText disp_gas_u = NexText(0, 24, "gasunt");
52 NexText disp_spg_u = NexText(0, 4, "spgunt");
53
54 NexText disp_drz_h = NexText(0, 10, "drzhint");
55 NexText disp_temp_h = NexText(0, 9, "temphint");
56 NexText disp_gas_h = NexText(0, 29, "gashint");
57 NexText disp_spg_h = NexText(0, 11, "spghint");
58
59 NexButton disp_but00_settings = NexButton(0, 22, "b00");
60 NexButton disp_but01_infos = NexButton(0, 23, "b01");
61
```

```
62 // ---- nextion objects for info page -----
63 //battery infos
64 NexText disp_atemp_01 = NexText(2, 14, "temp01");
65 NexText disp_atemp_02 = NexText(2, 15, "temp02");
66 NexText disp_atemp_03 = NexText(2, 16, "temp03");
67 NexText disp_atemp_04 = NexText(2, 17, "temp04");
68 NexText disp_atemp_05 = NexText(2, 18, "temp05");
69 NexText disp_atemp_06 = NexText(2, 19, "temp06");
70 NexText disp_atemp_07 = NexText(2, 20, "temp07");
71 NexText disp_atemp_08 = NexText(2, 21, "temp08");
72 NexButton disp_but20_back = NexButton(2, 1, "b20");
73
74 //gps infos
75 NexText disp_gpsinf_sats = NexText(2, 34, "gpssats");
76 NexText disp_gpsinf_speed = NexText(2, 41, "gpsspeed");
77 NexText disp_gpsinf_course = NexText(2, 40, "gpscourse");
78 NexText disp_gpsinf_lon = NexText(2, 36, "gpslon");
79 NexText disp_gpsinf_lat = NexText(2, 37, "gpslat");
80 NexText disp_gpsinf_height = NexText(2, 35, "gpsalt");
81
82 // ----- nextion objects for settings page -----
83 NexSlider disp_slid_lenk_offset = NexSlider(1, 3, "h1");
84 NexSlider disp_slid_maxbackpower = NexSlider(1, 2, "h0");
85 NexSlider disp_slid_maxdownpower = NexSlider(1, 4, "h2");
86 NexButton disp_but10_back = NexButton(1, 1, "b10");
87
88 NexTouch *nex_listen_list[] = //list of all clickable items on the nextion display
89 {
90     &disp_but00_settings,
91     &disp_but01_infos,
92     &disp_but20_back,
93     &disp_slid_lenk_offset,
94     &disp_slid_maxbackpower,
95     &disp_slid_maxdownpower,
96     &disp_but10_back,
97     NULL};
98
99 void but00_settings_push(void *ptr) //ISR for "Settings" Button on main screen
100 {
101     disp_slid_lenk_offset.setValue(steering_offset);
102     disp_slid_maxbackpower.setValue(max_power_back);
103     disp_slid_maxdownpower.setValue(max_power_lower);
104     nextion_page = 1;
105 }
106 void but01_infos_push(void *ptr) //ISR for "Infos" Button on main screen
107 {
108     nextion_page = 2;
109 }
110 void but20_back(void *ptr) //ISR for "Back" button on info screen
111 {
112     nextion_page = 0;
113 }
114 void but10_back(void *ptr) //ISR for "Back" button on settings screen
115 {
116     nextion_page = 0;
117 }
118 void slid_lenk_offset(void *ptr) //ISR for releasing steering offset slider
119 {
120     disp_slid_lenk_offset.getValue(&steering_offset);
121 }
122 void slid_maxbackpower(void *ptr) //ISR for releasing maximum back power slider
123 {
124     disp_slid_maxbackpower.getValue(&max_power_back);
```

```
125 }
126 void slid_maxdownpower(void *ptr) //ISR for releasing maximum lower power slider
127 {
128     disp_slid_maxdownpower.getValue(&max_power_lower);
129 }
130 #endif
131
132 struct can_frame canMsg; //CAN-Bus object used for writing to CAN-Bus
133 struct can_frame canMsg_r; //CAN-Bus object used for reading from CAN-Bus
134 MCP2515 mcp2515(7);
135
136 void setup()
137 {
138     pinMode(A1, INPUT);
139     pinMode(A2, INPUT);
140     pinMode(A3, INPUT);
141     pinMode(A4, INPUT);
142
143 //initialize NEXTION display
144 #ifdef NEXTION_ENABLED
145     Serial.begin(9600);
146     nexInit();
147     delay(100);
148     //attach interrupts to ISR
149     disp_but01_infos.attachPush(but01_infos_push);
150     disp_but00_settings.attachPush(but00_settings_push);
151     disp_but10_back.attachPush(but10_back);
152     disp_but20_back.attachPush(but20_back);
153     disp_slid_lenk_offset.attachPop(slid_lenk_offset);
154     disp_slid_maxbackpower.attachPop(slid_maxbackpower);
155     disp_slid_maxdownpower.attachPop(slid_maxdownpower);
156 #endif
157     ss.begin(9600); //initialize SoftwareSerial for GPS module
158     //Initialize CAN-Bus shield
159     mcp2515.reset();
160     mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
161     mcp2515.setNormalMode();
162     canMsg.can_id = CAN_ID_CONTROL_MOTORS_SERVOS;
163     canMsg.can_dlc = 3;
164 }
165
166 unsigned long last_update = 0;
167 unsigned long last_gps_update = 0;
168
169 void loop()
170 {
171     if (millis() - last_update > 50) //Send new data to CAN-Bus every 50ms
172     {
173         int int_0 = map(analogRead(A0), MIN_VALUE_A0, MAX_VALUE_A0, 30, max_power_back); //map analog
174         // value of A0 between 0 and max_power_back value
175         int int_1 = map(analogRead(A1), MIN_VALUE_A1, MAX_VALUE_A1, 30, max_power_lower); //map analog
176         // value of A1 between 0 and max_power_lower value
177         canMsg.data[0] = min(int_0, 255); //set power value for lower motor to CAN message
178         canMsg.data[1] = min(int_1, 255); //set power value of back motor to CAN message
179         canMsg.data[2] = (int)(steering_offset) + map(analogRead(A2) >> 2, 23, 88, 255, 0); //calculate
180         // and set steering value for CAN message
181         mcp2515.sendMessage(&canMsg); //send CAN message
182         last_update = millis();
183     }
184 #ifdef NEXTION_ENABLED
185     if (millis() - last_gps_update > 500)
186     { //refresh gps data on display every 500ms
187         if (nextion_page == 0 && gps.speed.isValid()) //when on main page and gps data is valid
```

```

185     {
186         disp_gps_speed.setText(String(gps.speed.kmph() / 1.0).c_str());
187     }
188     else if (nextion_page == 2) //when on Info page
189     {
190         disp_gpsinf_sats.setText(String(gps.satellites.value()).c_str());
191         disp_gpsinf_speed.setText(String(gps.speed.kmph()).c_str());
192         disp_gpsinf_course.setText(String(gps.course.deg()).c_str());
193         disp_gpsinf_lon.setText(String(gps.location.lng()).c_str());
194         disp_gpsinf_lat.setText(String(gps.location.lat()).c_str());
195         disp_gpsinf_height.setText(String(gps.altitude.meters()).c_str());
196     }
197     last_gps_update = millis();
198 }
199 if (mcp2515.readMessage(&canMsg_r) == MCP2515::ERROR_OK)
200 { //check for new CAN-Bus messages
201     if ((canMsg_r.can_id == CAN_ID_INFOS_LOWER_CONTROLLER || (canMsg_r.can_id ==
202         ↪ CAN_ID_INFOS_BACK_CONTROLLER)) && (canMsg_r.can_dlc == 6) && nextion_page == 0)
203     { //check if message is controller
204         ↪ telemetry and valid
205         int spannung = (canMsg_r.data[0] << 8) | canMsg_r.data[1]; //recombine High & Lowbyte of
206         ↪ voltage back to integer
207         int drehzahl = (canMsg_r.data[4] << 8) | canMsg_r.data[5]; //recombine High & Lowbyte of
208         ↪ rotations back to integer
209         int temperatur = canMsg_r.data[2];
210         int percent = canMsg_r.data[3];
211
212         if (canMsg_r.can_id == CAN_ID_INFOS_LOWER_CONTROLLER)
213         { //data from lower controller received
214             disp_drz_u.setText(String(drehzahl).c_str());
215             disp_temp_u.setText(String(temperatur).c_str());
216             disp_gas_u.setText(String(percent).c_str());
217             disp_spg_u.setText(String(spannung / 100.0).c_str());
218         }
219         else
220         { //data from back controller received
221             disp_drz_h.setText(String(drehzahl).c_str());
222             disp_temp_h.setText(String(temperatur).c_str());
223             disp_gas_h.setText(String(percent).c_str());
224             disp_spg_h.setText(String(spannung / 100.0).c_str());
225         }
226     }
227     else if ((canMsg_r.can_id == CAN_ID_BATTERY_TEMPS) && (canMsg_r.can_dlc == 8) && (nextion_page ==
228         ↪ 2))
229     { //battery temperatures received and valid
230         disp_atemp_01.setText(String(canMsg_r.data[0]).c_str());
231         disp_atemp_02.setText(String(canMsg_r.data[1]).c_str());
232         disp_atemp_03.setText(String(canMsg_r.data[2]).c_str());
233         disp_atemp_04.setText(String(canMsg_r.data[3]).c_str());
234         disp_atemp_05.setText(String(canMsg_r.data[4]).c_str());
235         disp_atemp_06.setText(String(canMsg_r.data[5]).c_str());
236         disp_atemp_07.setText(String(canMsg_r.data[6]).c_str());
237         disp_atemp_08.setText(String(canMsg_r.data[7]).c_str());
238     }
239 }
240 nexLoop(nex_listen_list); //check for button presses on nextion screen
241 #endif
242     smartDelay(0); //loop function for gps module
243 }
```

6.3.4 Fahnenansteuerung & Akkutemperaturen

Fahnenansteuerung

Die Ansteuerung der Servos erfolgt ebenfalls mit einem per CAN-Bus angeschlossenen Arduino. Um die für die Steuerung der drei Servos benötigten PWM-Signale, unabhängig von den durch den Arduino bereitgestellten Timer und den damit verbundenen GPIOs, zu erzeugen (Anbindung des CAN-Bus Moduls über SPI benötigt beispielsweise diese Pins und Timer), wurde dafür auf ein externes Board zurückgegriffen.

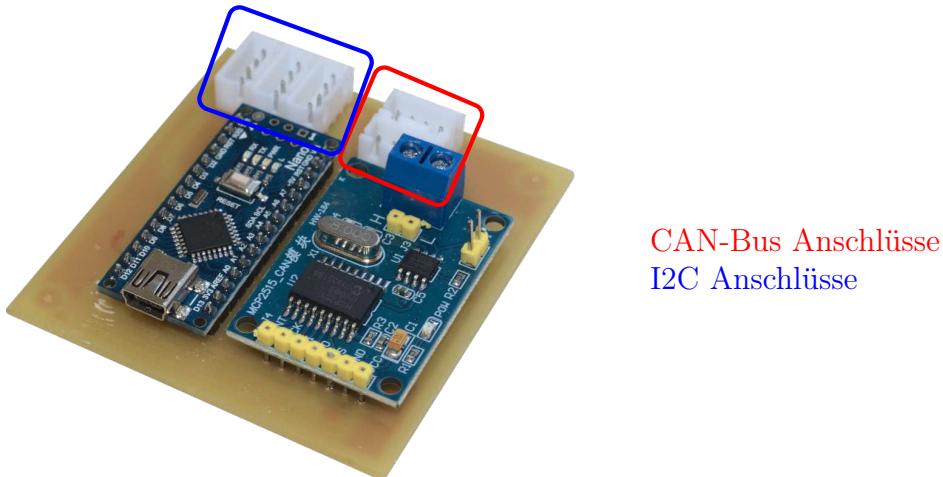


Abbildung 6.14: Platine zur Regleransteuerung

Konkret handelt es sich um das **16-Channel 12-bit PWM/Servo Driver Module** von Adafruit, welcher mittels I2C mit dem Mikrocontroller kommuniziert.

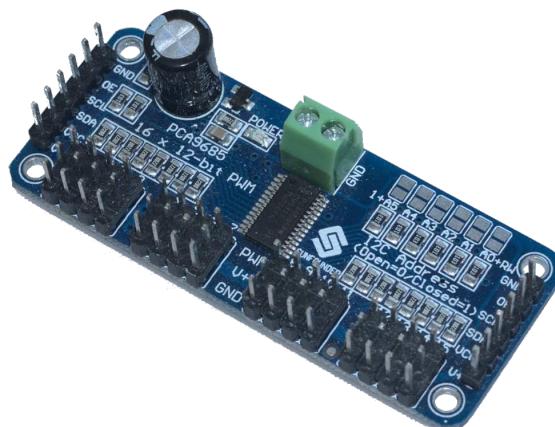


Abbildung 6.15: Adafruit 16-Channel Servo Driver

Schaltplan

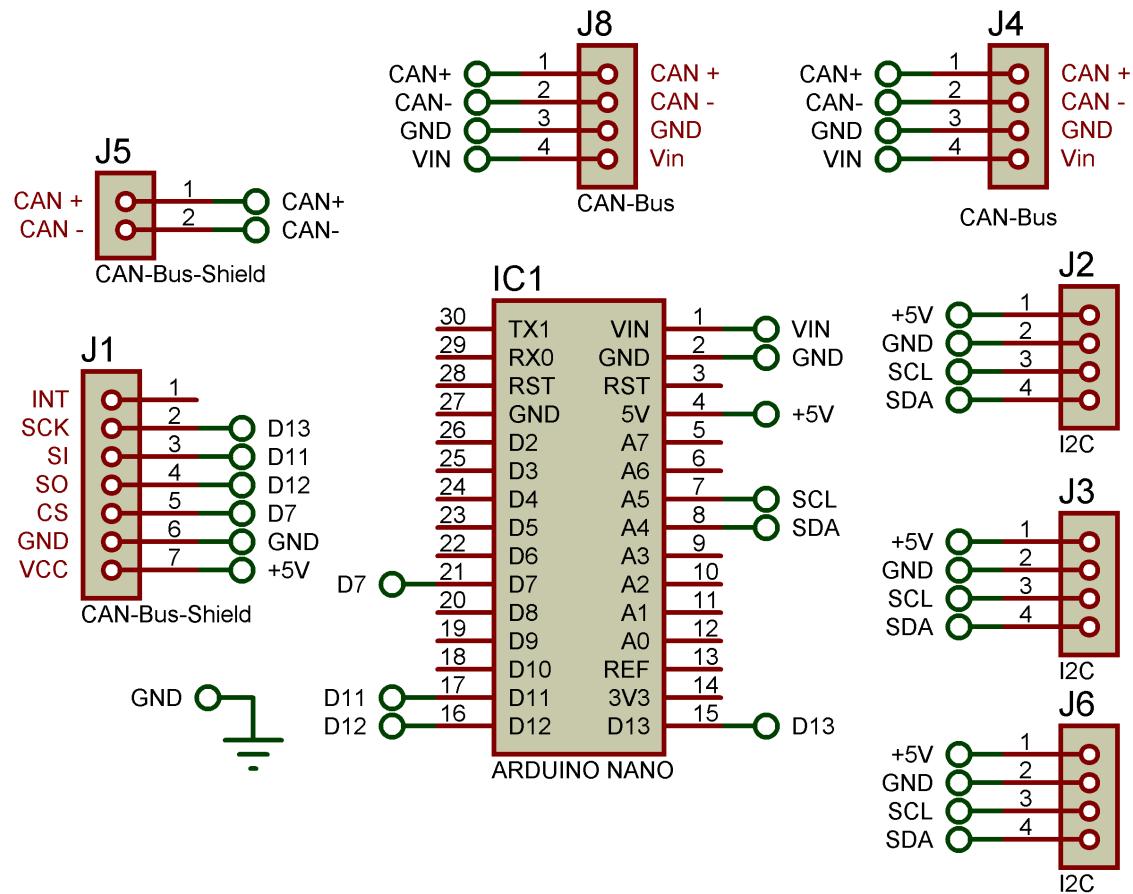


Abbildung 6.16: Schaltplan der Servoansteuerungs-Platine

PCB-Layout

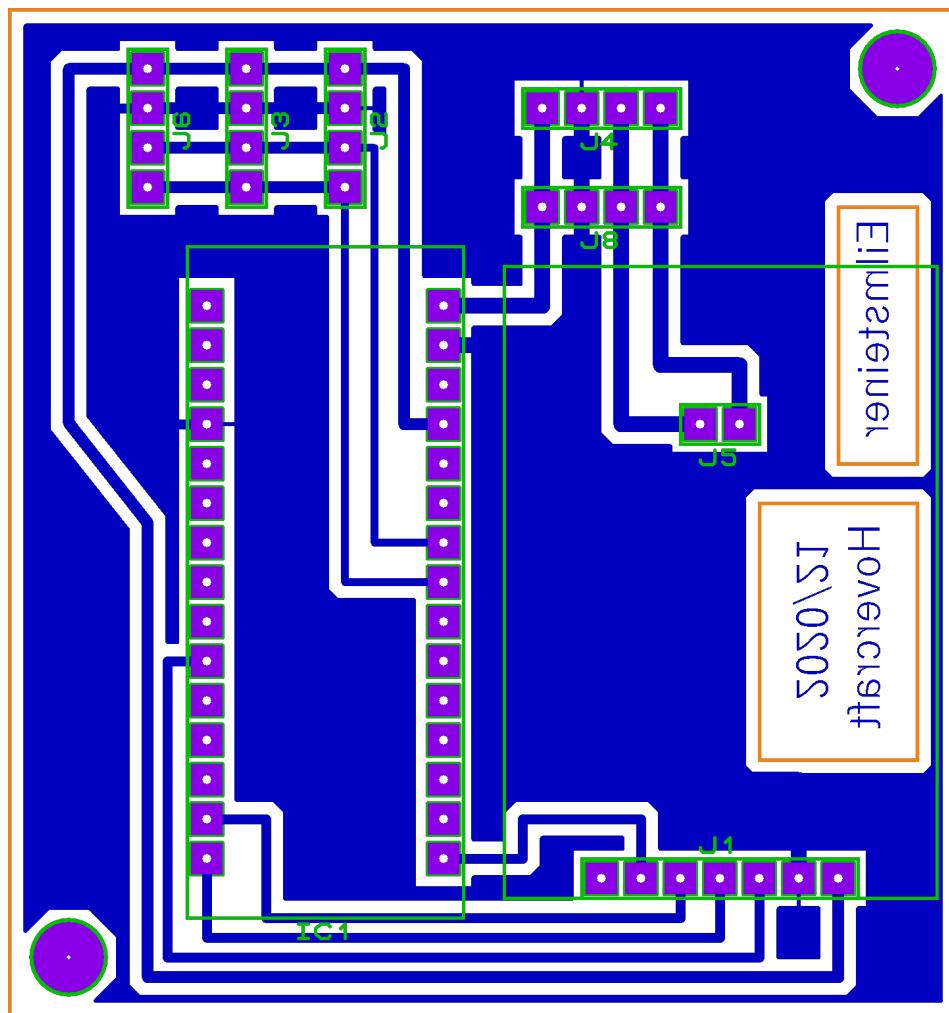


Abbildung 6.17: PCB-Layout der Servoansteuerungs-Platine

Programmcode

```
1 #include <Wire.h>
2 #include <Adafruit_PWMServoDriver.h>
3 #include <Adafruit_ADS1015.h>
4
5 Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
6 Adafruit_ADS1115 adcleft = Adafruit_ADS1115(0x48);
7 Adafruit_ADS1115 adcright = Adafruit_ADS1115(0x49);
8
9 #include <mcp2515.h>
10
11 #define CAN_ID_CONTROL_MOTORS_SERVOS 0xC0
12 #define CAN_ID_BATTERY_TEMPS 0xE0
13 #define SERVO_FREQ 60
14
15 #define ADC_ENABLED
16 //#define SERIAL_DEBUG
17
18 //Servo 1 Grenzwerte: 163 - 322 - 446
19
20 //Servo 2 (Mitte) Grenzwerte: 223 - 355 - 490
21
22 //Servo 3 Grenzwerte: 231 - 355 - 493
23
24 struct can_frame canMsg;
25 struct can_frame cantemps;
26
27 MCP2515 mcp2515(7);
28
29 unsigned long lastsentadc = 0;
30
31 void setup()
32 {
33     Serial.begin(115200);
34
35     pwm.begin();
36
37     pwm.setOscillatorFrequency(27000000);
38     pwm.setPWMFreq(SERVO_FREQ); // Analog servos run at ~50 Hz updates
39
40     delay(10);
41     //initialize all three servo outputs
42     pwm.setPWM(0, 0, 355);
43     pwm.setPWM(4, 0, 355);
44     pwm.setPWM(8, 0, 322);
45
46     mcp2515.reset(); //initialize CAN-Bus
47     mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
48     mcp2515.setNormalMode();
49
50 #ifdef ADC_ENABLED
51     adcleft.begin();
52     adcleft.setGain(GAIN_FOUR);
53     adcright.setGain(GAIN_FOUR);
54     adcright.begin();
55 #endif
56     cantemps.can_id = CAN_ID_BATTERY_TEMPS;
57     cantemps.can_dlc = 8;
58 }
59
60 void loop()
61 {
```

```
62 if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)
63 {
64     if ((canMsg.can_id == CAN_ID_CONTROL_MOTORS_SERVOS) && (canMsg.can_dlc == 3)) //check if new
65         ↪ servo position was received
66     {
67         int anw = map(canMsg.data[2], 0, 255, -127, 127);
68         //center all three servos
69         pwm.setPWM(0, 0, 355 + anw);
70         pwm.setPWM(4, 0, 355 + anw);
71         pwm.setPWM(8, 0, 322 + anw);
72     }
73 }
74 #ifdef ADC_ENABLED
75 if (millis() - lastsentadc > 2000)
76 {
77     int16_t battemp_left[4];
78     int16_t battemp_right[4];
79     for (int i = 0; i <= 3; i++) { //get temperatures from all four ADC channels of both ADS1115
80         ↪ modules
81         int16_t c_adc_left, c_adc_right;
82         int8_t c_temp_left, c_temp_right;
83         c_adc_left = adcleft.readADC_SingleEnded(i);
84         c_adc_right = adcright.readADC_SingleEnded(i);
85         c_temp_left = (int)map(c_adc_left, 0, 52428, 0, 100); //map raw value of left ADS115 to
86         ↪ coresponding temperature
87         c_temp_right = (int)map(c_adc_right, 0, 52428, 0, 100); //map raw value of right ADS115 to
88         ↪ coresponding temperature
89     #ifdef SERIAL_DEBUG
90         Serial.print("CH" + String(i) + ": l:" + c_temp_left + " r:" + c_temp_right + " ");
91     #endif
92     cantemps.data[i] = c_adc_left & 0xFF; //add temperature of left ADC to CAN-Bus message
93     cantemps.data[4 + i] = c_adc_right & 0xFF; //add temperature of right ADC to CAN-Bus message
94 }
95 #ifdef SERIAL_DEBUG
96     Serial.println();
97 #endif
98     mcp2515.sendMessage(&cantemps);
99     lastsentadc = millis(); //store time when data was sent
100 }
```

Akkutemperaturen

Die zweite Aufgabe dieses Controllers ist es, die Temperaturen aller acht im Boot verbauten Akkus zu überwachen, und diese zur Darstellung an den Controller des Lenkers zu übermitteln. Dies geschieht mithilfe zweier (je einer pro Seite), ebenfalls über I2C verbundenen, **ADS1115** Analog-Digital-Wandler, welche wiederum von Adafruit stammen.

Zur Temperaturmessung selbst kommt je ein **LM35** Temperatursensor pro Akku zum Einsatz.

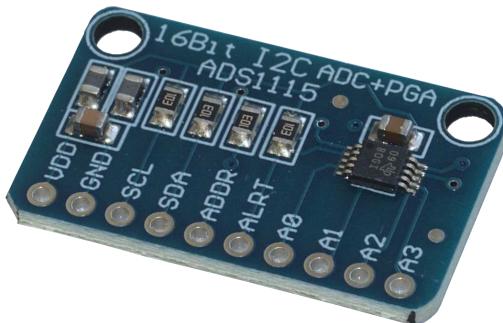


Abbildung 6.18: ADS1115 Modul



Abbildung 6.19: LM35

6.3.5 ADC-Platinen

Um den Anschluss der Temperatursensoren an diesem ADC zu vereinfachen, wurde folgende Platine entworfen:

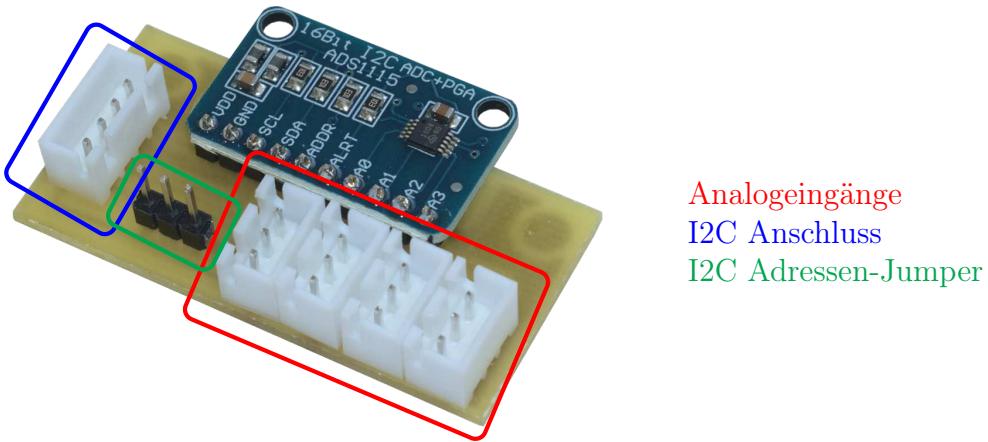


Abbildung 6.20: Platine Temperatursensoren

Schaltplan

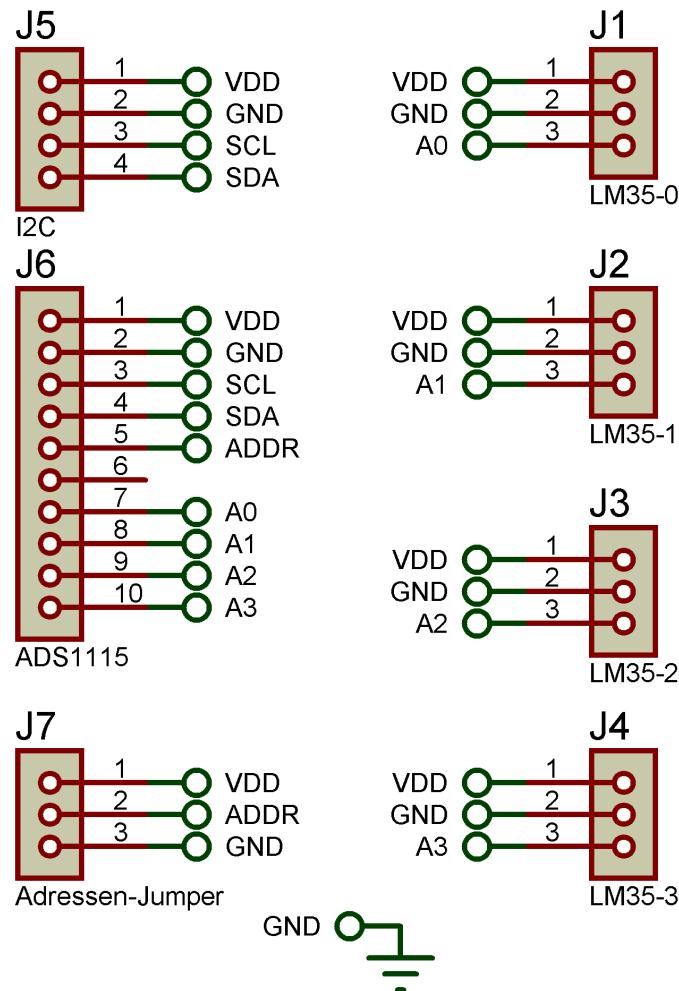


Abbildung 6.21: Schaltplan der Temperatursensoren-Platine

PCB-Layout

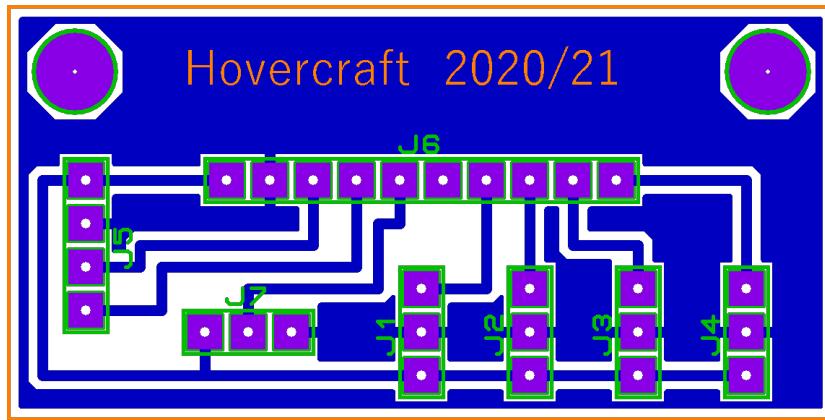


Abbildung 6.22: PCB-Layout der Temperatursensoren-Platine

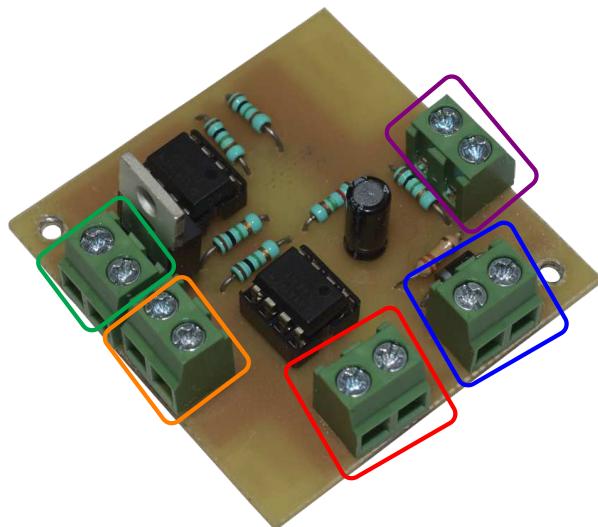
6.3.6 Precharge & Relaisansteuerung

Da sowohl die Motorregler als auch die Buck-Converter einen, bedingt durch die darin verbauten sehr großen Kondensatoren, sehr großen Einschaltstrom haben, müssen diese über einen Leistungswiderstand vorgeladen werden, bevor sie permanent mit Spannung versorgt werden können.



Abbildung 6.23: Zur Vorladung verwendeter Leistungswiderstand

Um diese Umschaltung so einfach wie möglich zu gestalten, wurde dies mithilfe einer im Folgenden genauer erläuterten Platine realisiert.



Hauptrelaisspule (A- & GND) (J6)
Micro-Switches im Ladedeckel (J1)
Buck-Converter (nicht verwendet) (J2)
+30 V von geschalteter Relaisseite (J7)
Konstante +30 V (J8)

Abbildung 6.24: Platine zur Relaisansteuerung

Anmerkung:

Die Relais der Buck-Converter werden nicht mehr, wie ursprünglich gedacht, über eine separate Zeitkonstante angesteuert, sondern verwenden nun den selben MOSFET wie das Hauptrelais.

Funktionsweise

Durch das Zusammenschalten der Akkus über den bereits zuvor gezeigten 10Ω Leistungswiderstand beginnen sich sowohl die Kondensatoren beider Motorregler und aller vier Buck-Converter, sowie jener des sich auf der Platine befindlichen RC-Gliedes aufzuladen.

Erreicht dieser eine bestimmte Schwellspannung, wird das Hauptrelais mithilfe eines MOSFETs durch den Operationsverstärker der Komperatorschaltung aktiviert.

Dies führt zu einer Selbsthaltung des Relais, welche nur durch Betätigung des Not-Aus-Schalters oder das Öffnen einer der beiden Mikroschalter unterbrochen werden kann.

Der genaue Anschluss an die Leistungselektronik kann Abbildung 5.7 entnommen werden.

Schaltplan

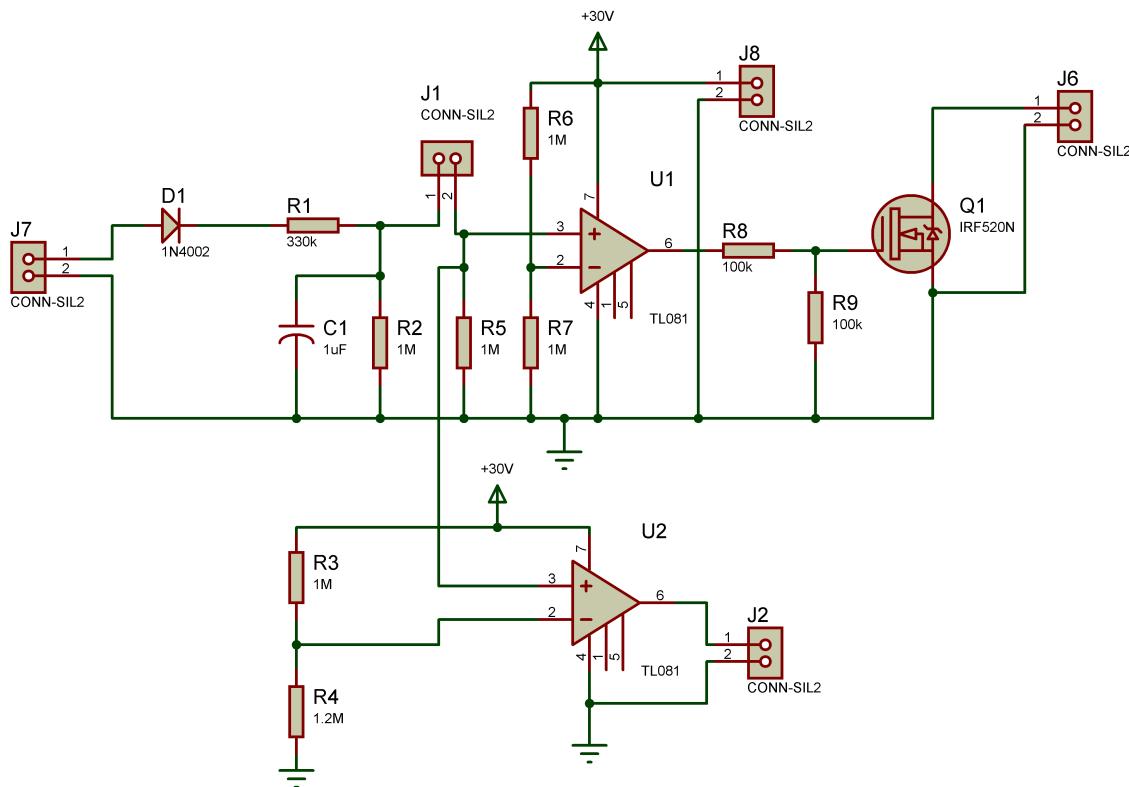


Abbildung 6.25: Schaltplan der Platine zur Relaisansteuerung

Anmerkung:

Der OPV **U2** und die damit verbundene Klemme **J2** werden aufgrund einer nachträglichen Änderung im Schaltplan nicht mehr benötigt.

PCB-Layout

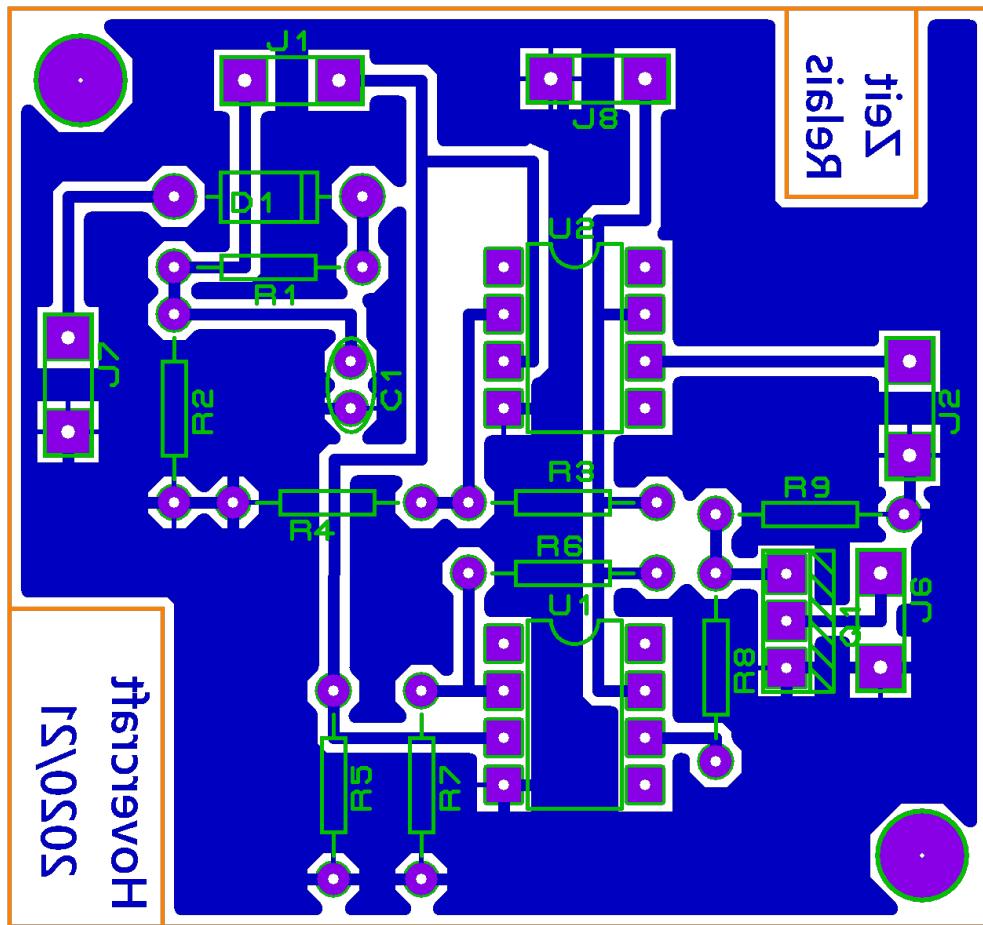


Abbildung 6.26: PCB-Layout der Platine zur Relaisansteuerung

6.3.7 Platzierung am Boot

Die Abbildungen 6.27 & 6.28 zeigen die Positionen aller verbauten Platinen am Boot.

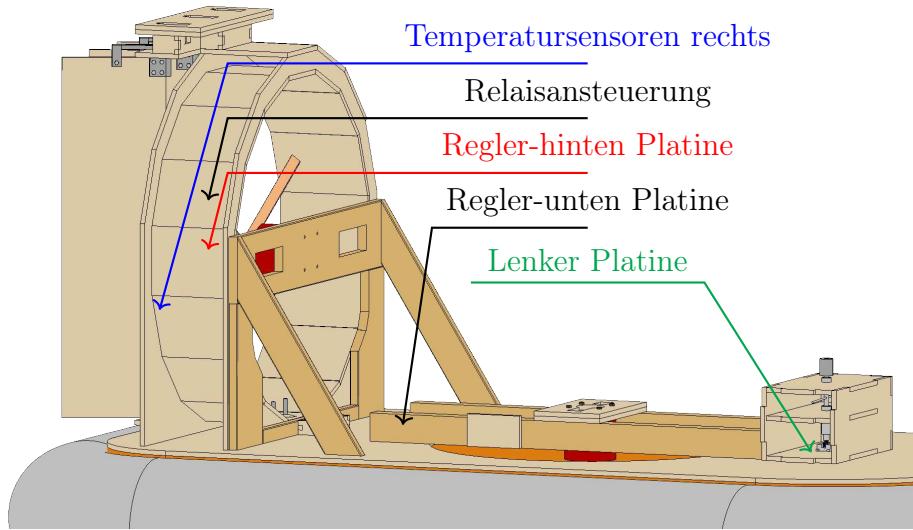


Abbildung 6.27: Platinen auf der rechten Seite des Bootes

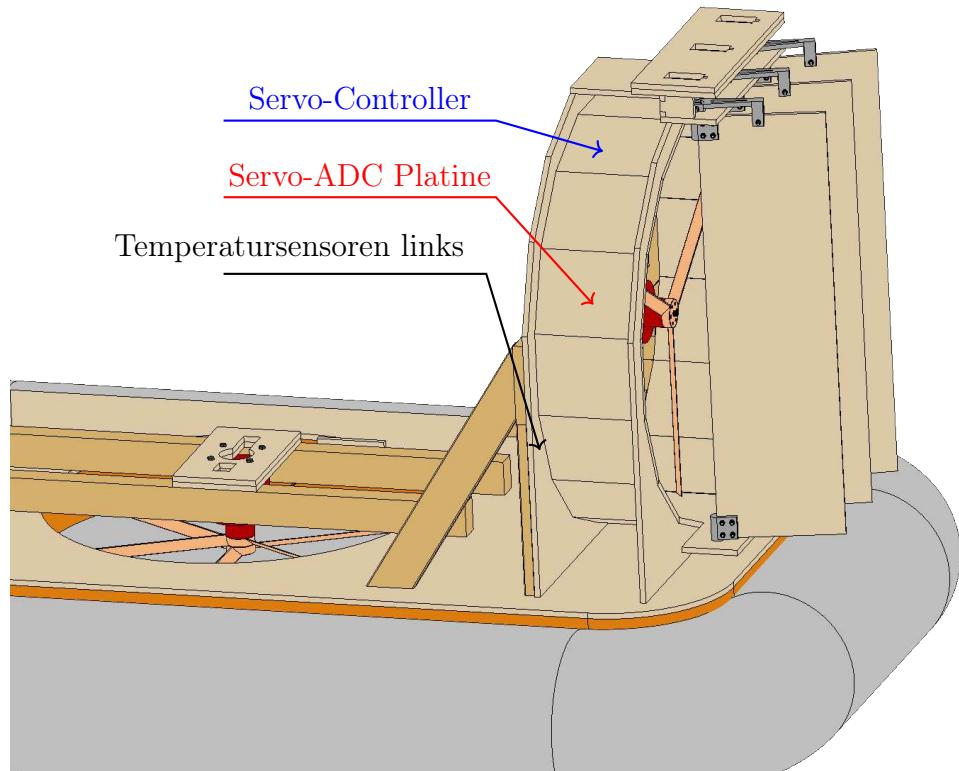


Abbildung 6.28: Platinen auf der linken Seite des Bootes

6.4 Bildschirm

Bei dem verwendeten Bildschirm handelt es sich um einen 5 Zoll großen Touchscreen aus dem Hause Nextion. Dieser bietet den großen Vorteil, die graphische Darstellung bequem über das verfügbare Gratis-Tool (**Nextion Editor**) designen zu können, während der Arduino den Bildschirm nur mit den nötigen Daten über eine serielle Verbindung versorgen muss.



Abbildung 6.29: Nextion-Display

6.4.1 Installation des Nextion-Editors

Nextion Editor Webseite

Das Programm kann entweder per heruntergeladener .exe-Datei installiert werden oder nach dem Entpacken der ebenfalls vorhandenen zip-Datei direkt ausgeführt werden.

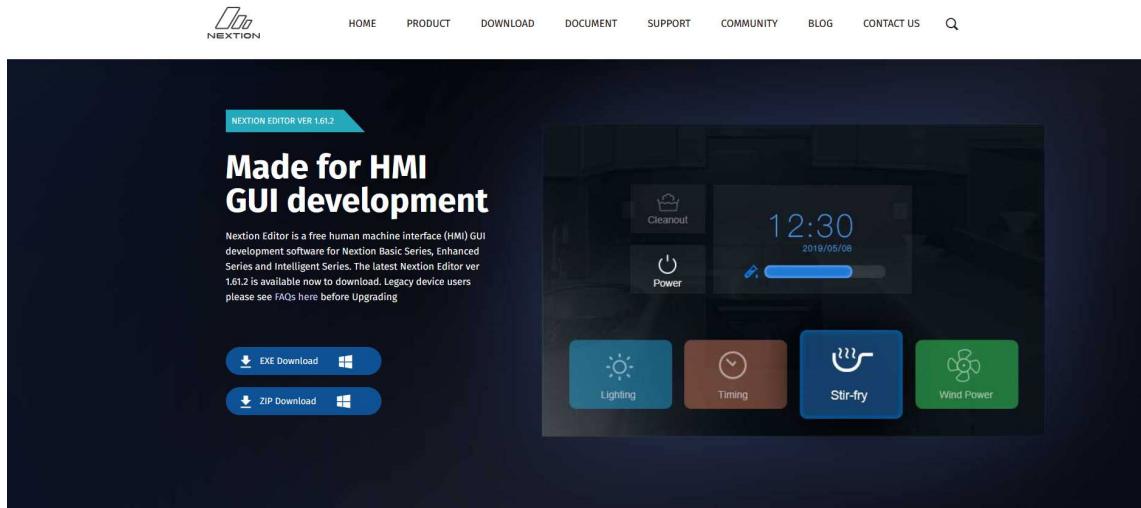


Abbildung 6.30: Webseite des Nextion Editors

6.4.2 Auswahl des richtigen Bildschirmes

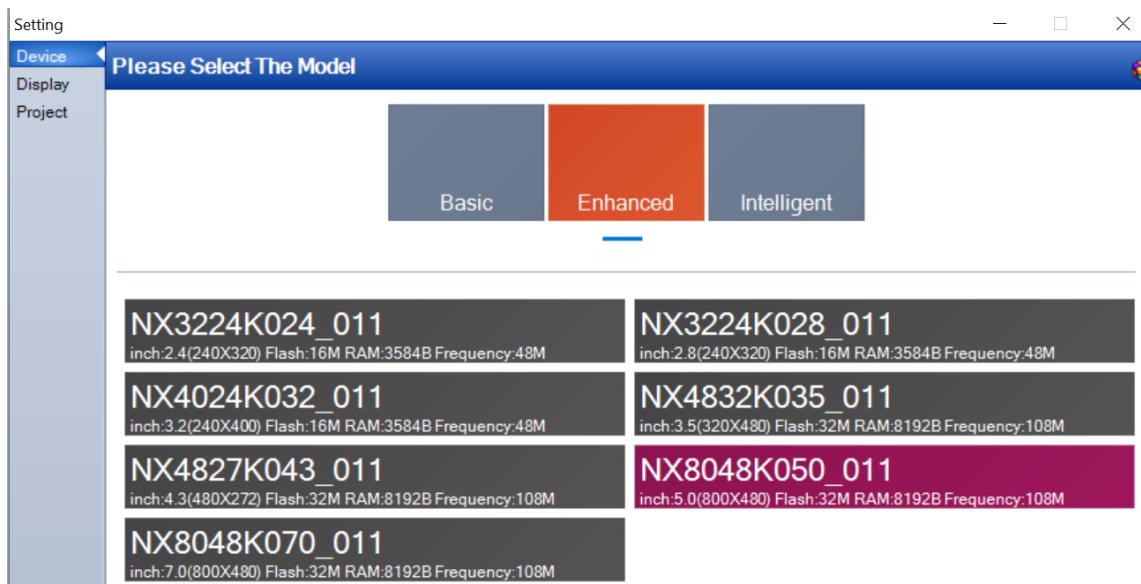


Abbildung 6.31: Auswahl des richtigen Bildschirmes im Nextion Editor

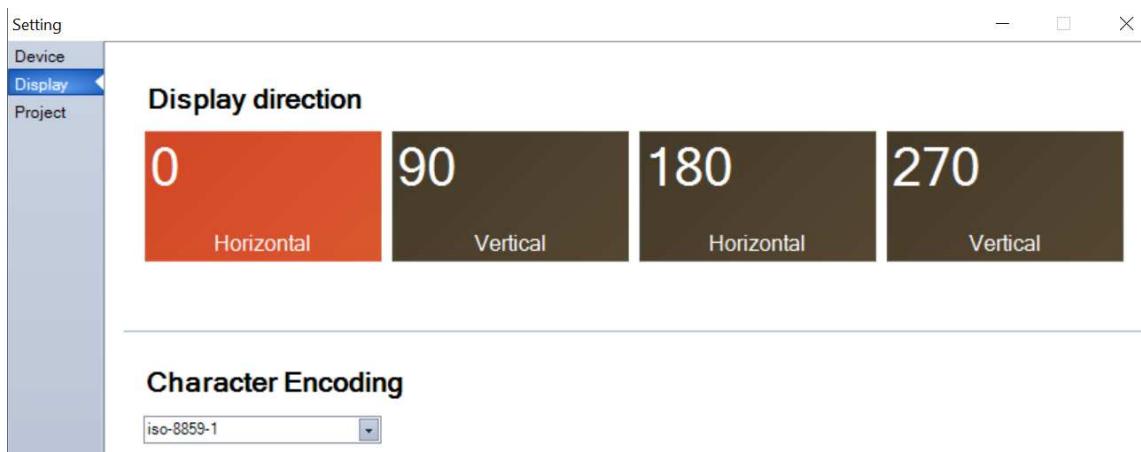


Abbildung 6.32: Auswahl der richtigen Bildschirmausrichtung im Nextion Editor

6.4.3 Entworfene Oberfläche

Hauptseite

Diese Seite zeigt die Telemetriedaten der beiden Regler, sowie die derzeitige vom GPS berechnete Geschwindigkeit an.

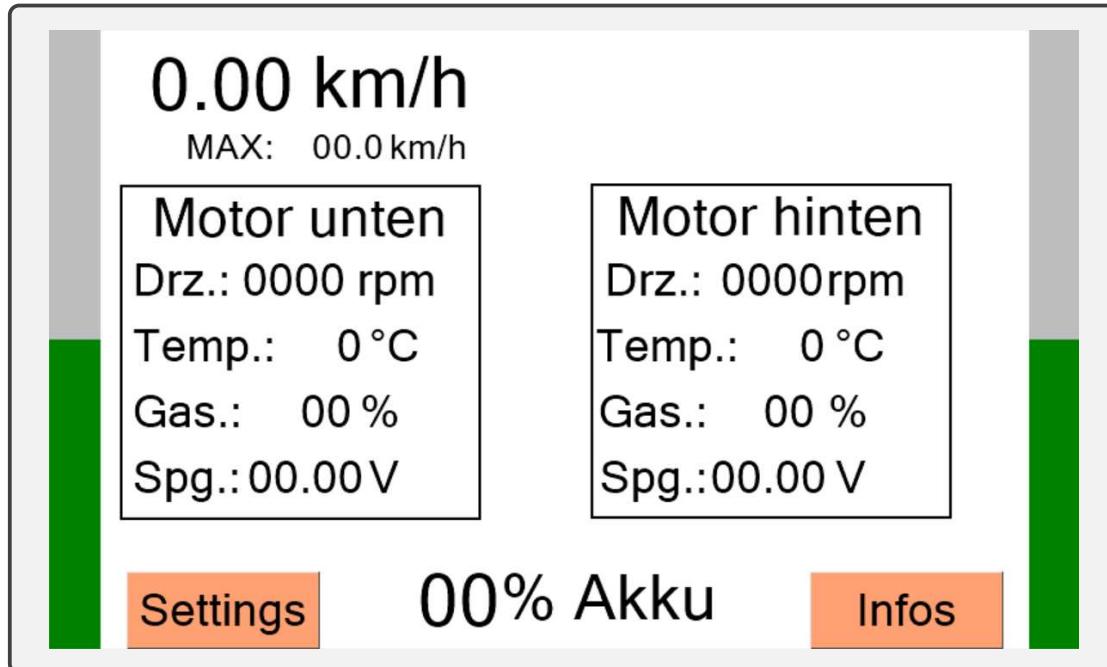


Abbildung 6.33: Hauptseite der Bedienoberfläche

Einstellungen

Während man mit dem ersten Slider auf dieser Seite den Nullpunkt der Lenkung einstellen kann, dienen die beiden unteren Schieberegler zur Begrenzung der maximal verfügbaren Leistung der beiden Motoren.

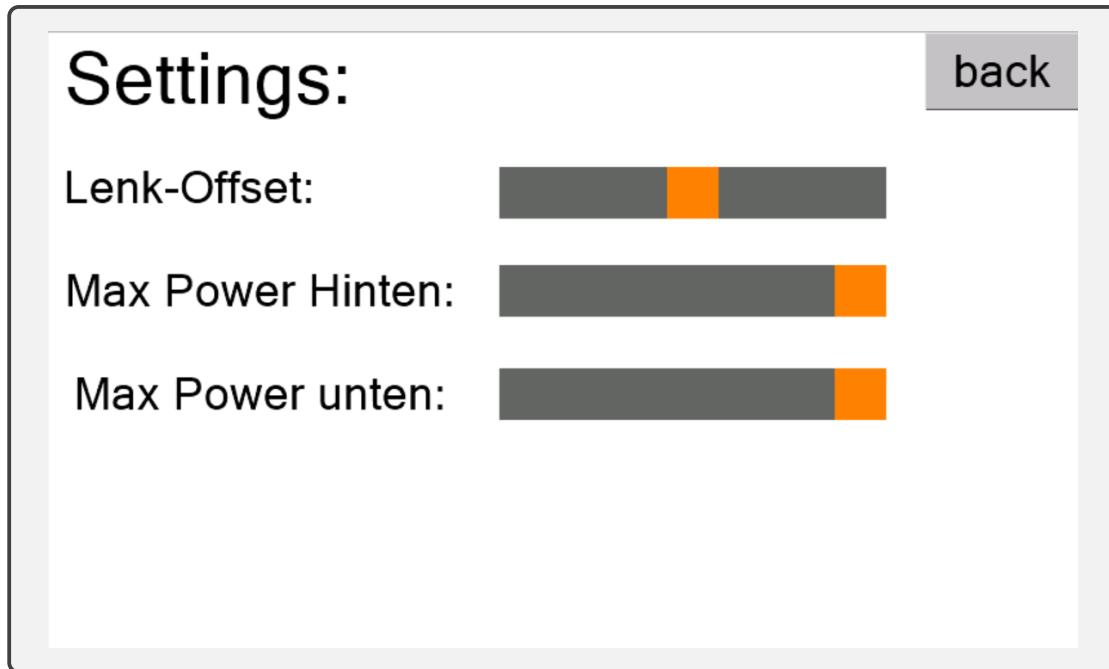


Abbildung 6.34: Einstellungsseite der Bedienoberfläche

Informationen

Diese Seite zeigt sowohl die Akkutemperaturen als auch einige wichtige Informationen über das GPS-Modul an.

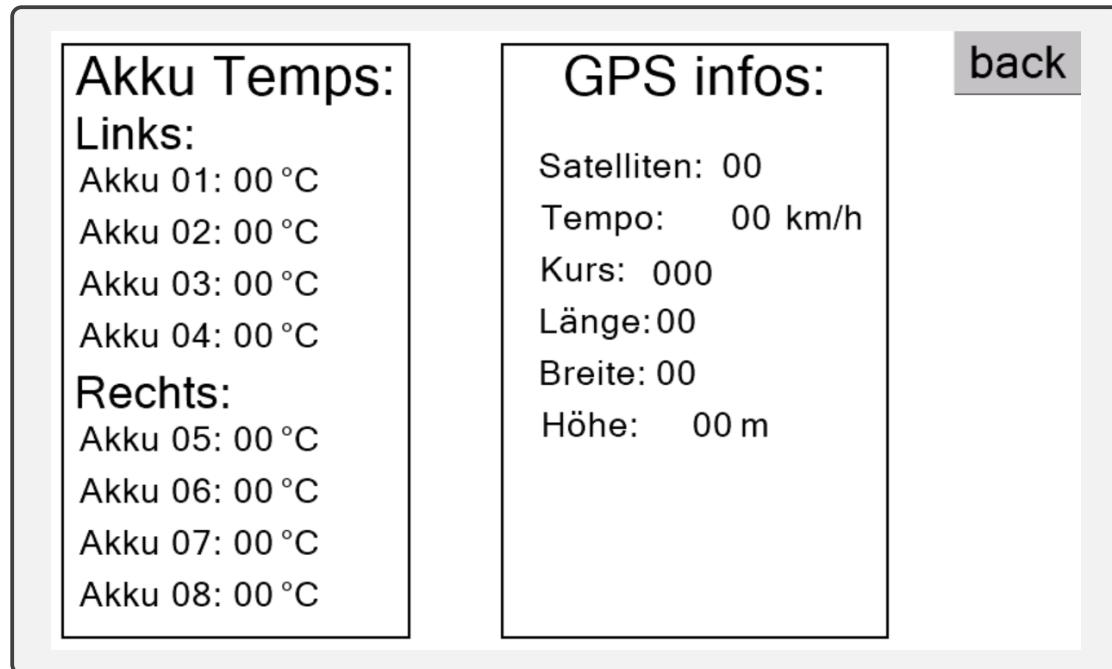


Abbildung 6.35: Infoseite der Bedienoberfläche

6.5 Stückliste

Name	Stück	Händler	Artikel Nr.
Leistungswiderstand 10 Ω	1	Amazon	Link
JST-XHP Verbinder Set	1	Amazon	Link
MCP2515 CAN Bus Shield	4	Amazon	Link
Arduino Nano	4	Amazon	Link
NEO-8M GPS Modul	1	Amazon	Link
ADS 1115 ADC Modul	2	Amazon	Link
16 Channel Servo Driver	1	Amazon	Link
Nextion Enhanced Display	1	itead.cc	IM160511007
Daumengas	2	Amazon	Link
Omrom Mikroschalter	2	RS-Components	682-2314
10 k Ω Widerstand	2	RS-Components	707-8906
100 k Ω Widerstand	2	RS-Components	707-8388
330 k Ω Widerstand	1	RS-Components	707-8962
1 M Ω Widerstand	5	RS-Components	707-7903
1.2 M Ω Widerstand	1	RS-Components	484-6311
1N4007 Diode	1	RS-Components	700-3665
Elko 1 μ F	1	RS-Components	711-1671
MCP6242 5 V OPVs	2	RS-Components	403-165
LM7805CT Spannungsregler	1	RS-Components	796-8060
LM35 Temperatursensor	8	RS-Components	811-5595
N-Kanal MOSFET	1	RS-Components	541-1180
TL081CP OPV	2	RS-Components	304-223
Leiterplattenklemmen	10	RS-Components	790-1064
Potenziometer	1	RS-Components	729-3492
Druckschalter Tastend	1	RS-Components	811-8480
Not-Aus Schlüsseltaster	1	Schrack	MM216514-

Tabelle 6.2: Stückliste der Boardelektronik

7 Sicherheitskonzept

Um die Sicherheit während der Benützung des Luftkissenboots zu gewährleisten, wurden mehrere Sicherheitsvorkehrungen getroffen.

7.1 Sicherheitsmaßnahmen

7.1.1 Not-Aus-Schalter mit Schlüssel

Das Betätigen des Not-Aus-Schalters führt zu einer Unterbrechung des Haltestroms für das Hauptrelais. Das heißt, das Relais fällt ab und unterbricht die beiden Akkustränge, was den Stromkreis öffnet.

Der Not-Aus-Schalter muss nach Betätigung mittels Schlüssel entsperrt werden.



Abbildung 7.1: Not-Aus-Schlüsselschalter

7.1.2 Daumengas für beide Propeller

Die beiden Motoren werden durch zwei Daumengashebel am Lenker angesteuert. Sollte der Fahrer vom Lenker abrutschen, lässt er somit automatisch die Daumengashebel los, was zu einem Bremsvorgang der Motoren führt.



Abbildung 7.2: Daumengas

7.1.3 Gitterabdeckungen

Die beiden Propeller sind durch Gitter geschützt, die es unmöglich machen, unbewusst in ein drehendes Teil zu fassen. Das untere Gitter ist so konzipiert, dass es das Gewicht des Fahrers problemlos tragen kann.

Außerdem dienen die Gitter als Schutz vor losen Teilen (wie zum Beispiel Steinen), die sich in den Propellern verfangen könnten.

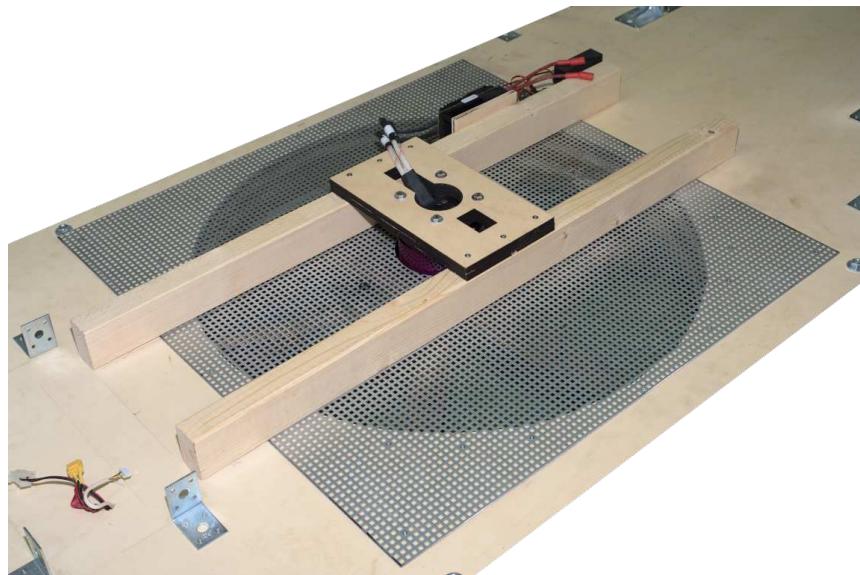


Abbildung 7.3: Unteres Gitter

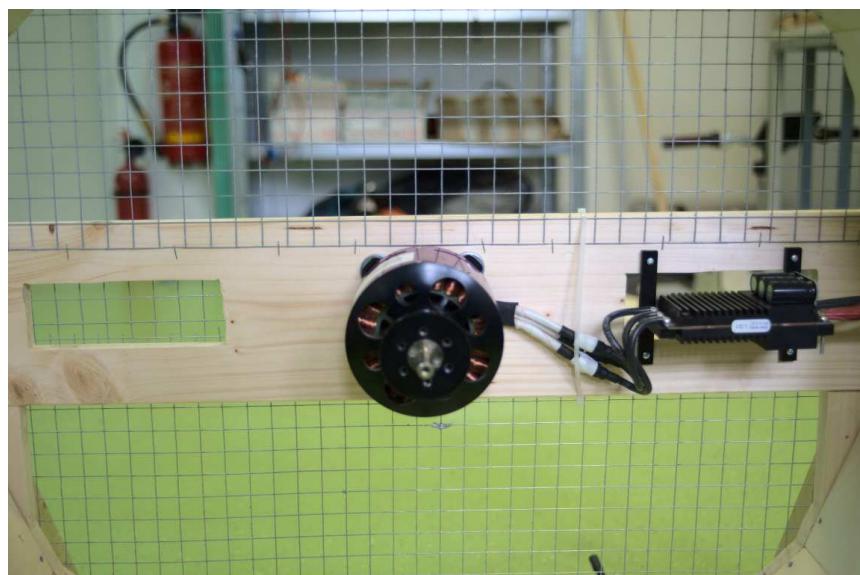


Abbildung 7.4: Hinteres Gitter

7.2 Gerätesicherheit

7.2.1 Ladevorgang

Während des Ladevorgangs werden die Minuspole der beiden in Serie geschaltenen Akkustränge verbunden. Um den Kurzschluss eines Akkustranges zu verhindern, muss deshalb das Hauptrelais geöffnet sein.

Zum Schutz vor Bedienungsfehlern wurden Schalter in die Kabelbox verbaut. Diese unterbrechen bei offenem Deckel, und damit vorhandenem Zugang zu den Ladekabeln, den Haltestrom des Hauptrelais.

Trotz dieser Sicherheitsvorkehrung muss der Not-Aus-Schalter während des Ladevorgangs gedrückt sein.

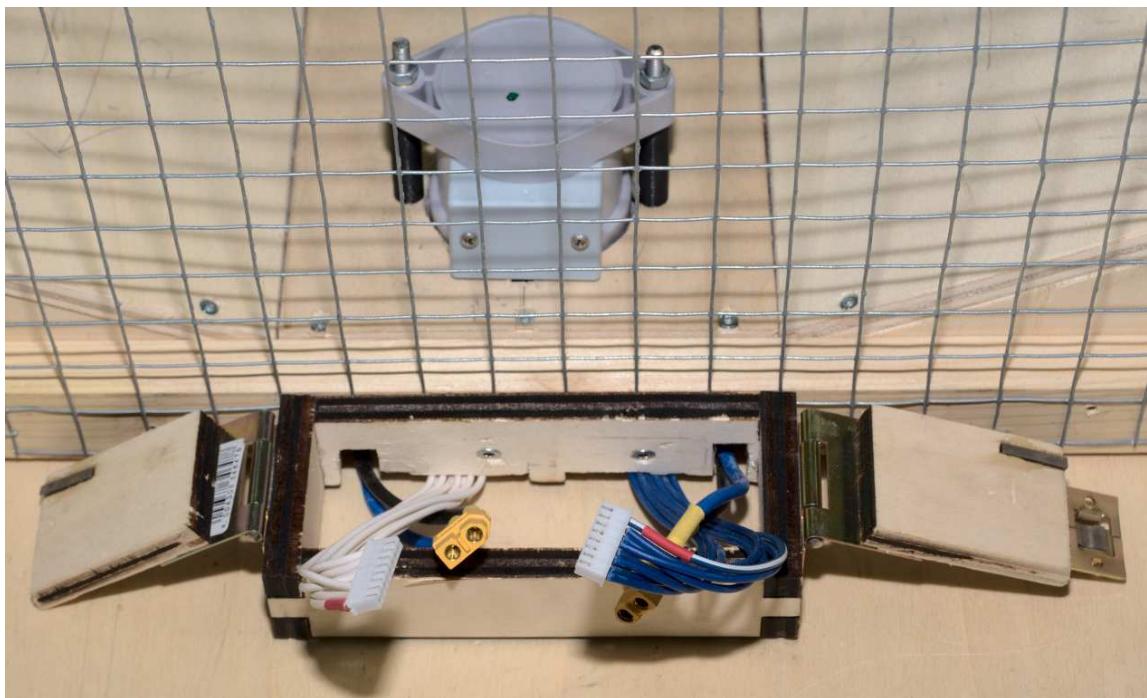


Abbildung 7.5: Offene Ladebox & Ladekabel

In der Abbildung 7.5 sind die Endschalter noch nicht vorhanden, da sie erst zu einem späteren Zeitpunkt verbaut wurden.

7.2.2 Bremsschutz

Zum Schutz des Schlauchbootbodens beim Bremsen wurde ein zusätzlicher Boden angebracht. Dieser wurde mit einem Kraftkleber und Lochband am Boot befestigt. Der zusätzliche Boden bietet Schutz vor Abschürfungen, die während des Bremsvorgangs entstehen, und vor Steinen, welche das Boot zerstören könnten.



Abbildung 7.6: Bremsschutz

7.2.3 Reglerkühlung

Die beiden Motorregler sind so positioniert, dass sie durch den vom Propeller erzeugten Luftstrom gekühlt werden. Außerdem schalten die Regler automatisch ab, sobald die innere Temperatur einen Schwellenwert von 90°C überschreitet.

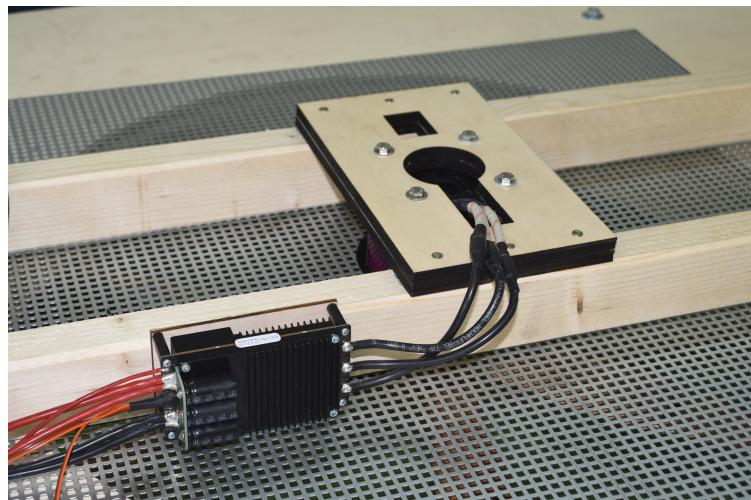


Abbildung 7.7: Position des Motorreglers

8 Kostenrechnung

8.1 Konstruktion

Beschreibung	Kosten
diverse Kleinteile	60€
Fahrradlenker	21€
Holzlatten & Staffel	21€
Holzschrauben	15€
Lochband	14€
metrische Schrauben & Muttern	60€
Pappelsperrholz	80€
Pattex Kraftkleber	18€
PETG Filament	22€
Propeller	276€
Teichfolie	22€
Winkelverbinde	50€
XPS	74€
Zwischensumme	733€

Tabelle 8.1: Kostenrechnung – Konstruktion

8.2 Leistungselektronik

Beschreibung	Kosten
Akkus	1 672€
Buck-Converter	47€
Hauptrelais	162€
Kupferschiene	62€
Ladegerät & Schaltnetzteil	520€
Motoren	2 100€
Motorregler & Jeti-Box	1 079€
Schmelzsicherungen	15€
Servos	135€
Zwischensumme	5 792€

Tabelle 8.2: Kostenrechnung – Leistungselektronik

8.3 Boardelektronik

Beschreibung	Kosten
4-poliges Kabel 100m	30€
Adafruit Servo-Modul	13€
Arduino-Nanos	20€
CAN-bus Module	33€
Daumengashebel	28€
diverse Kleinteile	200€
diverse Steckverbinder	200€
GPS-Modul	13€
Nextion-Display	60€
Not-Aus-Schalter	68€
Zwischensumme	665€

Tabelle 8.3: Kostenrechnung – Leistungselektronik

8.4 Gesamtkosten

Beschreibung	Kosten
Konstruktion	733€
Leistungselektronik	5 897€
Boardelektronik	665€
Summe	7 190€

Tabelle 8.4: Kostenrechnung – Gesamtkosten

Anmerkung:

Die hohen Gesamtkosten entstanden durch die hohe benötigte Motorleistung, welche den Preis der Motoren, der Motorregler und der Akkus begründet. Deshalb konnte diese Diplomarbeit nur durch die finanzielle Unterstützung unserer Sponsoren verwirklicht werden.

9 Anleitungen

9.1 Betriebsanleitung

1. Not-Aus-Schalter entriegeln
2. Starttaster gedrückt halten, bis man das Klicken des Hauptrelais hört
3. Fahrspaß genießen:
 - rechter Daumengashebel: Auftrieb
 - linker Daumengashebel: Vorschub
4. Verzögerungsvorgang: Vorschub abdrehen und Reibung durch Verringerung des Auftriebs erhöhen
5. Not-Aus-Schalter drücken, um das Hovercraft ganz abzuschalten

9.2 Anleitung Ladevorgang

Während des Ladevorgangs darf der Starttaster **nicht** gedrückt werden!

1. Not-Aus-Schalter drücken
2. Ladedeckel öffnen
3. Ladegerät anschließen: Am Ladegerät werden die Zellenspannungen angezeigt.

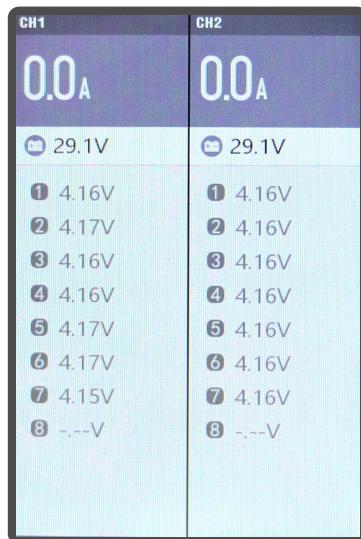


Abbildung 9.1: Ladegerät Hauptanzeige

4. Ladevorgang starten:

- a) Die beiden mittleren Tasten des Ladegerätes drücken und die Option „Dual Task“ auswählen.

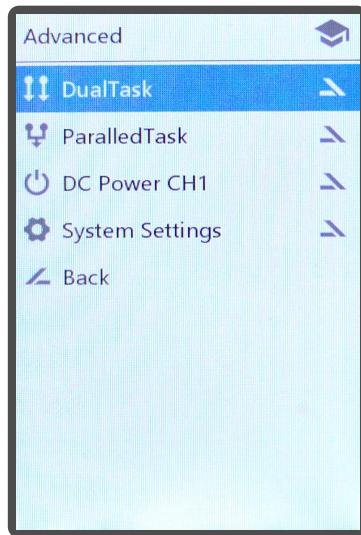


Abbildung 9.2: Ladegerät Task-Auswahl

- b) Die Parameter wie in Abbildung 9.3 einstellen und mit „Start“ den Ladevorgang starten.

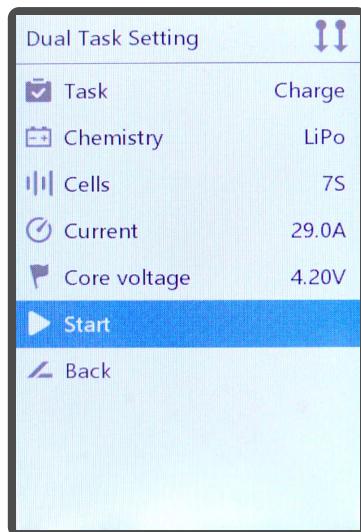


Abbildung 9.3: Ladegerät Einstellungen

5. Während des Schnellladevorgangs leuchtet der obere Bereich des Displays orange – siehe Abbildung 9.4



Abbildung 9.4: Ladegerät Schnellladevorgang

6. Sobald der obere Bereich des Displays grün leuchtet, ist das Schnellladen abgeschlossen, aber der Ladevorgang ist noch nicht fertig.
7. Erst wenn der obere Bereich des Displays blau leuchtet, ist der Ladevorgang abgeschlossen und kann durch Klicken auf die mittleren Touch-Buttons beendet werden.
8. Ladegerät abschließen & Kabel in Ladebox verstauen
9. Ladedeckel schließen

10 Besprechungsprotokolle & Meilensteine

10.1 Besprechungsprotokolle

Anwesende Schüler	Anwesender Betreuer	Datum	Inhalt
Deimel, Eilmsteiner, Stöger	Prof. Frank	28.02.2020	Organisation der Diplomarbeit
Deimel, Eilmsteiner, Stöger	Prof. Frank	12.03.2020	Besprechung betreffend Sponsoring
Deimel, Eilmsteiner, Stöger	Prof. Frank	18.06.2020	Besprechung der Leistungselektronik
Deimel, Eilmsteiner, Stöger	Prof. Frank	29.07.2020	Festlegung der Meilensteine
Deimel, Eilmsteiner, Stöger	Prof. Frank	23.10.2020	Planung der Konstruktion
Deimel, Eilmsteiner, Stöger	Prof. Frank	10.11.2020	Planung der Lenkung
Deimel, Eilmsteiner, Stöger	Prof. Frank	21.11.2020	Planung der Elektronik

10.2 Meilensteine

1. September 2020 - Überprüfung des Konzepts und der notwendigen Motorleistung mithilfe eines Prototyps

Der Meilenstein zur Überprüfung der Motorleistung wurde eingehalten. Der Prototyp konnte den Schwebezustand erreichen.

1. November 2020 - Propeller für Fortbewegung mit dazugehörigen Fahnen für Lenkung ist funktionstüchtig

Dieser Meilenstein wurde rechtzeitig eingehalten und die Lenkung war somit funktionstüchtig.

1. Dezember 2020 - Boardelektronik und Akkumanagement vollständig entwickelt

Dieser Meilenstein wurde rechtzeitig eingehalten und das Konzept und Design der gesamten Boardelektronik und des Akkumanagements war vollständig entwickelt.

1. Februar 2021 - Akkumanagement funktionstüchtig, getestet und im Luftkissenboot verbaut

Auf Grund der langen Lieferzeiten des Ladegerätes konnte dieser Meilenstein erst zwei Wochen später erreicht werden.

20. Februar 2021 - Gesamte Boardelektronik fertiggestellt

Dieser Meilenstein wurde rechtzeitig eingehalten und somit war die komplette Boardelektronik fertig im Luftkissenboot verbaut, funktionstüchtig und getestet.

1. März 2021 - Praktischer Teil der Diplomarbeit beendet

Da uns leider am 28. Februar 2021 bei der ersten Testfahrt ein Motorregler abgebrannt ist und sich somit auch das Hauptrelais verschweißt hat, mussten diese Bauteile nachbestellt werden. Aus diesem Grund konnte dieser Meilenstein erst ein paar Tage später, am **4. März 2021**, erreicht werden.

15. März 2021 - Diplomarbeit fertig geschrieben

Dieser Meilenstein wurde ein paar Tage danach erreicht, am **19. März 2021**. Das Hovercraft war somit fertig gebaut und die Dokumentation war ebenfalls fertig geschrieben.

11 Zeitaufzeichnung

11.1 Stefan Deimel

Datum	Zeit / h	Tätigkeit
14.02.2020	8	Konstruktion in Inventor
15.02.2020	4	Konstruktion in Inventor
28.02.2020	1	Besprechung mit Betreuungslehrer
12.03.2020	1	Besprechung mit Betreuungslehrer
13.03.2020	4	Erstes Zusammenbauen des Hovercrafts
14.05.2020	5	Akkukabel gebaut, Motorregler & Motorstecker montiert
23.05.2020	5	Propeller montiert, erste Tests am Hovercraft
06.06.2020	7	Jeti-Protokoll decodiert für Arduino
18.06.2020	1	Besprechung mit Betreuungslehrer, Bestellung vorbereitet
27.07.2020	1	Verbesserung der Konstruktion in Inventor
29.07.2020	1	Besprechung auf Teams, Festlegung von Meilensteinen
03.08.2020	8	Propeller zusammengebaut
04.08.2020	3	Tests am Hovercraft
10.08.2020	8	Bodenplatte/XPS gefertigt
11.08.2020	7	Bodenplatte/XPS + Holz gefertigt
14.08.2020	5	Motor montiert + Tests
25.08.2020	7	Transport des Luftkissenbootes in die Schule, Treffen mit SchrackForStudents
17.09.2020	2	Bestellliste für Elektronik erstellt, Bauteile ausgewählt
20.09.2020	2	Bestellliste für Elektronik erstellt, Bauteile ausgewählt
25.09.2020	2	Teichfolie geklebt
21.10.2020	6	Konstruktion in Inventor – Aufbau hinten
22.10.2020	2	Konstruktion in Inventor – Aufbau hinten, Fahnen + Halterung
23.10.2020	1	Logos von Kustec abgeholt
23.10.2020	2	Besprechung auf Teams, Obi-Einkauf & nächste Schritte geplant
24.10.2020	2	Obi-Einkauf
26.10.2020	1,5	Programmierung JetiBox-Decode für Arduino verbessert (Buffer hinzugefügt)
29.10.2020	6	Verbesserung der Konstruktion in Inventor

Tabelle 11.1: Zeitaufzeichnung Stefan Deimel

Datum	Zeit / h	Tätigkeit
30.10.2020	6,5	Verbesserung der Konstruktion in Inventor
09.11.2020	4	Konstruktion hinten am Boot befestigt
10.11.2020	3	Verstärkung des hinteren Aufbaues
10.11.2020	1	Besprechung mit Betreuungslehrer (Lenkung, derzeitiger Fortschritt)
17.11.2020	2	Test der Fahnensteuerung mit Servos
21.11.2020	2	Besprechung Teams: Elektronik und Lenkung, nächste Bestellung geplant
21.11.2020	0,5	Bauteile ausgewählt
21.11.2020	1	Lenkungsentwurf in Inventor gezeichnet
25.11.2020	6	hinteres Gitter montiert, Teichfolie geklebt
28.11.2020	1	RS-Bauteile ausgewählt
29.11.2020	2	Lenker in Inventor konstruiert
30.11.2020	1	Verbindungsteile für Lenkerstange in Inventor konstruiert
16.12.2020	4	Erste Versuche mit der fertigen Konstruktion
15.12.2020	3	Lasercutten
27.12.2020	2	Lasercutten und Inventor-Dateien vorbereitet
28.12.2020	10,5	Servos montiert, Lenkung getestet, CAN-Bus getestet
28.12.2020	3	Kupferschiene geschnitten (Gewinde), Schrauben zugeschnitten
29.12.2020	7	Kupferschienen und Bucks montiert
01.01.2021	1	M5 Schrauben geschnitten
02.01.2021	12	Kupferschiene und Temperatursensoren verkabelt, Reglerplatine gezeichnet
03.01.2021	11	Servos verkabelt, Lenkung montiert
19.01.2021	5	Platinen hergestellt, geätzt, gebohrt
25.01.2021	8	Taster, Not-Aus-Schalter montiert, HMI getestet, Balancer-Kabel gelötet
26.01.2021	5	weitere Platinen hergestellt, geätzt, gebohrt
29.01.2021	9	Platinen gebohrt, gelötet und im Hovercraft verbaut, Stecker an Kabel gelötet
30.01.2021	11	Erste Einschaltversuche durchgeführt
02.02.2021	8	Display getestet, Einschaltverzögerungsplatine getestet
02.02.2021	1	Proteus Platine Zeitverzögerung umgezeichnet
07.02.2021	3	Inventor-Modell in einzelnen Bauteile zerlegt, Doku weitergeschrieben

Tabelle 11.1: Zeitaufzeichnung Stefan Deimel

Datum	Zeit / h	Tätigkeit
13.02.2021	6	Halbleiterrelais eingebaut - Defekt festgestellt, Platine gelötet, Zeichnungen in Inventor erstellt
15.02.2021	4	Boden geklebt
16.02.2021	4	Lochband fertiggestellt, Leiste fertiggestellt, fotografiert
17.02.2021	11	neues hinteres Gitter montiert, Halbleiterrelais getestet - defekt
02.03.2021	4	Doku Konstruktion weitergeschrieben
01.03.2021	5	Strommessung am unteren Motor
04.03.2021	4	Probefahrt
16.03.2021	5	Doku weitergeschrieben
17.03.2021	2	Doku durchgelesen und verbessert
18.03.2021	3	Doku finalisiert
284		Summe

Tabelle 11.1: Zeitaufzeichnung Stefan Deimel

11.2 Philipp Eilmsteiner

Datum	Zeit / h	Tätigkeit
28.02.2020	1	Besprechung mit Betreuungslehrer
12.03.2020	1	Besprechung mit Betreuungslehrer
13.03.2020	4	Erstes Zusammenbauen des Hovercrafts
14.05.2020	5	Akkukabel gebaut, Motorregler & Motorstecker montiert
23.05.2020	1	Propeller abgeholt
23.05.2020	5	Propeller montiert, erste Tests am Hovercraft
18.06.2020	1	Besprechung mit Betreuungslehrer, Bestellung vorbereitet
21.07.2020	1	Sponsoren angeschrieben
29.07.2020	1	Besprechung auf Teams, Festlegung von Meilensteinen
03.08.2020	8	Propeller zusammengebaut
04.08.2020	3	Tests am Hovercraft
10.08.2020	8	Bodenplatte/XPS gefertigt
11.08.2020	7	Bodenplatte/XPS + Holz gefertigt
14.08.2020	5	Motor montiert + Tests
25.08.2020	7	Transport des Luftkissenbootes in die Schule, Treffen mit SchrackForStudents
26.08.2020	2	Kontakt mit Sponsor SchrackForStudents wegen Obi Rechnungen, Instagram
17.09.2020	2	Bestellliste für Elektronik erstellt, Bauteile ausgewählt
20.09.2020	2	Bestellliste für Elektronik erstellt, Bauteile ausgewählt
25.09.2020	2	Teichfolie geklebt
23.10.2020	2	Besprechung auf Teams, Obi-Einkauf & nächste Schritte geplant
23.10.2020	0,5	3D-Druck
24.10.2020	5	3D-Druck, erste Schritte mit CAN-Bus, Überlegungen zur Ansteuerung der Regler
26.10.2020	2	Programmierung von Gas Übertragung per CAN-Bus und PWM Erzeugung
29.10.2020	6	Verbesserung der Konstruktion in Inventor
30.10.2020	6,5	Verbesserung der Konstruktion in Inventor
08.11.2020	2,5	HMI begonnen
09.11.2020	4	Konstruktion hinten am Boot befestigt
10.11.2020	3	Verstärkung des hinteren Aufbaues

Tabelle 11.2: Zeitaufzeichnung Philipp Eilmsteiner

Datum	Zeit / h	Tätigkeit
10.11.2020	1	Besprechung mit Betreuungslehrer (Lenkung, derzeitiger Fortschritt)
17.11.2020	2	Test der Fahnensteuerung mit Servos
20.11.2020	1	Akkus gemessen + Kabel gelötet
21.11.2020	2	Besprechung Teams: Elektronik und Lenkung, nächste Bestellung geplant
25.11.2020	6	hinteres Gitter montiert, Teichfolie geklebt
27.11.2020	0,5	3D-Druck
28.11.2020	5	Verbessern des Fahnenhalters für Servo-Halterung + Halterung Regler hinten + 3D-Druck
05.12.2020	8	Kommunikation zwischen Lenker und CAN-Bus programmiert sowie mit Versuchsaufbau getestet
15.12.2020	0,5	3D-Druck
16.12.2020	4	Erste Versuche mit der fertigen Konstruktion
27.12.2020	0,5	3D-Druck
28.12.2020	10,5	Servos montiert, Lenkung getestet, CAN-Bus getestet
29.12.2020	7	Kupferschienen und Bucks montiert
30.12.2020	3	Display-Halterung fertig konstruiert und erste 3D-Drucke davon angefertigt
02.01.2021	12	Kupferschiene und Temperatursensoren verkabelt, Reglerplatine gezeichnet
03.01.2021	11	Servos verkabelt, Lenkung montiert
15.01.2021	1	Platinen in Proteus gezeichnet
17.01.2021	2	Platinen in Proteus gezeichnet
19.01.2021	5	Platinen hergestellt, geätzt, gebohrt
25.01.2021	8	Taster, Not-Aus-Schalter montiert, HMI getestet, Balancer-Kabel gelötet
26.01.2021	5	weitere Platinen hergestellt, geätzt, gebohrt
29.01.2021	9	Platinen gebohrt, gelötet und im Hovercraft verbaut, Stecker an Kabel gelötet
30.01.2021	11	Erste Einschaltversuche durchgeführt
31.01.2021	1,5	Display Benutzeroberfläche entworfen
01.02.2021	2	ADCs & HMI implementiert
02.02.2021	8	Display getestet, Einschaltverzögerungsplatine getestet
07.02.2021	3	HMI weiterentwickelt + Boardelektronik in Doku begonnen
08.02.2021	2	Bauteile fotografiert + Editierung in GIMP
12.02.2021	2	Platinen gefertigt

Tabelle 11.2: Zeitaufzeichnung Philipp Eilmsteiner

Datum	Zeit / h	Tätigkeit
13.02.2021	6	Halbleiterrelais eingebaut - Defekt festgestellt, Platine gelötet, Hornbach eingekauft
15.02.2021	4	Boden geklebt
15.02.2021	2	Fotos zugeschnitten, Schreiben an Boardelektronik in Doku
16.02.2021	4	Lochband fertiggestellt, Leiste fertiggestellt, fotografiert
16.02.2021	3	Fotos zugeschnitten, Doku weitergeschrieben
17.02.2021	11	neues hinteres Gitter montiert, Halbleiterrelais getestet - defekt
02.03.2021	4	Gimp, Doku
01.03.2021	5	Strommessung am unteren Motor
04.03.2021	4	Probefahrt
16.03.2021	5	Doku weitergeschrieben
18.03.2021	3	Doku finalisiert
282		Summe

Tabelle 11.2: Zeitaufzeichnung Philipp Eilmsteiner

11.3 Julia Stöger

Datum	Zeit / h	Tätigkeit
28.02.2020	1	Besprechung mit Betreuungslehrer
12.03.2020	1	Besprechung mit Betreuungslehrer
13.03.2020	4	Erstes Zusammenbauen des Hovercrafts
14.05.2020	5	Akkukabel gebaut, Motorregler & Motorstecker montiert
23.05.2020	5	Propeller montiert, erste Tests am Hovercraft
18.06.2020	1	Besprechung mit Betreuungslehrer, Bestellung vorbereitet
23.07.2020	3	Sponsoren angeschrieben
29.07.2020	1	Besprechung auf Teams, Festlegung von Meilensteinen
03.08.2020	8	Propeller zusammengebaut
04.08.2020	3	Tests am Hovercraft
10.08.2020	8	Bodenplatte/XPS gefertigt
11.08.2020	7	Bodenplatte/XPS + Holz gefertigt
14.08.2020	5	Motor montiert + Tests
25.08.2020	7	Transport des Luftkissenbootes in die Schule, Treffen mit SchrackForStudents
17.09.2020	2	Bestellliste für Elektronik erstellt, Bauteile ausgewählt
20.09.2020	2	Bestellliste für Elektronik erstellt, Bauteile ausgewählt
25.09.2020	2	Teichfolie geklebt
23.10.2020	2	Besprechung auf Teams, Obi-Einkauf & nächste Schritte geplant
29.10.2020	6	Verbesserung der Konstruktion in Inventor
30.10.2020	6,5	Verbesserung der Konstruktion in Inventor
09.11.2020	4	Konstruktion hinten am Boot befestigt
10.11.2020	3	Verstärkung des hinteren Aufbaues
10.11.2020	1	Besprechung mit Betreuungslehrer (Lenkung, derzeitiger Fortschritt)
17.11.2020	2	Test der Fahnensteuerung mit Servos
21.11.2020	2	Besprechung Teams: Elektronik und Lenkung, nächste Bestellung geplant
25.11.2020	6	hinteres Gitter montiert, Teichfolie geklebt
16.12.2020	4	Erste Versuche mit der fertigen Konstruktion
29.12.2020	7	Kupferschienen und Bucks montiert
02.01.2021	12	Kupferschiene und Temperatursensoren verkabelt, Reglerplatine gezeichnet

Tabelle 11.3: Zeitaufzeichnung Julia Stöger

Datum	Zeit / h	Tätigkeit
03.01.2021	11	Servos verkabelt, Lenkung moniert
19.01.2021	5	Platinen hergestellt, geätzt, gebohrt
25.01.2021	8	Taster, Not-Aus-Schalter montiert, HMI getestet, Balancer-Kabel gelötet
26.01.2021	5	weitere Platinen hergestellt, geätzt, gebohrt
29.01.2021	9	Platinen gebohrt, gelötet und im Hovercraft verbaut, Stecker an Kabel gelötet
30.01.2021	11	Erste Einschaltversuche durchgeführt
31.01.2021	1,5	Display Benutzeroberfläche entworfen
02.02.2021	8	Display getestet, Einschaltverzögerungsplatine getestet
12.02.2021	2	Platinen gefertigt
13.02.2021	5	Halbleiterrelais eingebaut - Defekt festgestellt, Platine gelötet
15.02.2021	4	Boden geklebt
16.02.2021	4	Lochband fertiggestellt, Leiste fertiggestellt, fotografiert
16.02.2021	3	Funktionsprinzip gezeichnet, Doku weitergeschrieben
17.02.2021	11	neues hinteres Gitter montiert, Halbleiterrelais getestet - defekt
01.03.2021	5	Strommessung am unteren Motor
04.03.2021	4	Probefahrt
14.03.2021	2	Doku weitergeschrieben
15.03.2021	3	Doku weitergeschrieben
16.03.2021	5	Doku weitergeschrieben
18.03.2021	3	Doku finalisiert
230		Summe

Tabelle 11.3: Zeitaufzeichnung Julia Stöger

12 Anhang

12.1 Arduino Bibliotheken

Im Zuge der Programmierung der Arduinos wurde auf eine Vielzahl von bereits vorhandenen Bibliotheken zurückgegriffen. Konkret wurden für die CAN-Bus Module^[6], das Servo-Driver-Modul^[7], die ADS1115 Module^[8], das GPS-Modul^[10], sowie für die Kommunikation mit dem Nextion Display^[9], frei im Internet zugängliche Bibliotheken verwendet.

12.1.1 Jeti Decoder

Hierbei handelt es sich um eine selbstgeschriebene Arduino-Bibliothek, mit welcher die über eine serielle Verbindung zur Verfügung gestellten Telemetrie-Daten der Motorregler ausgelesen werden können. Konkret wurde das „Simple-Text“-Protokoll aus dem Datenblatt der Jeti-Telemtry^[11] verwendet.

Jeti.hpp

```

1 /*
2 * Jeti-Library by Stefan Deimel, 23.10.2020
3 * Jeti_UNO.hpp -- platform dependent functions for Arduino Uno
4 */
5 #include <JetiModes.hpp>
6 #if JETI_MODE == ARDUINO_UNO
7
8 #ifndef __JETI_UNO_HPP__
9 #define __JETI_UNO_HPP__
10
11 #include <Arduino.h>
12 #include <inttypes.h>
13 #include <stdlib.h>
14 #include "JetiBase.hpp"
15
16 // pointer to jeti-base object, to preventent multiple objects Jeti_UNO from causing problems
17 JetiBase *jeti_pointer = nullptr;
18
19 class Jeti_UNO : public JetiBase
20 {
21 public:
22     Jeti_UNO() : JetiBase()
23     {
24         jeti_pointer = this;
25     }
26
27     // initialises serial communication (still need to call sei() in main setup)
28     void init()
29     {
30         // initialise serial communication -- 9600baud, 9bit, odd parity, 2 stop bits
31         UCSROA = 0;
32         UCSROB = 1 << RXCIE0 | 1 << RXENO | 1 << UCSZ02;

```

```

33     UCSROC = 3 << UPM00 | 1 << USBS0 | 3 << UCSZ00;
34     UBRROH = 0;
35     UBRROL = 103;
36     DDRD &= ~(1 << PD1); // set input
37     PORTD |= (1 << PD1); // set pullup
38 }
39
40 // overwrite interrupt functions
41 void interrupt_RX()
42 {
43     receiving = true;
44     recieve_buffer[recieve_buffer_write].bit9 = UCSROB & (1 << RXB80); // 9th bit, must be read
45     ↪ first
46     recieve_buffer[recieve_buffer_write].databyte = UDR0;
47     recieve_buffer_write++;
48     recieve_buffer_write &= (RECIEVE_BUFFER_SIZE - 1);
49 }
50 void interrupt_TX()
51 {
52     send_count--;
53     sendDone = (send_count > 0);
54 }
55 void interrupt_UDRE()
56 {
57     send();
58 }
59 private:
60     // function to initialise send
61 void send_init()
62 {
63     UCSROB &= ~(1 << RXENO); // disable receive
64     UCSROB &= ~(1 << RXCIE0); // disable receive interrupt
65     DDRD |= (1 << PD1); // set output
66     UCSROB |= 1 << TXENO; // enable transmit
67     UCSROB |= (1 << TXCIE0); // enable transmitem complete
68     if (UCSROA & (1 << UDRE0))
69     {
70         send();
71         if (send_count > 1)
72             goto enableInterrupt;
73     }
74     else
75     {
76         enableInterrupt:
77             UCSROB |= 1 << UDRIE0; // enable send interrupt
78     }
79 }
80
81 // actual send function
82 void send()
83 {
84     UCSROB &= ~(1 << TXB80);
85     UDR0 = ((!buttonL) & 1) << 7 | ((!buttonD) & 1) << 6 | ((!buttonU) & 1) << 5 | ((!buttonR) &
86     ↪ 1) << 4;
87 }
88
89 // reenable receive mode
90 void send_done()
91 {
92     UCSROB &= ~(1 << UDRIE0); // disable send interrupt
93     UCSROB &= ~(1 << TXCIE0); // disable transmit interrupt
94     UCSROB &= ~(1 << TXENO); // disable transmit

```

```

94     UCSROA |= (1 << TXCO);
95     DDRD &= ~(1 << PD1); // set input
96     PORTD |= (1 << PD1); // set pullup
97     while (!(PIND & (1 << PD1)))
98     ;
99     UCSROB |= (1 << RXENO); // enable receive
100    UCSROB |= (1 << RXCIEO); // disable receive interrupt
101    sendDone = false;
102 }
103 };
104
105 // interrupt vectors
106
107 ISR(USART_RX_vect)
108 {
109     if (jeti_pointer != nullptr)
110         jeti_pointer->interrupt_RX();
111 }
112
113 ISR(USART_UDRE_vect)
114 {
115     if (jeti_pointer != nullptr)
116         jeti_pointer->interrupt_UDRE();
117 }
118
119 ISR(USART_TX_vect)
120 {
121     if (jeti_pointer != nullptr)
122         jeti_pointer->interrupt_TX();
123 }
124
125 #endif /*__Jeti_ATmega_Serial1_HPP__*/
126
127 #endif

```

Jeti_ATmega_Serial1.hpp

```

1 /*
2 * Jeti-Library by Stefan Deimel, 23.10.2020
3 * Jeti_ATmega_Serial1.hpp -- platform dependent functions for Arduino Mega using Serial 1
4 */
5 #include <JetiModes.hpp>
6 #if JETI_MODE == ARDUINO_MEGA_1
7
8 #ifndef __Jeti_ATmega_Serial1_HPP__
9 #define __Jeti_ATmega_Serial1_HPP__
10
11 #include <Arduino.h>
12 #include <inttypes.h>
13 #include <stdlib.h>
14 #include "JetiBase.hpp"
15
16 // pointer to jeti-base object, to prevent multiple objects Jeti_UNO from causing problems
17 JetiBase *jeti_pointer = nullptr;
18
19 class Jeti_ATmega_Serial1 : public JetiBase
20 {
21 public:
22     Jeti_ATmega_Serial1() : JetiBase()
23     {
24         jeti_pointer = this;
25     }
26

```

```

27 // initialises Serial Communication (still need to call sei() in main setup)
28 void init()
29 {
30     // initialise serial communication -- 9600baud, 9bit, odd parity, 2 stop bits
31     UCSR1A = 0;
32     UCSR1B = 1 << RXCIE1 | 1 << RXEN1 | 1 << UCSZ12;
33     UCSR1C = 3 << UPM10 | 1 << USBS1 | 3 << UCSZ10;
34     UBRR1H = 0;
35     UBRR1L = 103;
36     DDRD &= ~(1 << PD3); // set input
37     PORTD |= (1 << PD3); // set pullup
38 }
39
40 // overwrite interrupt functions
41 void interrupt_RX()
42 {
43     receiving = true;
44     receive_buffer[recieve_buffer_write].bit9 = UCSR1B & (1 << RXB81); // 9th bit, must be read
45     ↪ first
46     receive_buffer[recieve_buffer_write].databyte = UDR1;
47     recieve_buffer_write++;
48     recieve_buffer_write &= (RECIEVE_BUFFER_SIZE - 1);
49 }
50
51 void interrupt_TX()
52 {
53     send_count--;
54     sendDone = (send_count > 0) ? 0 : 1;
55 }
56
57 void interrupt_UDRE()
58 {
59     send();
60 }
61 private:
62     // function to initialise send
63     void send_init()
64     {
65         UCSR1B &= ~(1 << RXEN1); // disable receive
66         UCSR1B &= ~(1 << RXCIE1); // disable receive interrupt
67         DDRD |= (1 << PD3); // set output
68         UCSR1B |= 1 << TXEN1; // enable transmit
69         UCSR1B |= (1 << TXCIE1); // enable transmitem complete
70         //delay_ms(1);
71         if (UCSR1A & (1 << UDRE1))
72         {
73             send();
74             if (send_count > 1)
75                 goto enableInterrupt;
76         }
77         else
78         {
79             enableInterrupt:
80                 UCSR1B |= 1 << UDRIE1; // enable send interrupt
81             }
82         }
83
84     // actual send function
85     void send()
86     {
87         UCSR1B &= ~(1 << TXB81);

```

```

88     UDR1 = ((!buttonL) & 1) << 7 | ((!buttonD) & 1) << 6 | ((!buttonU) & 1) << 5 | ((!buttonR) &
89     ↪ 1) << 4;
90 }
91 // reenable receive mode
92 void send_done()
93 {
94     UCSR1B &= ~(1 << UDRIE1); // disable send interrupt
95     UCSR1B &= ~(1 << TXCIE1); // disable transmit interrupt
96     UCSR1B &= ~(1 << TXEN1); // disable transmit
97     UCSR1A |= (1 << TXC1);
98     DDRD &= ~(1 << PD3); // set input
99     PORTD |= (1 << PD3); // set pullup
100    while (!(PIN0 & (1 << PD3)))
101        ;
102    //_delay_ms(1);
103    UCSR1B |= (1 << RXEN1); // enable receive
104    UCSR1B |= (1 << RXCIE1); // disable receive interrupt
105    sendDone = false;
106 }
107 };
108
109 // interrupt vectors
110
111 ISR(USART1_RX_vect)
112 {
113     if (jeti_pointer != nullptr)
114         jeti_pointer->interrupt_RX();
115 }
116
117 ISR(USART1_UDRE_vect)
118 {
119     if (jeti_pointer != nullptr)
120         jeti_pointer->interrupt_UDRE();
121 }
122
123 ISR(USART1_TX_vect)
124 {
125     if (jeti_pointer != nullptr)
126         jeti_pointer->interrupt_TX();
127 }
128
129 #endif /*__Jeti_ATmega_Serial1_HPP__*/
130
131 #endif

```

Jeti_UNO.hpp

```

1 /*
2 * Jeti-Library by Stefan Deimel, 23.10.2020
3 * Jeti_UNO.hpp -- platform dependent functions for Arduino Uno
4 */
5 #include <JetiModes.hpp>
6 #if JETI_MODE == ARDUINO_UNO
7
8 #ifndef __JETI_UNO_HPP__
9 #define __JETI_UNO_HPP__
10
11 #include <Arduino.h>
12 #include <inttypes.h>
13 #include <stdlib.h>
14 #include "JetiBase.hpp"
15

```

```
16 // pointer to jeti-base object, to prevent multiple objects Jeti_UNO from causing problems
17 JetiBase *jeti_pointer = nullptr;
18
19 class Jeti_UNO : public JetiBase
20 {
21 public:
22     Jeti_UNO() : JetiBase()
23     {
24         jeti_pointer = this;
25     }
26
27     // initialises serial communication (still need to call sei() in main setup)
28     void init()
29     {
30         // initialise serial communication -- 9600baud, 9bit, odd parity, 2 stop bits
31         UCSROA = 0;
32         UCSROB = 1 << RXCIE0 | 1 << RXENO | 1 << UCSZ02;
33         UCSROC = 3 << UPM00 | 1 << USBS0 | 3 << UCSZ00;
34         UBRROH = 0;
35         UBRROL = 103;
36         DDRD &= ~(1 << PD1); // set input
37         PORTD |= (1 << PD1); // set pullup
38     }
39
40     // overwrite interrupt functions
41     void interrupt_RX()
42     {
43         receiving = true;
44         receive_buffer[receive_buffer_write].bit9 = UCSROB & (1 << RXB80); // 9th bit, must be read
45         ↪ first
46         receive_buffer[receive_buffer_write].databyte = UDR0;
47         receive_buffer_write++;
48         receive_buffer_write &= (RECEIVE_BUFFER_SIZE - 1);
49     }
50     void interrupt_TX()
51     {
52         send_count--;
53         sendDone = (send_count > 0);
54     }
55     void interrupt_UDRE()
56     {
57         send();
58     }
59 private:
60     // function to initialise send
61     void send_init()
62     {
63         UCSROB &= ~(1 << RXENO); // disable receive
64         UCSROB &= ~(1 << RXCIE0); // disable receive interrupt
65         DDRD |= (1 << PD1); // set output
66         UCSROB |= 1 << TXENO; // enable transmit
67         UCSROB |= (1 << TXCIE0); // enable transmited complete
68         if (UCSROA & (1 << UDRE0))
69         {
70             send();
71             if (send_count > 1)
72                 goto enableInterrupt;
73         }
74     else
75     {
76         enableInterrupt:
77             UCSROB |= 1 << UDRIE0; // enable send interrupt
```

```

78     }
79 }
80
81 // actual send function
82 void send()
83 {
84     UCSROB &= ~(1 << TXB80);
85     UDR0 = ((!buttonL) & 1) << 7 | ((!buttonD) & 1) << 6 | ((!buttonU) & 1) << 5 | ((!buttonR) &
86     ↪ 1) << 4;
87 }
88
89 // reenable receive mode
90 void send_done()
91 {
92     UCSROB &= ~(1 << UDRIEO); // disable send interrupt
93     UCSROB &= ~(1 << TXCIEO); // disable transmit interrupt
94     UCSROB &= ~(1 << TXENO); // disable transmit
95     UCSROA |= (1 << TXCO);
96     DDRD &= ~(1 << PD1); // set input
97     PORTD |= (1 << PD1); // set pullup
98     while (!(PIND & (1 << PD1)))
99     ;
100    UCSROB |= (1 << RXENO); // enable receive
101    UCSROB |= (1 << RXCIEO); // disable receive interrupt
102    sendDone = false;
103 }
104
105 // interrupt vectors
106
107 ISR(USART_RX_vect)
108 {
109     if (jeti_pointer != nullptr)
110         jeti_pointer->interrupt_RX();
111 }
112
113 ISR(USART_UDRE_vect)
114 {
115     if (jeti_pointer != nullptr)
116         jeti_pointer->interrupt_UDRE();
117 }
118
119 ISR(USART_TX_vect)
120 {
121     if (jeti_pointer != nullptr)
122         jeti_pointer->interrupt_TX();
123 }
124
125 #endif /*_Jeti_ATmega_Serial1_HPP_*/
126
127 #endif

```

JetiBase.hpp

```

1 /*
2 * Jeti-Library by Stefan Deimel, 23.10.2020
3 * JetiBase.hpp -- all platform independant functions
4 */
5 #ifndef __JetiBase_HPP__
6 #define __JetiBase_HPP__
7
8 #include <Arduino.h>
9

```

```
10 #define RECIEVE_BUFFER_SIZE 16 // size of serial recieve buffer
11
12 // struct for data/telemtrie from jetibox
13 typedef struct
14 {
15     char mode;
16     int percent;
17     long rpm;
18     double voltage;
19     int temperature;
20 } jetiTelemetry_t;
21
22 // struct for 9 bit Serial
23 typedef struct
24 {
25     char databyte; // data byte (first 8 bits)
26     bool bit9;    // 9. bit of data
27 } Serial9bit_t;
28
29 class JetiBase
30 {
31 public:
32     JetiBase();           // Constructor
33     virtual void init() = 0; // inits Serial Communication (still need to call sei() in main setup)
34     void loop();          // loop function, must be called periodically
35
36     void send(bool l, bool r, bool u, bool d);           // sends buttons
37     void send(bool l, bool r, bool u, bool d, int count); // sends buttons multiple times
38
39     bool isNewMsg(); // checks if new msg has been recieved
40     String getMsg(); // get message (this function deletes message afterwards)
41
42     // convert message (String) to jetiTelemetry_t struct
43     static jetiTelemetry_t getTelemetry(String msg);
44
45     // interrupt functions (internal functions, must not be called in main, are overwritten in each
        ↪ platform-dependent file)
46     virtual void interrupt_RX() = 0;
47     virtual void interrupt_TX() = 0;
48     virtual void interrupt_UDRE() = 0;
49
50 protected:
51     virtual void send_init() = 0; // initialize send and start sending first msg, enables send
        ↪ interrupts if send_count is higher then one
52     virtual void send() = 0;      // send msg (gets called in interrupt and send_init())
53     virtual void send_done() = 0; // disable send interrupts and reenable recieve mode
54
55     volatile Serial9bit_t recieve_buffer[RECIEVE_BUFFER_SIZE]; // recieve buffer
56     volatile size_t recieve_buffer_write;                      // points where to write into recieve
        ↪ buffer
57     volatile size_t recieve_buffer_read;                     // points where to read from recieve
        ↪ buffer
58
59     volatile bool receiving; // currently receiving message (start has been send, but message is not
        ↪ finished yet, important to avoid sending while receiving)
60     volatile bool sendDone; // wether sending has finished
61
62     // buttons status to send (gets send in send function)
63     volatile bool buttonL; // left button
64     volatile bool buttonR; // right button
65     volatile bool buttonD; // down button
66     volatile bool buttonU; // up button
67     volatile int send_count; // how often message has to bee send
```

```
68     String receiving_msg; // storage for messages which are currently received (second receive buffer
69     ↪ )
70
71     String newMsgString; // fully received message, gets overwritten by next completed msg
72 };
73
74 #endif /*_JetiBase_HPP__*/
```

JetiBase.cpp

```
1 /*
2 * Jeti-Library by Stefan Deimel, 23.10.2020
3 * JetiBase.cpp -- all platform independant functions
4 */
5 #include "JetiBase.hpp"
6
7 JetiBase::JetiBase()
8 {
9     // initialize variables
10    recieve_buffer_write = 0;
11    recieve_buffer_read = 0;
12
13    receiving = false;
14    sendDone = false;
15
16    buttonL = false;
17    buttonR = false;
18    buttonD = false;
19    buttonU = false;
20    send_count = 0;
21
22    receiving_msg = "";
23    newMsgString = "";
24 }
25
26 void JetiBase::loop()
27 {
28     // receiving data
29     for (; recieve_buffer_read != recieve_buffer_write; recieve_buffer_read = (recieve_buffer_read +
30     ↪ 1) & (RECEIVE_BUFFER_SIZE - 1))
31     {
32         if (!recieve_buffer[recieve_buffer_read].bit9 && receiving_msg != "")
33         {
34             // end of message
35             newMsgString = receiving_msg;
36             receiving_msg = "";
37             receiving = false;
38         }
39         else
40         {
41             // add values from buffer to String
42             receiving_msg += (char)recieve_buffer[recieve_buffer_read].databyte;
43             receiving = true;
44         }
45
46     // sending data
47     if (send_count && !receiving)
48     {
49         send_init();
50         _delay_ms(4); // delay for stability reasons
51     }
52 }
```

```

52     if (sendDone)
53     {
54         send_done();
55         sendDone = false;
56     }
57 }
58
59 void JetiBase::send(bool l, bool r, bool u, bool d)
60 {
61     send(l, r, u, d, 1);
62 }
63
64 void JetiBase::send(bool l, bool r, bool u, bool d, int count)
65 {
66     buttonL = l;
67     buttonR = r;
68     buttonU = u;
69     buttonD = d;
70     send_count = ((count > 0) ? (count) : 1); // send_count must be 1 ore higher
71 }
72
73 bool JetiBase::isNewMsg()
74 {
75     return (newMsgString != "");
76 }
77
78 String JetiBase::getMsg()
79 {
80     String ret = newMsgString;
81     newMsgString = "";
82     return ret;
83 }
84
85 jetiTelemetry_t JetiBase::getTelemetry(String msg)
86 {
87     String voltage = msg.substring(17, 22);
88     voltage.replace(",","");
89     return {.mode = (char)msg.charAt(1), .percent = (int)msg.substring(2, 5).toInt(), .rpm = msg.
90         substring(10, 14).toInt(), .voltage = voltage.toDouble(), .temperature = (int)msg.substring
91         (29, 31).toInt()};
}

```

JetiModes.hpp

```

1 /*
2 * Jeti-Library by Stefan Deimel, 23.10.2020
3 * JetiModes.hpp -- all platform definitions
4 */
5 #ifndef __JETIMODES_HPP__
6 #define __JETIMODES_HPP__
7
8 // it is necessary to define JETI-MODE in precompiler because otherwise interrupt vectors could be
9 // non existend or defined multiple times
10 #define ARDUINO_UNO 1    // arduino uno
11 #define ARDUINO_MEGA_1 2 //arduino mega serial 1
12
13 #endif /* __JETIMODES_HPP__ */

```

12.2 Datenblätter

12.2.1 JetiBox

Link zur Datei^[4]



DE

Jeti Produkte, welche mit der JetiBox kommunizieren, werden mit dem Logo "JetiBox compatible" gekennzeichnet.

Anwendung JETIBOX:

1. Messung der Impulslänge der Empfänger-Kanalausgänge
2. Impulsgenerator für Servos
3. Servocycler
4. Messung der Servogeschwindigkeit
5. Kommunikation mit den Drehzahlstellern **SPIN**
(siehe Bedienungsanleitung zu den DS SPIN)
6. Kommunikation mit den sensor Drehzahlstellern für **BLDC**
7. Kommunikation mit den Empfänger **REX JBC**
8. Kommunikation mit den **DUPLEX-System**

Für die Applikation **Nr. 1** braucht man den Empfänger mit Sender und Empfängerakkus (4,8 – 8,4V). Die Akkus werden an die **graue Buchse**, der Empfänger an die **blaue Buchse** an der rechten Seite der **JETI BOX** angeschlossen.

Zu den Applikationen **Nr. 2**, **Nr. 3** und **Nr. 4** benötigen wir den Empfänger-Akku (4,8 – 8,4V) und ein Servo. Die Akkus werden in die **graue Buchse**, das Servo in die **blaue Buchse** eingesteckt.

Im Falle einer Änderung der Applikation müssen die Akkus von der **JETI BOX** getrennt werden und dann wieder angeschlossen werden. Die gewünschte Applikation wählen Sie mit den Tasten **R** und **L**.

Wenn Sie keine Empfängerbatterien oder eine andere Stromquelle (im Bereich von 4,8 – 8,4V) zur Verfügung haben, kann die **JETI BOX** vom BEC des Drehzahlstellers versorgt werden. Den JR-Stecker des DS stecken Sie in die Buchse B (Impuls ist das orangene Kabel, nicht näher bezeichnete Position). Schließen Sie die Flugakkus an und schalten Sie den Schalter ein (gilt nicht für SPIN 11).

DE

1. Messen der Impulslänge der Empfänger-Kanalausgänge

Mit Hilfe dieser Applikation kann die Impulslänge jedes beliebigen Kanal-Ausgangsimpulses des Empfängers gemessen werden. Weiterhin kann die Versorgungsspannung des Empfängerakkus gemessen werden. Schließen Sie den Empfängerakku am Empfänger an. Mit Hilfe des Verbindungskabels, welches im Lieferumfang der **JETI BOX** ist, verbinden Sie die **blaue Buchse** mit dem gewünschten Empfänger Kanalausgang. Schalten Sie den Sender und Empfänger ein. In der Anzeige erscheint **IMPULS DETECTION**, wo sie die Impulslänge in ms und die Spannung der Empfängerakkus ablesen können.

2. Impulsgenerator für Servos

Diese Applikation der JETI Box ermöglicht die Generierung von Servo-Steuerimpulsen und gleichzeitig die Messung der Servo-Versorgungsspannung. Schließen Sie die Akkus und das Servo an und wählen Sie mit Hilfe der Tasten L und P die Funktion IMPULSGENERATOR.

Mit den Tasten können Sie den Bereich zwischen 1,024 ms bis 2,047 ms ändern, und dies in Schritten von tausendsteln oder hundertsteln ms. Diese Funktion eignet sich z. B. zum Einstellen der Mittellage des Servos (1,500 ms) ohne Verwendung des Empfängers und Senders. Schließen Sie den Akku und das Servo an.

Die Impulslänge kann mit Hilfe aller vier Tasten eingestellt werden.

Mit der Taste **L** wird der Impuls in Schritten von 0,001 ms verkürzt

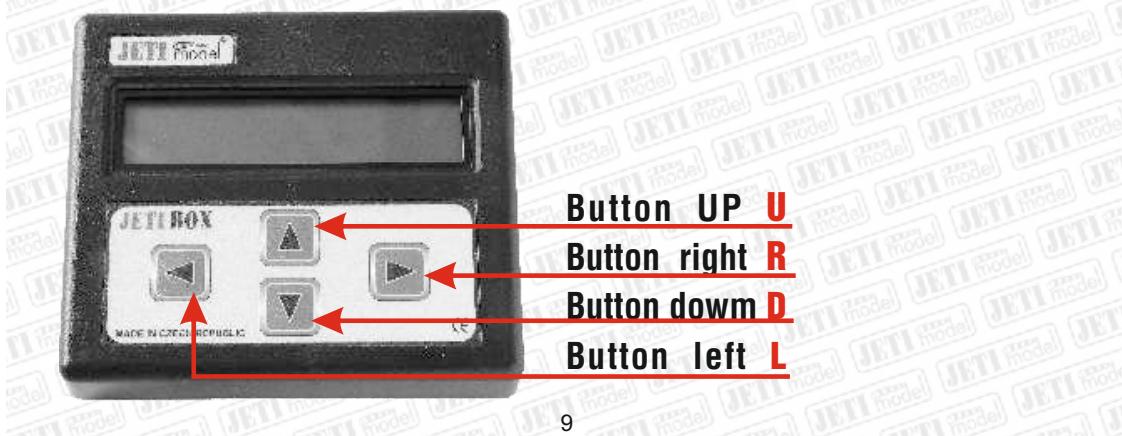
Mit der Taste **D** wird der Impuls in Schritten von 0,01 ms verkürzt

Mit der Taste **U** wird der Impuls in Schritten von 0,01 ms verlängert

Mit der Taste **R** wird der Impuls in Schritten von 0,001 ms verlängert

3. Servocycler

Bei dieser Applikation kann die Anzahl der Zyklen, der Servoweg und die Zyklusgeschwindigkeit eingestellt werden. Man kann damit die Servolebensdauer testen, die Servos Einbrennen und ihre Funktion testen.



DE

Schließen Sie die Akkus und das Servo an und wählen Sie mit Hilfe der Tasten **L** und **R** die Funktion **SERVO CYCLE**. Mit den Tasten **U** und **D** wählen Sie die Anzahl der Zyklen im Bereich von 10 bis 990 (nur in Schritten von 10 Zyklen).

Mit den Tasten **L** und **R** kann die Geschwindigkeit von 1 bis 99 eingestellt werden. Die Geschwindigkeit $v=1$ bedeutet, dass jeder nachfolgende Impuls im Vergleich zu vorangehenden Impuls sich um 0,001 ms verändert bis der eingestellte Wert erreicht wird (Analog $v=20$ bedeutet eine Änderung um 0,020 ms). Die Impulsperiode ist 20 ms. Mit den Tasten **U** und **D** kann ein Wert eingestellt werden, der die Größe des Servoweges in μs vom Mittelwert 1,5 ms im Bereich von 100 bis 500 μs angibt.

Bei eingestelltem $\alpha=500 \mu s$ wird sich der Steuerimpuls für Servos von $1,000 \div 2,000$ ms (d. h. $1,500$ ms $\pm 500 \mu s$) verändern. Der Wert hinter **#** gibt die Anzahl der Zyklen an, die bis zum Testende verbleiben.

Nach Beendigung des Tests kehrt das Programm zurück zum Anfang **SERVO CYCLE**.

4. Messen der Servogeschwindigkeit

Durch Anwendung dieser Messung können wir die Zeit feststellen, in welcher das Servo von einer definierten Position in eine andere übergeht. Die Messung kann entweder ohne Belastung oder am installierten Servo direkt im Modell bei konkreten Hebelverhältnissen durchgeführt werden. Die Impulslänge der ersten Endlage des Servos kann im Bereich von 1,024 bis 1,400 ms und der zweiten Endlage im Bereich von 1,600 ms bis 2.047 ms eingestellt werden. Wenn wir z. B. die Geschwindigkeit bei einer Drehung der Austrittsachse des Servos um 60° messen wollen, muss diese Einstellung z. B. mit Hilfe eines Winkelmessers durchgeführt werden.

Schließen Sie den Akku und das Servo an, stellen Sie mit den Tasten **L** und **R** die Funktion **SERVO SPEED** ein. Mit den Tasten **U** und **D** stellen wir die erste Endlage des Servos ein. Dann gehen wir mit Hilfe der Taste **R** zur Einstellung der zweiten Endlage über, die auch mit den Tasten **U** und **D** eingestellt wird.

Die Messung wird gestartet. In der Anzeige zeigt sich die resultierende Zeit des Servos, die zur Fahrt von einer eingestellten Position zur zweiten in Sekunden gebraucht wird. Diese Messung können Sie beliebig wiederholen, ggf. andere Endlagen einstellen.

Wir wünschen Ihnen mit unseren Produkten viele angenehme Stunden.

JETI model s.r.o, Lomena 1530, 742 58 Pribor,
www.jetimodel.com, e-mail: jeti@jetimodel.cz

12.2.2 MasterSPIN 220 Pro OPTO

Link zur Datei^[5]



1. Einleitung

Die Controller der Reihe **SPIN pro** sind Nachfolger der für bürstenlose Motoren bestimmten Drehzahlregler der Serie SPIN. Die Reihe mit dem Attribut PRO bietet entgegen den älteren Verwandten erweiterte Einstellmöglichkeiten, wie z. B. ein automatisches Motor-Timing, eine direkte Einstellung der Abschaltspannung in Volt, ein Tuning des Anlaufverhaltens bei verschiedenen Motortypen und auch die Möglichkeit einer speziellen Modusaktivierung für die Einstellung der Autorotation bei Hubschraubern. Das Leistungsspektrum der Controller **SPIN pro** deckt den gleichen Bereich, wie die Vorgänger der Reihe SPIN, d. h. von 11A bis 300A ab.

1.1 Drehzahlsteller SPINpro

Alle Drehzahlsteller (**mit Ausnahme der OPTO-Typen**) sind mit einem getakteten BEC zur Versorgung des Empfängers und der Servos ausgestattet. Diese Art der Spannungsversorgung ist auch bei Antriebsakkus mit hoher Zellenanzahl einsetzbar, weiters hängt die Anzahl der versorgbaren Servos nicht von der Eingangsspannung des Drehzahlstellers ab.



DE

Technische Daten der SPINpro Drehzahlsteller mit BEC:

Typ	Dauerstrom [A]	Spannung [V]	BEC [A]	BEC [V]	Abmessungen* ¹⁾ [mm]	Gewicht* ²⁾ [g]
SPIN 11 pro	11	5 - 17	2,5	5,5	32x23x6	12
SPIN 22 pro	22	5 - 17	2,5	5,5	32x23x7	26
SPIN 33 pro	33	5 - 21	3	5,5	42x23x7	32
SPIN 44 pro	44	5 - 26	5	5,5	52x25x10	44
SPIN 55 pro	55	5 - 34	5	5,5	52x25x12	60
SPIN 66 pro	70	5 - 26	5	5,5	52x25x12	56

1.2 Drehzahlsteller SPINpro OPTO:

Diese Drehzahlsteller haben einen galvanisch getrennten Ein- und Ausgang, deshalb ist eine unabhängige Stromversorgung für Empfänger und Servos erforderlich (4-5 NiXX oder 2-3 LiXX in Verbindung mit einem Spannungsregler, z.B. Jeti MAX BEC)

Drehzahlsteller **SPIN OPTO** sind mit zwei Anschlußkabel System JR ausgerüstet. Der schwarze Stecker am längeren dreipoligen Kabel wird am Empfänger eingesteckt. Der rote Stecker am kürzeren dreipoligen Kabel ist für die Kommunikation mit der Jeti Box; zum Programmieren oder Auslesen der Daten schließen Sie den roten Stecker an die mit **Impuls +/-** beschriftete Buchse der **JetiBox**.

Technische Daten der SPINpro OPTO Drehzahlsteller:

Typ	Dauer- strom [A]	Span- nung [V]	BEC [A]	BEC [V]	Abmessungen* ¹⁾ [mm]	Gewicht* ²⁾ [g]
SPIN 66 pro opto	70	6 - 26	-	-	52x25x12	45
SPIN 75 pro opto	75	12 - 42	-	-	52x25x15	55
SPIN 77 pro opto	77	12 - 50	-	-	65x55x17	110
SPIN 99 pro opto	90	12 - 50	-	-	65x55x17	110
SPIN 125 pro opto	125	12 - 50	-	-	65x55x25	120
SPIN 200 pro opto	170	18 - 59	-	-	63x120x27	326
SPIN 300 pro opto	220	18 - 59	-	-	63x120x27	360



DE

2. Drehzahlsteller anschließen

2.1 Allgemeine Anschlußbedingungen für den Drehzahlsteller:

- benutzen Sie grundsätzlich nur überprüfte und neue Anschlußstecker, die sorgfältig an die Kabel angelötet sein müssen
- für Drehzahlsteller des Typs SPIN 11 und SPIN 22 empfehlen wir Stecker G2, für höhere Leistungsdurchsätze die Typen G3,5 oder G4. Nach dem Anlöten der Stecker kontrollieren Sie bitte, ob das federnde Vorderteil des Steckers drehbar geblieben ist. Es kann vorkommen, dass das Flussmittel entlang der Steckeroberfläche infolge von Kapillarität emporsteigt und im Extremfall den federnden Teil vom Steckerkörper galvanisch trennt. Als Gegenmaßnahme dient das Auswaschen des Steckers mit Hilfe eines Pinsels in Nitroverdünnung. Während des Betriebs achten Sie auf Sauberkeit des Steckers und auf seine Haltekraft. Falls diese nachlässt, wechseln Sie die Stecker sofort aus. **Wir empfehlen das Auswechseln aller Stecker nach 1- bis 2-jährigem Flugbetrieb.**



perfekte Lötstelle



unbrauchbare ("kalte") Lötstelle

- die Entfernung zwischen dem Motor und dem Drehzahlsteller sollte 10 – 15 cm nicht übersteigen. Eine weitere Verlängerung der Kabel zu den Antriebsakkus wird dadurch möglich, wenn parallel zu den Leitern Elektrolytkondensatoren geschaltet werden (mit niedrigem Innenwiderstand, nur Low ESR Typen, mit entsprechender Spannungsfestigkeit und einer Kapazität rund 300 Mikrofarad), und zwar je ein Kondensator pro 25 cm Kabellänge.
- stecken Sie den JR – Stecker in die Empfängerbuchse, die dem Motordrossel-Kanal entspricht.
- Den JR-Stecker (rotes Gehäuse) stecken Sie in die EXT-Buchse des DUPLEX-EX Empfängers oder ggf. in den DUPLEX-EX Expander.



DE

2.2 Anschluß eines SPINpro 125 opto, SPINpro 200 opto oder SPINpro 300

Drehzahlstellers:

Diese Reglertypen sind zusätzlich mit einem zusätzlichen Anschluss zur Blitzvermeidung beim Ansteckvorgang ausgerüstet.

Anschlussreihenfolge:

- 1.) Verbinden Sie Regler-Minus mit dem Minuspol des Akkus
- 2.) Verbinden Sie den roten (1.5-2mm dünnen) Regleranschluß mit dem PLUS-Pol des Akkus
- 3.) Plus-Pol des Reglers mit Akku-Plus verbinden
- 4.) Trennen Sie die dünne Plusleitung wieder zwischen Regler und Akku

Behandeln Sie das Modell nach dem Anschließen des Antriebsakkus mit extremer Vorsicht - vermeiden Sie unbedingt und jederzeit, dass sich jemand im Gefahrenbereich der Luftschaube aufhält!!!

3. Online Telemetrie (gilt nicht für Regler SPIN PRO 11,22 und 33)

Der Drehzahlsteller **SPIN PRO** ermöglicht mit Hilfe der **JetiBox** die Darstellung des Reglerzustands in Realzeit. Der JR-Stecker mit rotem Gehäuse kann direkt an die JetiBox oder an den **Eingang EXT. eines DUPLEX 2.4 GHz Empfängers** angeschlossen werden. Die Einstellung der Reglerparameter mit Hilfe der JetiBox kann nur dann durchgeführt werden, wenn der JR-Stecker mit schwarzem Gehäuse (Motordrossel) vom Empfänger abgezogen wird (oder wenn der Empfänger abgeschaltet ist). Falls der Regler nach Einschaltung der Versorgungsspannung am Eingang des JR-Steckers mit schwarzem Gehäuse einen Servoimpuls detektiert, geht er automatisch in den Abbildungsmodus von Telemetriedaten über. In diesem Modus reagiert der Regler nicht auf die Bedienung der JetiBox-Tasten.

Auf dem LCD-Display der JetiBox sind folgende Daten dargestellt:

Aktuelle Leistung in Prozent:

- R 80% - der Motor läuft, der Prozentwert gibt die Spannung am Motor an
B 100% - der Motor bremst, der Prozentwert gibt die Bremswirkung an
B 0% - Motor-Stop, ohne Bremse



DE

Aktuelle Motordrehzahl:

Diese wird in der rechten oberen Displayecke angezeigt. Der Wert ist entsprechend der Reglereinstellung umgerechnet (Motorpol-Anzahl und Untersetzung des Getriebes) und wird in Umdrehungen pro Minute angezeigt.

Aktuelle Spannung:

in der linken unteren Ecke wird die aktuelle Spannung des Antriebsakkus angezeigt.

Aktuelle Temperatur:

in der rechten unteren Ecke wird die aktuelle Temperatur des Reglers angezeigt

Fehlerzustände:

in der Mitte der unteren Zeile werden Fehlerzustände angezeigt, falls sie aufgetreten sind.

U - der Regler hat einen Wert der Versorgungsspannung gemessen, der unterhalb der eingestellten Abschaltspannung liegt

C - es ist ein Kommutierungsfehler aufgetreten, der Regler hat einen Auswertefehler beim Messen der Motorposition verzeichnet

T - der Temperaturschutz des Reglers wurde aktiviert, es wurde die eingestellte Maximaltemperatur überschritten

4. Einstellung mit Hilfe der Fernsteueranlage

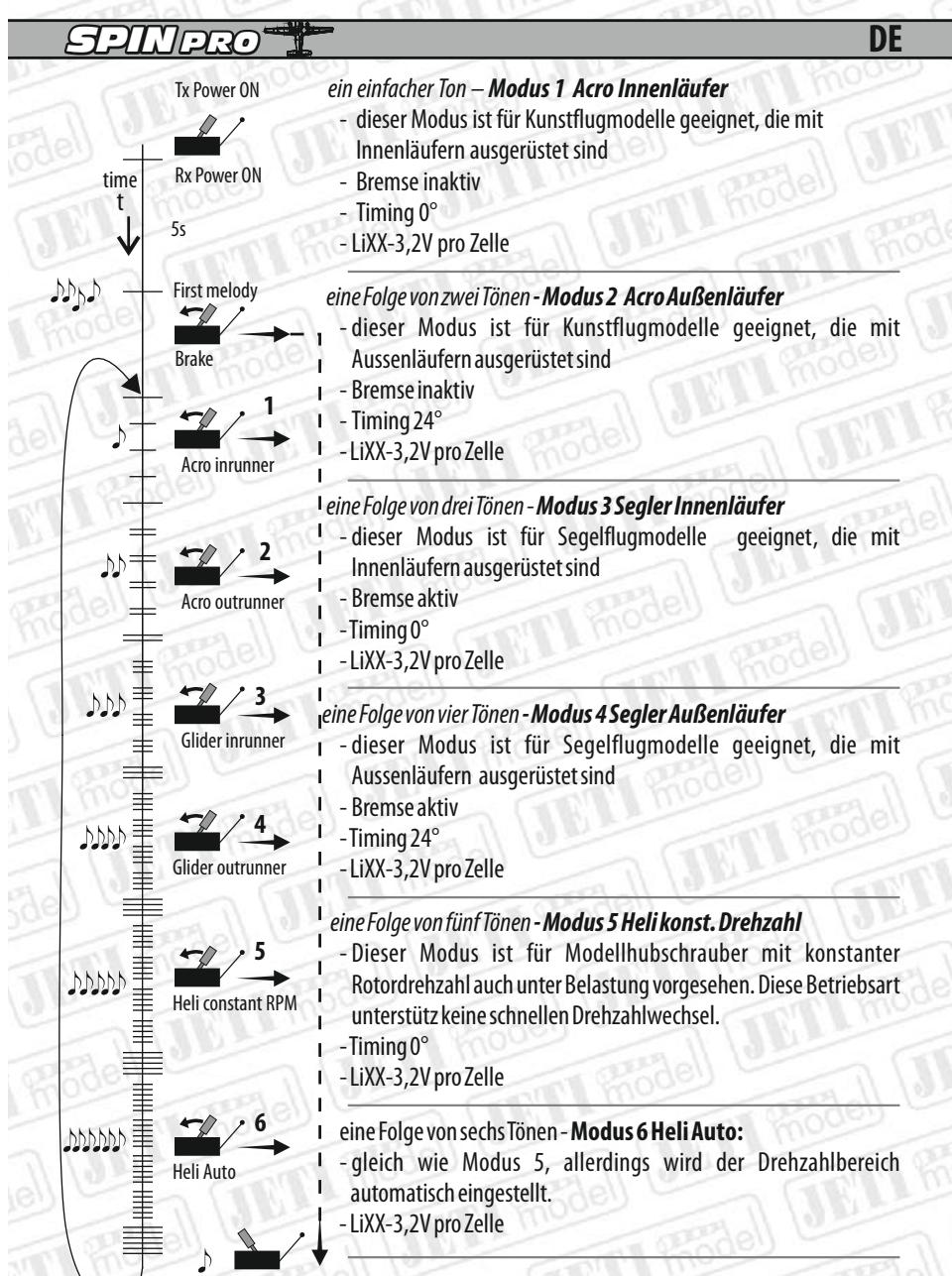
Voraussetzung: Im Menü "MAN Setting" muss die Einstellung des Parameters "Setting thru R/C" auf "ON" stehen (Werkseinstellung).

Programmievorgang:

1. verbinden Sie den SPINpro mit dem Empfänger, indem Sie den JR-Stecker in den Motordrosselkanal einstecken und schließen Sie den Motor an.

2. schieben Sie den Steuerknüppel in „Vollgasstellung“, schalten Sie den Sender ein und schließen den Flugakku an.

3. schalten Sie den Schalter ein (gilt nicht für SPIN 11), nach fünf Sekunden ertönen vier Töne. Wenn der Steuerknüppel sofort in Position Motor AUS geschoben wird, wird der Vollgas - Ausschlag gespeichert (END POINT), andernfalls folgt eine Fünfergruppe von sich wiederholenden Tönen, die den einzelnen Modi entsprechen:



Die gewählte Einstellung wird bestätigt, in dem der Drosselknüppel während der Ausgabe der Tonfolge des gewünschten Modus in die Leerlaufstellung gebracht wird.



DE

5. Einstellung mit Hilfe der JETIBOX

Die Einstellung erfolgt unter Zuhilfenahme von vier Tasten:

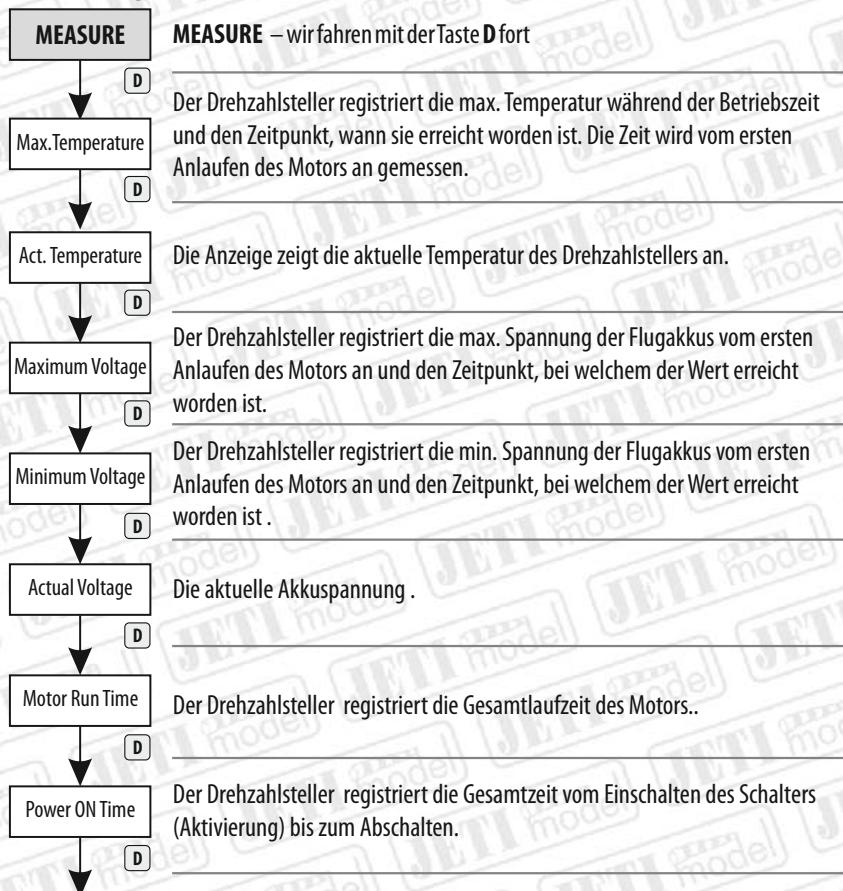
nach links **L** , nach rechts **R** , nach oben **U** , nach unten **D** . Stecken Sie den JR-Stecker des Drehzahlsteller in die Buchse mit der Bezeichnung **Impuls +** - an der rechten Seite der **JETIBOX**.

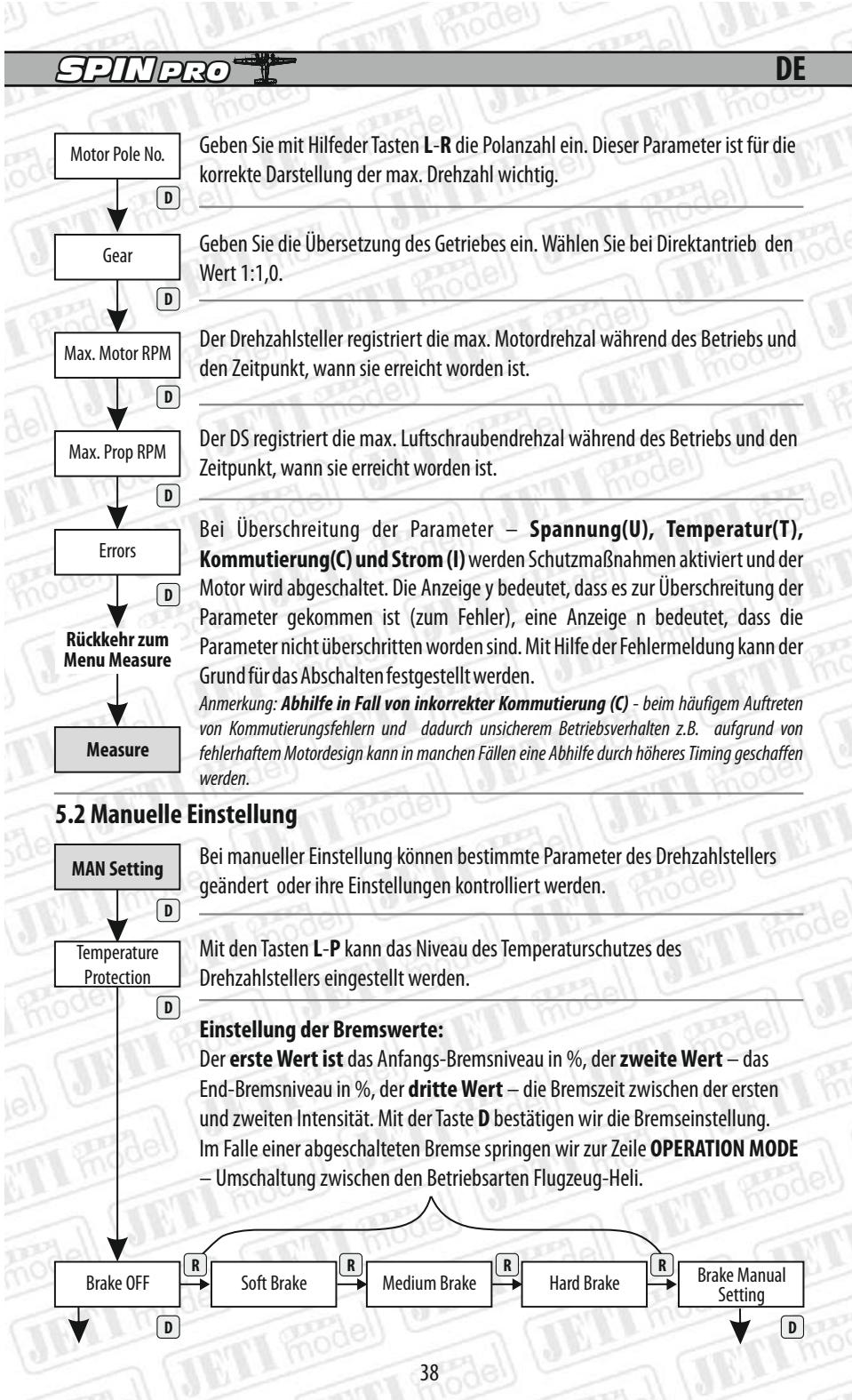
Bevor Sie den Flugakku anschließen, nehmen Sie aus Sicherheitsgründen die Luftschraube des Modells ab. An die Buchse mit der Bezeichnung **+-** schließen Sie nichts an.

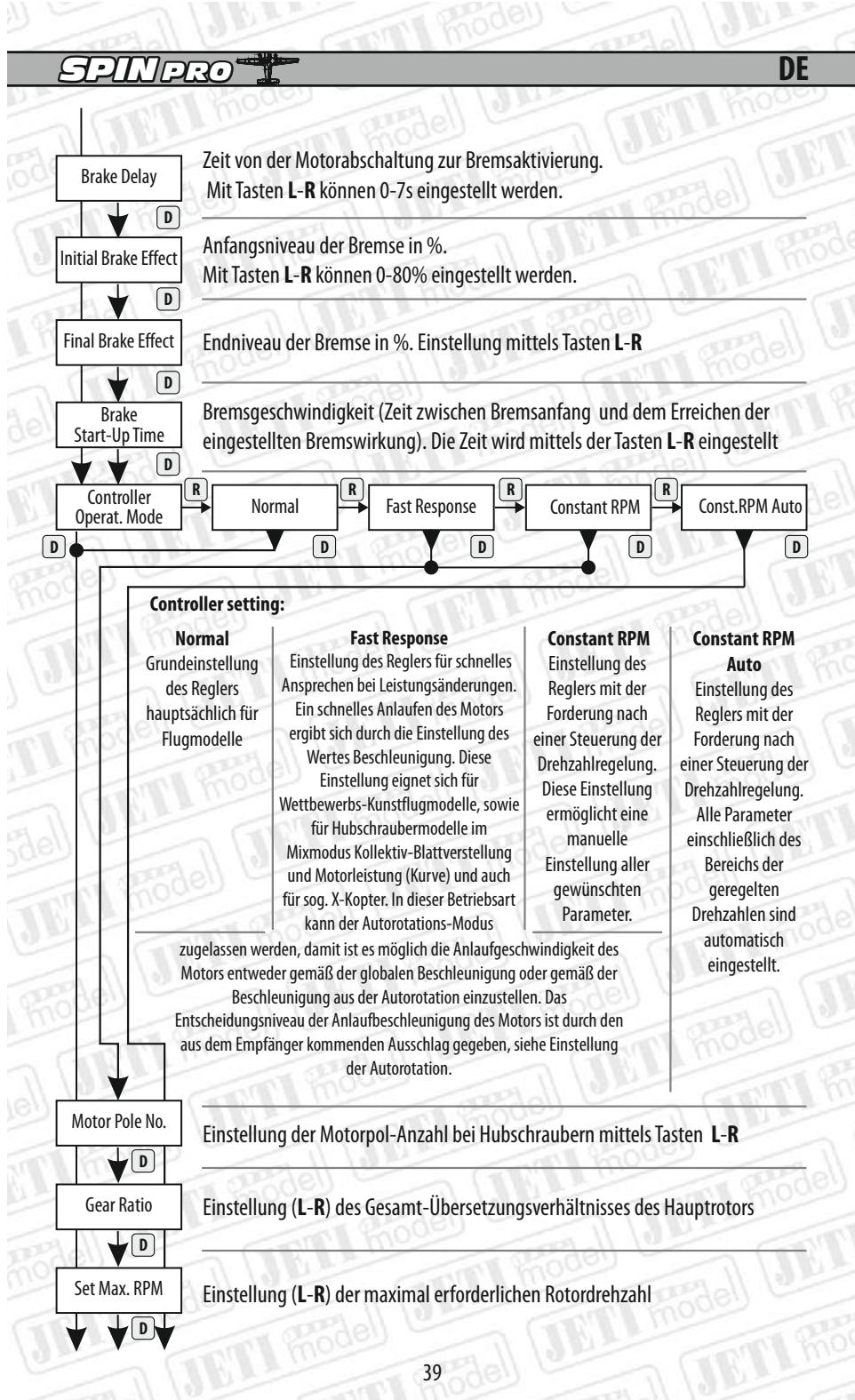
Schließen Sie die Flugakkus an und schalten den Schalter ein (gilt nicht für Spin11). In der Anzeige erscheint die Bezeichnung des angeschlossenen Drehzahlstellers. Mit Hilfe der Tasten **L** und **P** erhalten Sie weitere detaillierte Informationen.

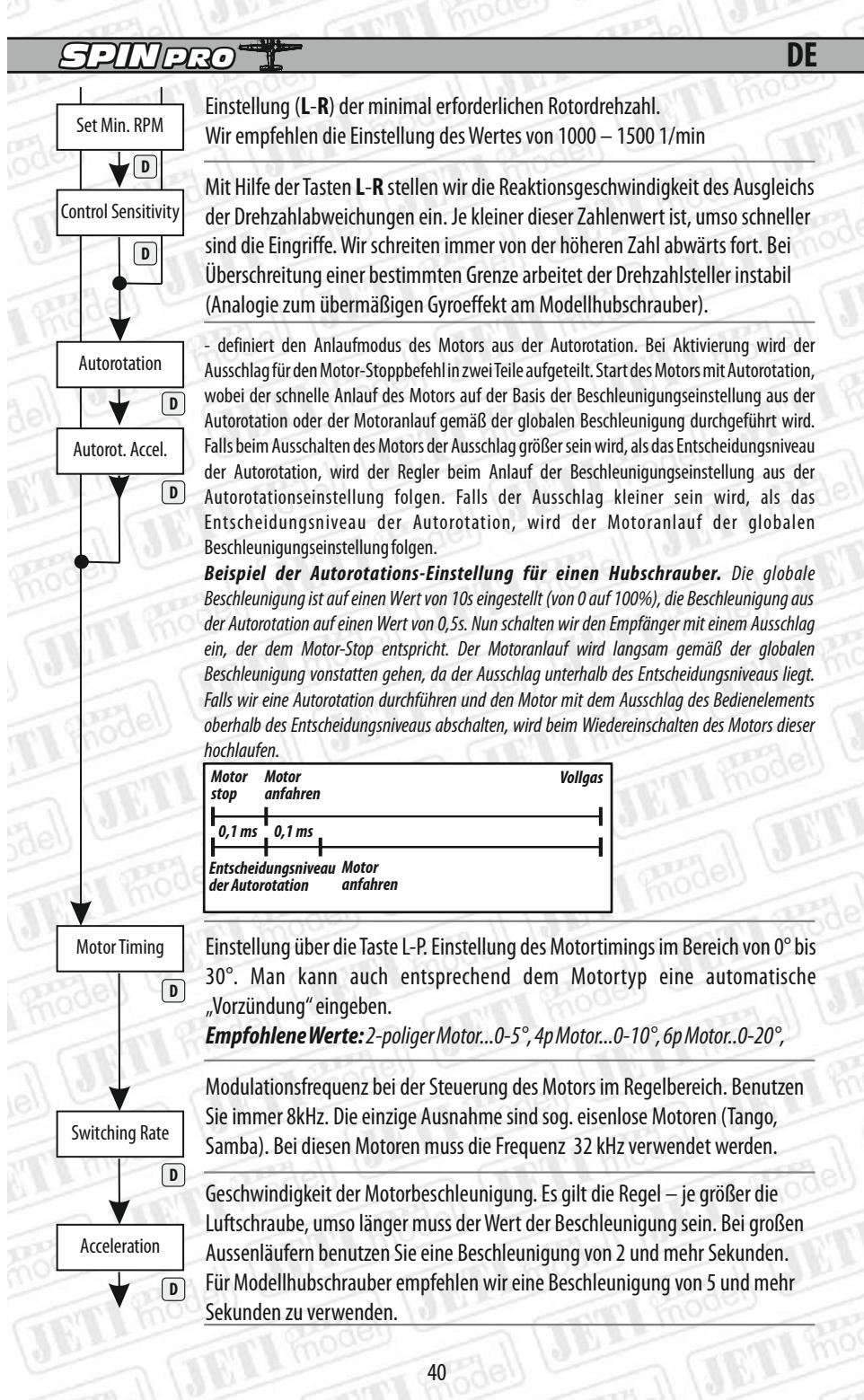
Über die Taste **D** gelangen Sie in die Auswahlzeile des Grundmenüs und wählen entweder das Auslesen der gemessenen Werte oder die Parameter-Einstellung des Drehzahlstellers. (**Measure or Setting**), mit den Tasten **L** und **P** wählen wir **MEASURE – MAN.SETTING – AUTO SET**.

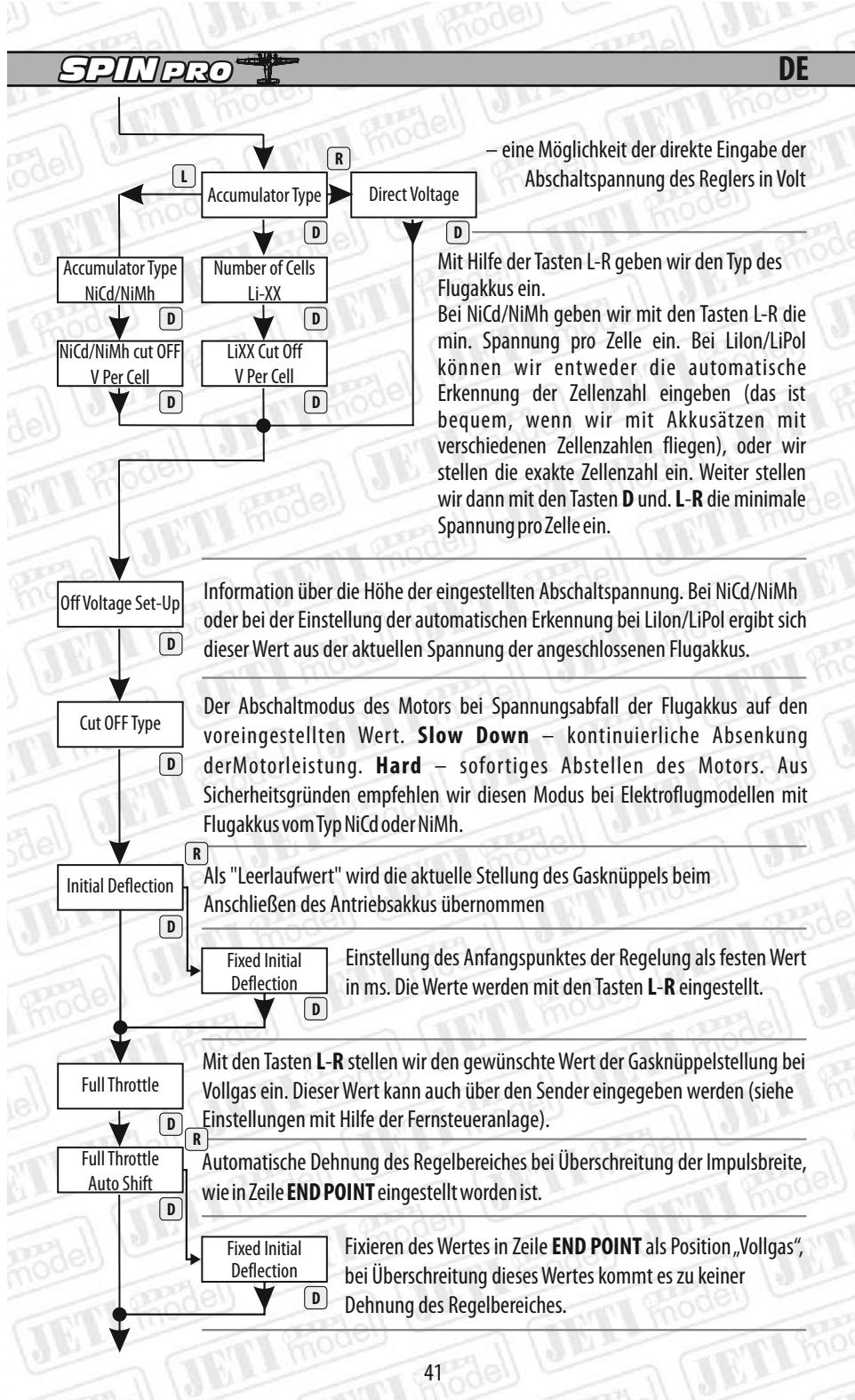
5.1 Messungen

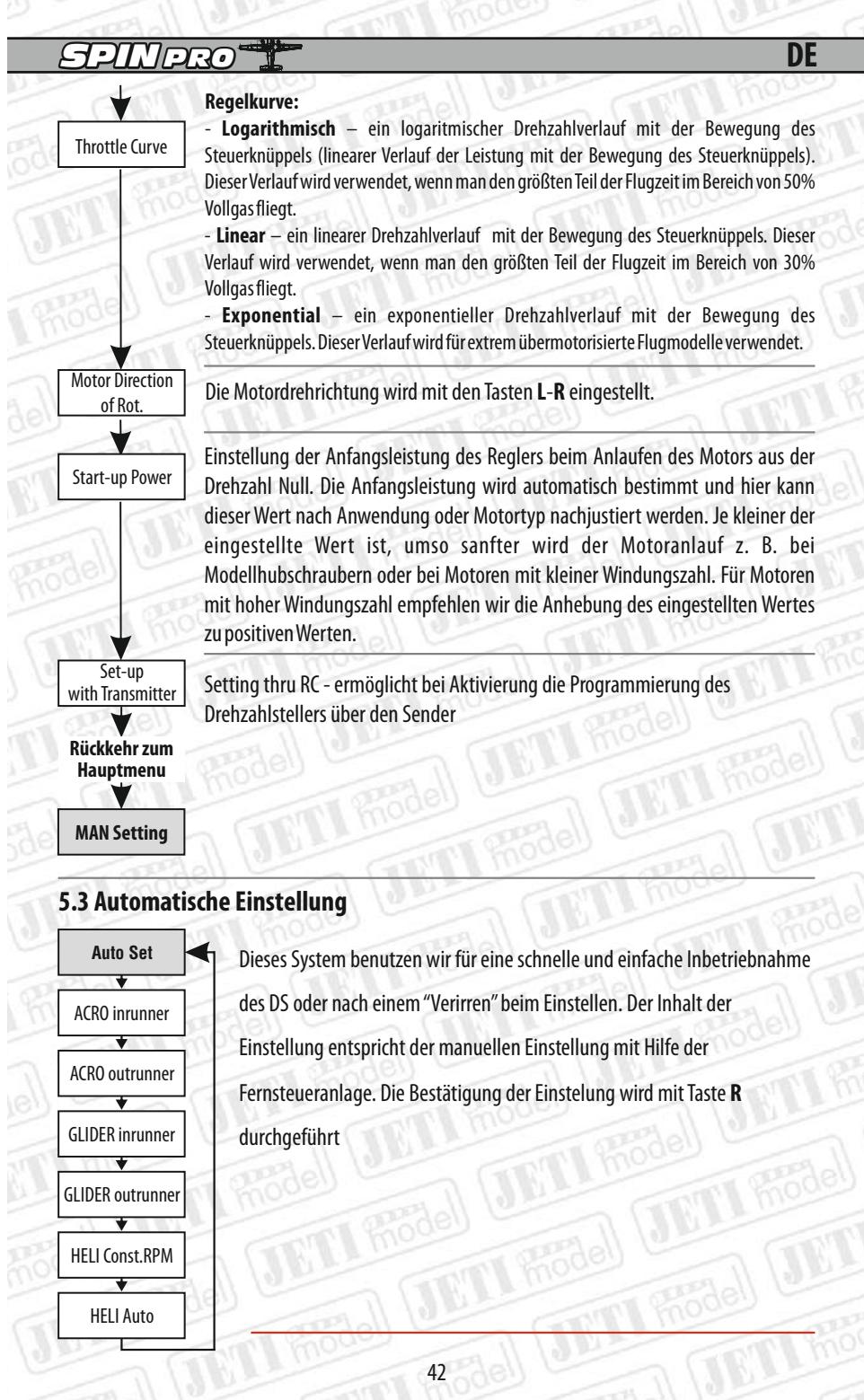














DE

6. Empfehlungen

6.1 Verlängerung von Anschlusskabeln.

Grundsätzlich verlängern wir Kabel vom Akku zum Drehzahlsteller. Bei einer Verlängerung um mehr als 20 cm muss zwischen die Leiter ein Elektrolytkondensator mit niedriger Impedanz und einer Kapazität von rund 300 μF eingefügt werden. Diese Kondensatoren müssen alle 25-30cm Kabellänge eingefügt werden.

6.2 Mehrmotorige Modelle

Wir empfehlen für jeden der Motoren den gleichen DS-Typ zu verwenden. Bei gleichzeitiger Verwendung mehrerer SPINpro BEC Drehzahlsteller darf nur ein BEC verwendet werden! Bei den weiteren DS bleiben die Schalter in der Position "AUS". Bei der Verwendung von DS mit BEC im allgemeinen muss nur ein gemeinsamer Antriebsakkumulator verwendet werden. Wenn wir 2 oder mehrere Akkus verwenden wollen, müssen diese parallel geschaltet werden.

6.3 Verfahren zur Feststellung der Anzahl von Motorpolen

Wenn Sie die Anzahl der Pole Ihres Akkus nicht kennen, wenden Sie sich an den Hersteller.

Falls Sie einen Drehzahlmesser haben und kennen die Übersetzung des Getriebes bei Direktantrieb können Sie durch nachfolgendes Verfahren die Polanzahl feststellen.

Lassen Sie den Motor laufen und messen Sie mit dem Drehzahlmesser die maximale Drehzahl der Luftschaube (des Rotors). Schließen Sie die JETI Box an und gehen Sie im Menü MESSEN über zur Anzeige der maximalen Luftschaubendrehzahl (Max. Prop RPM). Falls der angezeigte Wert nicht mit dem von Ihnen gemessenen Wert übereinstimmt, kontrollieren Sie die Einstellung der Getriebeübersetzung (Gear) und ändern weiterhin die Einstellung der Polanzahlen so lange, bis die von Ihnen gemessene Drehzahl mit der Messung der JETI Box übereinstimmen werden (Max. Prop RPM). Das Ergebnis ist die Polanzahl Ihres Motors (Motor Pole No.)

7. Sicherheitsinformationen, Garantie, Service und technischer Support

7.1 Sicherheit und Garantiebedingungen

- Nach Anschluss der Antriebsakkus behandeln Sie das Modell so, als ob jederzeit die Luftschaube anlaufen könnte!!!
- Achten Sie stets darauf die Controller SPINpro im Trockenen zu betreiben. Feuchtigkeit kann zur Korrosion der Elektronik führen. Falls in das Produkt Flüssigkeit/Feuchtigkeit eindringt, schalten sie es ab und lassen es austrocknen. Ein durch Feuchtigkeit beschädigter Controller ist meistens nicht mehr reparabel und Garantieansprüche werden nicht anerkannt.
- Öffnen Sie die Controller SPINpro nicht und versuchen Sie keine Änderungen durchzuführen. Dies kann zu einer irreversiblen Beschädigung des Produkts führen und damit zum Erlöschen des Garantieanspruchs.
- Achten Sie auf richtiges Verlöten und Sauberkeit aller Anschlussstecker. Bei falsch angelöteten Anschlusssteckern (hauptsächlich am Motor) kann der Controller abbrennen. Ein derartig beschädigter Controller ist nicht mehr reparabel und Reklamationen werden nicht anerkannt.



DE

- Betreiben Sie den Controller immer mit der vorgeschriebenen Versorgungsspannung und mit empfohlenem Strom. Andernfalls kann es zu einer irreversiblen Beschädigung des Controllers kommen und eine Reklamation wird nicht anerkannt.
- Beim Anschluss des Controllers achten Sie auf die richtige Polarität. Im Falle einer Verpolung kann es zu einer irreversiblen Beschädigung des Controllers kommen und eine Reklamation wird nicht anerkannt!
- Bevor Sie einen nicht funktionierenden Controller zum Service einschicken, kontrollieren Sie, ob vielleicht nicht eine Falscheinstellung über die JETIBOX vorliegt, versuchen Sie ggf. einen der voreingestellten Modi (siehe Punkt 6.3).
- Stellen Sie eine genügende Kühlung des Controllers sicher. Andernfalls kann es zu einer Abschaltung des Wärmeschutzes oder im Extremfall zur irreversiblen Beschädigung des Controllers kommen.
- Die Controller SPINpro dürfen nur von einem kualifizierten Service repariert werden, andernfalls erlischt die Garantie.

7.2 Garantie und Service

Für das Gerät wird eine Garantie von 24 Monaten ab Verkaufstag unter der Voraussetzung gewährt, dass es in Übereinstimmung mit dieser Anleitung bei vorgeschriebener Spannung betrieben worden ist und keine mechanischen Schäden aufweist. Bei der Reklamation des Produkts legen Sie immer den Kaufbeleg bei. Der Service im Garantiefall und auch danach wird vom Hersteller durchgeführt.

7.3 Technische Unterstützung

Falls Sie sich nicht sicher sind bei der Einstellung und Funktion des Produkts, kontaktieren Sie unseren technischen Dienst. Eine technische Untestützung bietet Ihnen entweder der Händler oder Jeti model direkt. Nähere Informationen finden Sie auf durch den Thermoschutz www.jetimodel.de.

SPIN PRO**DE****8. Entsorgung von gebrauchten elektronischen Geräten****DEUTSCH****Benutzerinformationen zur Entsorgung von elektrischen und elektronischen Geräten (private Haushalte)**

Entsprechend der grundlegenden Firmengrundsätzen der Panasonic-Gruppe wurde Ihr Produkt aus hochwertigen Materialien und Komponenten entwickelt und hergestellt, die recycelbar und wieder verwendbar sind.

Dieses Symbol auf Produkten und/oder begleitenden Dokumenten bedeutet, dass elektrische und elektronische Produkte am Ende ihrer Lebensdauer vom Hausmüll getrennt entsorgt werden müssen. Bringen Sie bitte diese Produkte für die Behandlung, Rohstoffrückgewinnung und Recycling zu den eingerichteten kommunalen Sammelstellen bzw. Wertstoffsammelhöfen, die diese Geräte kostenlos entgegennehmen.

Die ordnungsgemäße Entsorgung dieses Produkts dient dem Umweltschutz und verhindert mögliche schädliche Auswirkungen auf Mensch und Umwelt, die sich aus einer unsachgemäßen Handhabung der Geräte am Ende Ihrer Lebensdauer ergeben könnten. Genauere Informationen zur nächstgelegenen Sammelstelle bzw. Recyclinghof erhalten Sie bei Ihrer Gemeindeverwaltung.

Für Geschäftskunden in der Europäischen Union

Bite treten Sie mit Ihrem Händler oder Lieferanten in Kontakt, wenn Sie elektrische und elektronische Geräte entsorgen möchten. Er hält weitere Informationen für sie bereit.

Informationen zur Entsorgung in Ländern außerhalb der Europäischen Union

Dieses Symbol ist nur in der Europäischen Union gültig.

Abkürzungsverzeichnis

SPI	Serial Peripheral Interface
I²C	Inter-Integrated Circuit
XPS	Extrudiertes Polystyrol
CAN	Controller Area Network

Abbildungsverzeichnis

2.1	Funktionsprinzip eines Hovercrafts	2
3.1	Aufbau erster Prototyp	3
4.1	Konstruktion 3D-Modell gesamt	4
4.2	Bodenplatte 3D-Modell	5
4.3	Bodenplatte Ansicht unten	7
4.4	Bodenplatte Ansicht unten seitlich	7
4.5	Zeichnung Bodenplatte Pappel	8
4.6	Zeichnung Bodenplatte XPS 1	9
4.7	Zeichnung Bodenplatte XPS 2	10
4.8	Zeichnung Verstärkung Holzlatte links	11
4.9	Zeichnung Verstärkung Holzlatte rechts	12
4.10	Zeichnung Staffel Motorhalterung	13
4.11	Zeichnung Motorhalterung	14
4.12	Zeichnung Motorregler-Halterung	15
4.13	Foto Gitter	16
4.14	Zusammenbau Bodenplatte Sicht von unten	17
4.15	Zusammenbau Bodenplatte	18
4.16	Bodenplatte Winkelverbinder	18
4.17	Befestigung Lochband	19
4.18	Hinterer Aufbau 3D-Modell	20
4.19	Hinterer Aufbau Ansicht schräg hinten	22
4.20	Hinterer Aufbau Ansicht oben	22
4.21	Zeichnung Hauptplatte	23
4.22	Zeichnung Platte innen unten	24
4.23	Zeichnung Platte innen 1 schräge Kante	25
4.24	Zeichnung Platte innen 2 schräge Kanten	26
4.25	Zeichnung Platte außen oben	27
4.26	Zeichnung Platte außen 1 schräge Kante	28
4.27	Zeichnung Platte außen 2 schräge Kanten	29
4.28	Zeichnung Platte außen Seite	30
4.29	Zeichnung Verstärkung hinten Mitte	31
4.30	Zeichnung Verstärkung hinten unten	32
4.31	Zeichnung Verstärkung hinten schräg	33
4.32	Zeichnung Verstärkung unten	34
4.33	Zeichnung Servo Halter seitlich	35
4.34	Zeichnung Servo Halter seitlich 2	36
4.35	Zeichnung Servo Halter hinten	37

4.36 Zeichnung Servo Halter lang	38
4.37 Zeichnung Servo Halter Hauptplatte	39
4.38 Zeichnung Kabelbox Platte kurz	40
4.39 Zeichnung Kabelbox Platte lang	41
4.40 Zeichnung Kabelbox Deckel	42
4.41 Zeichnung Fahnenhalter Hauptteil	43
4.42 Zeichnung Fahnenhalter Servo	44
4.43 Zeichnung Fahnenhalter Rechteck	45
4.44 Zusammenbau hinterer Aufbau	46
4.45 Lenkung 3D-Modell	47
4.46 Lenkung Ansicht vorne, offen	49
4.47 Lenkung Ansicht Display-Halterung	49
4.48 Zeichnung Bodenplatte	50
4.49 Zeichnung Platte groß mit Loch	51
4.50 Zeichnung Platte Seite lang	52
4.51 Zeichnung Platte Seite kurz	53
4.52 Zeichnung Poti Halterung	54
4.53 Zeichnung Adapter Poti Teil 1	55
4.54 Zeichnung Adapter Poti Teil 2	56
4.55 Zeichnung Adapter Lenker	57
4.56 Zeichnung Stange Mitte	58
4.57 Zeichnung Gewindestange	59
4.58 Zeichnung Displayhalterung Halterung	60
4.59 Zeichnung Displayhalterung Hauptteil	61
4.60 Zeichnung Displayhalterung Deckel	62
4.61 Foto Federsystem Lenkung	63
4.62 Zusammenbau Lenkung	64
4.63 Gesamter Zusammenbau	66
5.1 Hacker A200 Elektromotor	67
5.2 Hinterer Propeller	68
5.3 Unterer Propeller	69
5.4 MasterSPIN 220 Pro OPTO	70
5.5 Jetibox	70
5.6 Servos zur Fahnensteuerung	72
5.7 Schaltplan Leistungselektronik	73
5.8 Verkabelung Kupferschienen	74
5.9 TE Connectivity EV200 Kfz-Relais	75
5.10 Automotive Kfz Sicherung	75
5.11 Buck Converter	76
5.12 Relais	77
5.13 TopFuel Power-X 7500mAh 7S	78

5.14	Ladegerät	79
5.15	Schaltnetzteil	80
6.1	Arduino Nano	82
6.2	CAN-Bus Modul	82
6.3	Visual Studio Code Webseite	84
6.4	PlatformIO Webseite	85
6.5	Schaltplan Boardelektronik	86
6.6	Platine zur Regleransteuerung	87
6.7	Schaltplan der Regler-Platine	88
6.8	PCB-Layout der Regler-Platine	89
6.9	Platine des Lenkers	92
6.10	NEO-8M GPS-Modul	92
6.11	Daumengas	92
6.12	Schaltplan der Lenker-Platine	93
6.13	PCB-Layout der Lenker-Platine	94
6.14	Platine zur Regleransteuerung	99
6.15	Adafruit 16-Channel Servo Driver	99
6.16	Schaltplan der Servoansteuerungs-Platine	100
6.17	PCB-Layout der Servoansteuerungs-Platine	101
6.18	ADS1115 Modul	104
6.19	LM35	104
6.20	Platine Temperatursensoren	104
6.21	Schaltplan der Temperatursensoren-Platine	105
6.22	PCB-Layout der Temperatursensoren-Platine	106
6.23	Zur Vorladung verwendeter Leistungswiderstand	107
6.24	Platine zur Relaisansteuerung	107
6.25	Schaltplan der Platine zur Relaisansteuerung	108
6.26	PCB-Layout der Platine zur Relaisansteuerung	109
6.27	Platinen auf der rechten Seite des Bootes	110
6.28	Platinen auf der linken Seite des Bootes	110
6.29	Nextion-Display	111
6.30	Webseite des Nextion Editors	112
6.31	Auswahl des richtigen Bildschirmes im Nextion Editor	113
6.32	Auswahl der richtigen Bildschirmausrichtung im Nextion Editor	113
6.33	Hauptseite der Bedienoberfläche	114
6.34	Einstellungsseite der Bedienoberfläche	115
6.35	Infoseite der Bedienoberfläche	116
7.1	Not-Aus-Schlüsselschalter	118
7.2	Daumengas	119
7.3	Unteres Gitter	120

7.4	Hinteres Gitter	120
7.5	Offene Ladebox & Ladekabel	121
7.6	Bremsschutz	122
7.7	Position des Motorreglers	122
9.1	Ladegerät Hauptanzeige	126
9.2	Ladegerät Task-Auswahl	127
9.3	Ladegerät Einstellungen	127
9.4	Ladegerät Schnellladenvorgang	128

Tabellenverzeichnis

4.1	Stückliste Bodenplatte	6
4.2	Stückliste Aufbau hinten	21
4.3	Stückliste Aufbau hinten	48
4.4	Bestelliste Konstruktion	65
5.1	Hacker A200 Leistungsdaten	67
5.2	Durch JetiBox am Regler konfigurierte Parameter	71
5.3	Stückliste der Leistungselektronik	81
6.1	Verwendete CAN-Bus Adressen	83
6.2	Stückliste der Boardelektronik	117
8.1	Kostenrechnung – Konstruktion	123
8.2	Kostenrechnung – Leistungselektronik	123
8.3	Kostenrechnung – Leistungselektronik	124
8.4	Kostenrechnung – Gesamtkosten	124
11.1	Zeitaufzeichnung Stefan Deimel	133
11.2	Zeitaufzeichnung Philipp Eilmsteiner	136
11.3	Zeitaufzeichnung Julia Stöger	138

Literaturverzeichnis

- [1] <https://youtu.be/fBdnjvSfNkg>
Abgerufen am 17.3.2021
- [2] <https://www.autodesk.com/products/inventor/overview?term=1-YEAR&support=null>
Abgerufen am 16.3.2020
- [3] <https://platformio.org/platformio-ide>
Abgerufen am 19.3.2020
- [4] <http://www.jetimodel.com/en/show-file/47>
Abgerufen am 25.3.2020
- [5] <http://www.jetimodel.com/de/show-file/125>
Abgerufen am 25.03.2021
- [6] <https://github.com/autowp/arduino-mcp2515>
Abgerufen am 31.03.2021
- [7] <https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library>
Abgerufen am 31.03.2021
- [8] https://github.com/adafruit/Adafruit_ADS1X15
Abgerufen am 31.03.2021
- [9] https://github.com/itead/ITEADLIB_Arduino_Nextion
Abgerufen am 31.03.2021
- [10] <https://github.com/mikalhart/TinyGPSPlus>
Abgerufen am 31.03.2021
- [11] <http://www.jetimodel.com/en/show-file/935/>
Abgerufen am 01.04.2021