
wiremenl

Wirement

Florian Frank

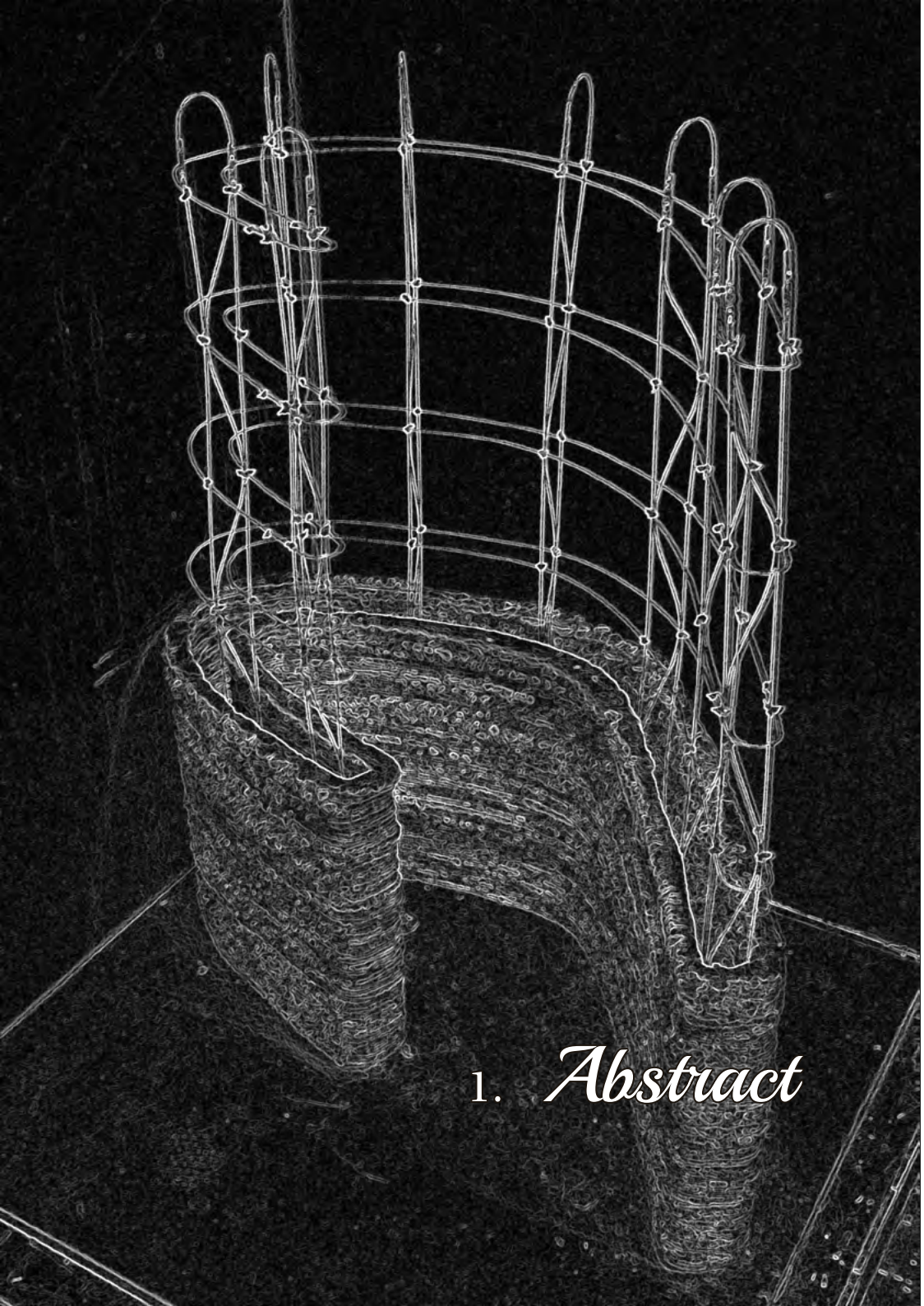
Diplomarbeit
eingereicht an der
Leopold-Franzens-Universität Innsbruck
Fakultät für Architektur

zur Erlangung des akademischen Grades
Diplom-Ingenieur

Beurteiler:
Univ.-Prof. Dipl.-Ing. Marjan Colletti, PhD
Institut für Experimentelle Architektur / Hochbau

Innsbruck, Oktober 2016

Index



1. Abstract

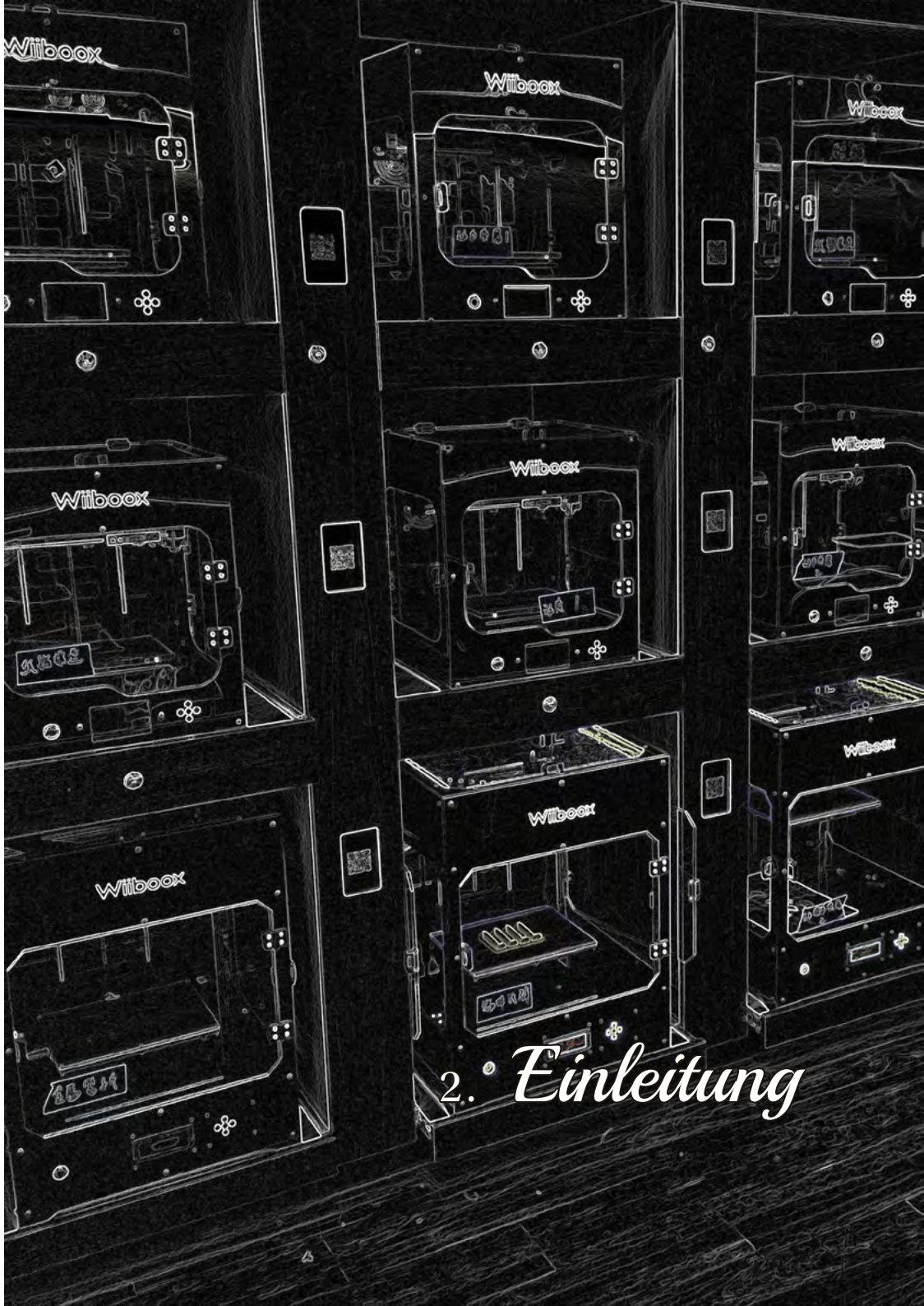
In der zeitgenössischen Architektur spielt der Ausdruck über Freiformen eine zentrale Rolle. Die aktuellen Möglichkeiten in der Herstellung von doppelt gekrümmten Formen aus Beton werden durch zwei Faktoren gebremst: Die aufwendige und kostenintensive Herstellung von Betonschalungen und die komplexe Armierung dieser Formen. Die Produktion von Bewehrungen für doppelt gekrümmte Betonformen erweist sich zurzeit als sehr schwierig und wird in der Praxis nur von Hand ausgeführt. Obwohl zurzeit fast alle Produktionsschritte des Bauens mit digitaler Technik erleichtert bzw. schon automatisiert worden sind, gibt es in diesem Bereich noch technische Herausforderungen. Die Automatisierung in diesem Bereich birgt enormes Potential für neue Gestaltungsmöglichkeiten.

Diese Arbeit befasst sich mit der plastischen 3D-Verformung von Metallstäben für den konstruktiven Einsatz. Ziel ist es, eine Reihe von digitalen Werkzeugen zur Verfügung zu stellen, die den Gestaltungsprozess für Designer und Konstrukteure vereinfachen und zugleich eine Rückmeldung über mögliche Probleme in der Fertigung geben. Diese ermöglichen neue Planungsprozesse und Entwürfe für Architekten und Künstler.

Als Grundlage wurde die Theorie verschiedener Umformtechniken evaluiert, um eine neue digital kontrollierte Verformung zu entwerfen. Diese Technik nützt die Biegeeigenschaften von Stahl und verteilt die Materialbelastungen bei der plastischen Verformung gleichmäßig.

Da die Verformung von Metallen sehr schwer zu berechnen bzw. nicht exakt genau vorhergesagt werden kann, wurde ein Prototyp gebaut, welcher die digitale Technik physisch evaluiert. Im Prototyp werden Metallstäbe (Durchmesser: 1-2mm) räumlich gebogen und die gemessenen Ergebnisse wiederum in die Berechnung einbezogen, um nachfolgende Ergebnisse zu verbessern. Über einer Feedback-Loop wird dem Benutzer die Möglichkeit gegeben, Umformung als Gestaltungswerkzeug zu verwenden, ohne sich mit den technischen Hintergrundprozessen beschäftigen zu müssen.

Die Versuche zeigen, wie durch geometrische Berechnung die Auswertung empirisch erhobener Daten und die digitale Kontrolle dieser Umformtechnik eine Verbesserung von Bautoleranzen bei komplexen Geometrien erreicht werden kann. Die Ergebnisse der Versuche wurden zu digitalen Werkzeugen für Designer und Planer weiterentwickelt. Um die Design- und Fabrikationsaspekte zu evaluieren, wurde an einem internationalen Workshop mit Studenten verschiedener Jahrgänge diese Techniken anhand von Konstruktionen und Designobjekten erprobt.



2. Einleitung

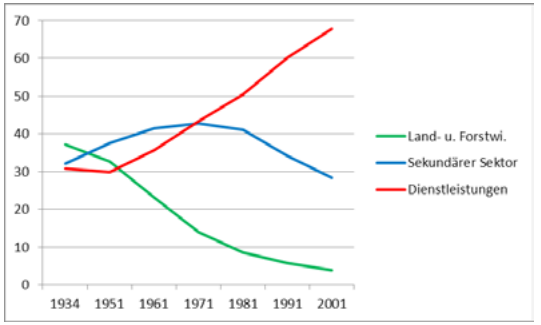
Der Titel **„Wirement“** setzt sich aus den englischen Begriffen wire (dt. Draht) und reinforcement (dt. Bewehrung/ Armierung) zusammen. Zur Beweisführung für ein Umformverfahren von Bewehrungs-Stahlstäben wurde in dieser Arbeit ein Material verwendet, das den technischen Aufwand und die Kosten möglichst gering hält. Stahldraht erwies sich als gutes Material, da es von den Materialeigenschaften sehr nahe am Bewehrungsstahl liegt und somit gut als Maßstabsmodell verwendet werden kann.

Ziel und Beweggrund dieser Arbeit ist das Voranbringen der Rolle des Architekten in der Branche und somit in der Gesellschaft. Mit technologischem Fortschritt will diese Arbeit neue Gestaltungsmöglichkeiten für Architektur und Design bieten, sowie Anregung für den aktuellen Architekturdiskurs schaffen.

Der Begriff „Wirement“ soll deshalb auch ein Impuls für die Vernetzung (to wire = dt. verkabeln) der Rolle des Architekten und Designers im kulturellen, sowie technischen Kontext der Gesellschaft sein.

2.1. Von der Produktions- zur Dienstleistungsgesellschaft

Gegenwärtig findet eine angeregte Diskussion über mögliche Auswirkungen der Verwirklichung von „Industrie 4.0“ bzw. „Wirtschaft 4.0“ statt. Besonders der rasante technologische Fortschritt in der Automatisierung verunsichert und lädt dazu ein, einen Blick auf die Verteilung der Beschäftigung in den einzelnen Wirtschaftssektoren im Laufe des letzten Jahrhunderts zu werfen.



Quellen: ÖSTAT bzw. Statistik Austria: Volkszählungen 1934-2001.
Erwerbspersonen = Selbstständige, Mithelfende, unselbstständig Beschäftigte (ohne geringf. Besch.), Arbeitslose.
Sekundärer Sektor = Verarbeitende Industrie und produzierendes Gewerbe, Bergbau, Bauwirtschaft, Energie- und Wasserversorgung.

Abb. 2.1: Anteile der Wirtschaftssektoren an der Gesamtheit der Erwerbspersonen in Österreich 1934-2001 (in %) Quelle: Michael Mensch - AK Wien

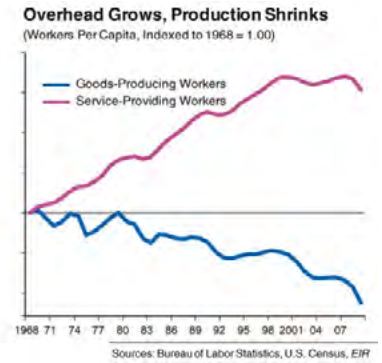


Abb. 2.2: Zeitlich veränderte Beschäftigungsverteilung zwischen Dienstleistung und Produktion in den Vereinigten Staaten

Quelle: solidaritaet.com

Ab dem Ende des zweiten Weltkrieges in Europa, aber besonders ab Ende des ersten Weltkrieges und der Verbreitung des Fordismus in der westlichen Industrie entsteht ein Paradigmenwechsel in der Verteilung der Beschäftigung. Durch die Entstehung der Massenfertigung und die Effizienzsteigerung ist es erstmals möglich über Arbeitsverteilung mit Hilfe spezialisierter monofunktionaler Maschinen in einer Fließbandanordnung die Produktivität zu

steigern (Kernelemente des Fordismus). Mit der Erweiterung des Postfordismus und dem Outsourcing, sowie der Globalisierung gegen Ende des letzen Jahrhunderts wurde die Flexibilität erhöht, was eine weitere Verteilung der Produktionsschritte zur Folge hatte. Eine Steigerung in der spezialisierten Tätigkeit von Beschäftigten kennzeichnete die Entwicklung in fast jeder Branche. Die Komplexität von Produkten und Standards stiegen gleichermaßen, wie der Lebensstandard in der westlichen Welt. Auf die Beschäftigungsverteilung wirkte sich (vereinfacht dargestellt) die Steigerung von Produktivität auf eine sinkende Beschäftigung im produzierenden Gewerbe aus. Die steigende Komplexität von Produkten bewirkte unter anderem eine Zunahme im Dienstleistungssektor.

Längst hat sich die Theorie des Fordismus auf andere Bereiche der Wirtschaft, selbst auf die Dienstleistung, ausgedehnt. Zunehmend wurden die gesteigerten Qualitätsansprüche in Gesetzen und Standards verankert. Bei der Steigerung der Komplexität wurde auch der Entstehungsprozess eines Produktes von seinem Entwurf, der Planung des ersten Prototyps über die Prüfung bis hin zur Serienproduktion erfasst. Längst ist es durch das nötige Spezialwissen in den einzelnen Phasen nicht mehr möglich, alle Schritte von einer Person genau erfassen zu können. Mit der Digitalisierung verschiedener Produktionsphasen in den letzten Jahrzehnten wurde dieser Effekt noch einmal gesteigert und führte zu einer noch höher spezialisierten Beschäftigung.

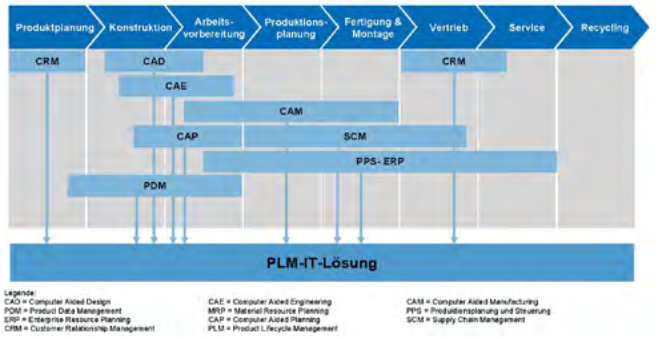


Abb. 1.1: Struktur eines Product Lifecycle Management im IT-Kontext

Quelle: plm-info.de

Diese Auswirkungen in der Wirtschaft änderten auch den Verantwortungsbereich der Architekten, Designer oder Gestalter. Die neue Aufgabenverteilung und Spezialisierung führte zu klar getrennten Berufsgruppen mit genau definierten Verantwortungsbereichen innerhalb der Branche. In den verschiedenen Detailbereichen eines Projektes wurden Aufgaben durch erforderliche Fachkenntnisse

von Bauingenieuren, Spezialisten und Fachplanern übernommen. Diese Verteilung von Verantwortungsbereichen bei Projekten führt im Planungsprozess zu einem verminderten Austausch von Spezialwissen und erschwert in einigen Fällen die kreative Arbeit. Die kreativen Berufe und vor allem das klassische Berufsbild Architekt als Generalist, Kombinierer und Universal-Denker leidet unter der zunehmenden Komplexität von Herstellungsprozessen, entstanden durch den Wandel der Wirtschaft.

2.2. Isolierter Architekturdiskurs

Der Freiraum eines Architekten in der Gestaltungsmöglichkeit steht im engen Zusammenhang mit der Komplexität einer Planungsaufgabe. Längst ist nicht mehr ein Zusammenhang zwischen Größe und Komplexität eines Projektes feststellbar. Durch den digitalen Fortschritt ist eine große Bandbreite an neuen Gestaltungsmöglichkeiten entstanden. Auch ist durch die teilweise Automatisierung ein immer höherer Detaillierungsgrad in der Planung möglich. Dieser Detaillierungsgrad wird vom produzierenden Gewerbe, je nach Komplexität der Fertigung gefordert um die Umsetzbarkeit zu sichern. Meist werden Planungen im Bereich der hohen Detaillierung aber von Fachplanern und ausführenden Unternehmen übernommen und fallen deshalb nicht mehr in den Gestaltungsbereich des Architekten. Nur große Architekturbüros können sich bei komplexen Projekten eine Aufgabenverteilung auf Spezialbereiche innerhalb des Büros leisten und in Rücksprache mit den einzelnen produzierenden Gewerben die Detailplanung selbst auf sich nehmen.

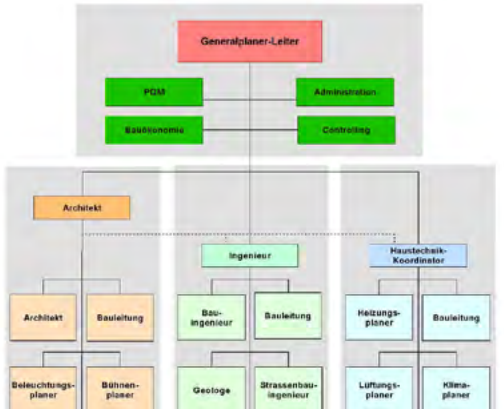


Abb. 2.4: Organisationsstruktur Generalplaner
Quelle: Assocation suisse pour l'Economie de la Construction

Für kleine Büros bzw. selbstständige Architekten und Designer ist bei solchen Projekten meist nur eine Abwicklung der Planungsleistung über Fachplaner möglich. Der Zugang zu Spezialwissen in den verschiedenen Bereichen eines Projektes wird immer schwieriger. Gerade für den kreativen Teil der Planung ist es aber wichtig Zugang zum Knowhow über die Ausführung der Gestaltung zu erlangen. Die für das Design wesentlichen Aspekte können so in der Umsetzung des Vorhabens nicht zu Problemen führen. Gerade dieser Zugang ist nur mit entsprechendem Zeitaufwand möglich und längst nicht für alle Kreative machbar.

Im aktuellen Architekturdiskurs bewirkt diese Entwicklung einen immer größer werdenden Abstand zwischen Gestaltung und Herstellung. Der Diskurs verurteilt zum Teil das aktuelle Baugeschehen und fordert gleichzeitig einen größeren Einfluss darauf. Die für das Baugeschehen verantwortlichen Fachgruppen und Entscheidungsträger bewegen sich vorwiegend im Kontext der Funktion und Wirtschaftlichkeit, weniger im Bereich der Gestaltung. Selten ist es möglich mit dem nötigen Wissen über diese Faktoren, Bauvorhaben mit dem Diskurs über die bauliche Umsetzung von Architektur in Einklang zu bringen. Die innovativen Büros unserer Zeit machen es jedoch vor, wie mit den nötigen Anstrengungen in Fachbereichen gute Lösungen für die Verwirklichung von komplexen Bauvorhaben möglich ist.

Abb. 2.5: Modell, Foto während der Bewehrungseinlegung und gebautes Projekt: National Taichung Opera von Toyo Ito
Quelle: Marco Rinaldi & Florian Frank

2.3. *Generalist vs. Spezialist*

Durch die genaue Kompetenzverteilung in den Planungsebenen gibt es für den Generalisten ohne Spezialwissen nahezu kein Mitspracherecht bezüglich der Entscheidungen, die die technische Umsetzung betreffen. Den Spezialisten in der Projektierung ist es aufgrund ihrer fachlichen Kompetenz möglich Entscheidungen über Lösungen zu treffen, auch wenn diese sich negativ auf das Verständnis des Architekten und seiner Gestaltungsidee auswirken. Den Fachberufen ist es durch ihre Position auch möglich. Umsetzung und Machbarkeit komplexer Aufgaben abzuwenden, obwohl diese mit dem nötigen Zusammenwirken von kreativen und technischen Planungsgruppen realisierbar wären. Das häufig begrenzte Interesse des Spezialisten gegenüber dem Kreativdiskurs, aber auch das mangelhafte technische Verständnis des Gegenübers führt leider oft zu ungewollten Kompromissen. Als Beispiel zu einem dieser Fälle kann das Projekt Spittelau (Wien, AT), der Architektin Zaha Hadid genannt werden, wo die bauliche Ausführung nicht den Vorstellungen der Architektin entsprach.

Abb. 2.6: Entwurfsmodell von Zaha Hadid und gebautes Projekt der Siedlung Spittelau.

Quelle: wien.gv.at

In kreativen Branchen wie Architektur und Design bilden sich verschiedene Spezialisierungen und Nischen, um den Einfluss und die Gestaltungsmöglichkeit in den tieferen Planungsprozess zu erlangen. Gleichermaßen widmen sich die Designer und Architekten in der Rolle als Generalisten auch vermehrt den Aufgaben des Koor-

dinierens, Analysierens und Entdeckens von speziellen Lösungen für ihre Gestaltungsfreiheit. Die Rolle des Generalisten birgt durch die Zunahme an technischen Möglichkeiten und dem vereinfachten Zugang zu Technologien ein großes Potential für neue kreative Ansätze. Durch das breit gefächerte Interesse und der hinreichenden Kenntnis auf verschiedenen Gebieten ist es Kreativschaffenden möglich, spezielle Lösungen zu bewerten, zu kombinieren und für ihre Projektideen anzuwenden.

Vereinzelt ermöglichen Spezialisten aus Bereichen außerhalb des kreativen Kontexts einen vereinfachten Zugang zu ihrer Fachkenntnis. Etwa über Software reduzieren sie Komplexität und Aufwand von Planungsprozessen für ihren Fachbereich. Somit ist es auch Generalisten möglich die wesentlichen Merkmale spezieller Themenbereiche zu erkennen und diese in ihren kreativen Prozess einzubinden. Viele dieser digitalen Werkzeuge sind Anfang dieses Jahrhunderts aus gemeinnützigen Gruppierungen oder akademischen Projekten hervorgegangen. Während der letzten Jahre entdeckten vermehrt auch kommerziell orientierte Unternehmen die Möglichkeiten der Verfügbarmachung dieser Ressourcen als neues Geschäftsmodell. U.a. McNeel mit der Softwareerweiterung Grasshopper seit 2007, Autodesk mit Dynamo seit 2012 und Vectorworks mit Marionette seit 2015.

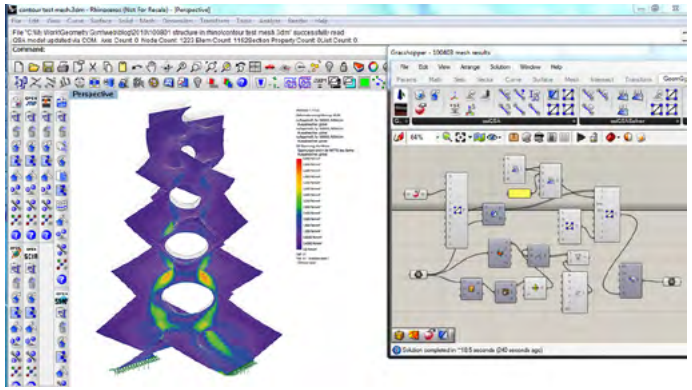


Abb. 2.7: FEM Berechnung der Statik einer komplexen Geometrie mithilfe einer Schnittstelle für die Designsoftware Rhinoceros

2.4. *Design Repräsentation und Fabrikation*

Die Darstellung und Präsentation von Entwürfen ist für Architekten und Designer das Medium, um ihre Entwurfsideen zu kommunizieren. Die Qualität des Dargestellten, ob als Zeichnung oder Modell, ist entscheidend für den späteren Entstehungsprozess. Über Jahrzehnte fand durch die Digitalisierung ein ständiger Wandel an Repräsentationsmöglichkeiten statt. Wo es anfangs nur möglich war über Pläne, Skizzen und handgefertigten Modellen ihre Konzeptideen zu präsentieren, ist es durch neue Technologien nun auch möglich Komplexität zu veranschaulichen. Neue Gestaltungsmöglichkeiten können über digitale Zeichnung, Renderings und unlängst auch „virtual reality“-Modellen kommuniziert werden.

Abb. 2.8: 3-D gedrucktes Modell einer dreifach periodischen Minimalfläche, geplant als erschließbarer Turm.

Durch Technologien wie dem 3D-Druck ist es nun auch möglich vielschichtige Lösungen mit geringem Aufwand im physischen Modell darzustellen. Fast jede Geometrie, die in digitale Geometrieinformation gefasst wurde, kann durch verschiedene 3D-Druckverfahren physisch repräsentiert werden. Einzig die Materialeigenschaften und der Detailgrad des Druckverfahrens begrenzen die Machbarkeit im 3D-Druck. Durch die rasante Weiterentwicklung verschiedener digitaler Druckverfahren werden diese Grenzen immer wieder verschoben und Verfahren erschwinglicher. Durch die Entstehung dieser neuen Möglichkeiten stiegen bei nicht Fachkundigen auch Erwartungen und Anforderungen an Produkten. Dass diese Erwartungen nicht gleichgesetzt werden können mit den Möglichkeiten des „rapid prototyping“, liegt auf der Hand. Wieder ist Fachwissen nötig, um das eigentliche Produkt mit anderen Materialien und in einem anderen Maßstab als das 3D-Druck Modell herstellen zu können.

Die gesteigerte Vielschichtigkeit von Entwürfen bewirkt in der In-

dustrie eine Aufrüstung durch digitale Automatisierung. Numerisch gesteuerte Maschinen erlauben die digitalen Daten aus der Planung für die Produktion zu nutzen. Mit vergleichsweise einfachen Mitteln können nun komplexe Einzelfertigungen bewerkstelligt werden, die vorher nur mit enormen Zeit- und Kostenaufwand realisierbar gewesen wären. Ein Zugang zu diesen digitalen Fertigungsverfahren ist jedoch nur mit entsprechendem Fachwissen und Erfahrung möglich.

Abb. 2.9: Fertigung komplexer Akustikwände, gestaltet durch die Künstlerin Loredana Sperini
Umsetzung durch den Autor
Quelle: Kaulquappe GmbH

2.5. Additive, subtraktive und Umform-Fertigungsverfahren

Neue digitale Fertigungsverfahren sind Erweiterungen alter Fertigungsverfahren. Um die Fülle der verschiedenen Verfahren zu überblicken, wurde von der Fertigungsindustrie eine Klassifizierungen in der DIN 8580 festgelegt. Diese teilt alle Verfahren in Hauptgruppen ein, welche unter dem Merkmal des Zusammenhalts von (Material-)Teilchen gekennzeichnet sind. Es wird unterschieden zwischen:

- **Urformen** (Zusammenhalt schaffen), dem das Gießen, Sintern und die vergleichsweise jungen generativen Fertigungsverfahren, wie 3D-Druck bzw. Rapid Prototyping zugeordnet werden.
- **Umformen** (Zusammenhalt beibehalten), zu dem u.a. das Walzen, Schmieden, Tiefziehen und Biegen zählen.
- Unter **Trennen** (Zusammenhalt vermindern) fallen Verfahren wie Bohren, Sägen, Fräsen und Brennschneiden.
- Als **Fügen** (Zusammenhalt vermehren) werden Schweißen, Löten, Schrauben und Nieten klassifiziert.

Zwei weitere Hauptgruppen sind das „Beschichten“ und „Stoffeigenschaften ändern“, auf welche hier nicht näher eingegangen wird. Die meisten Neuerungen durch die digitale Automation waren bisher in den Fertigungsverfahren der Hauptgruppe Trennen zu verzeichnen. Numerisch gesteuertes Bohren und Sägen, sowie CNC-Fräsen und digitale Drehbänke werden in fast jedem holz- sowie metallverarbeitendem Betrieb verwendet. Auch wurde durch die Digitalisierung eine Vereinfachung für komplexe Fertigungsprozesse bei den Brennschneideverfahren wie dem Laser-, Wasser- oder Plasmaschneiden erreicht.

Im Unterschied zu den additiven (Urformen/Fügen) und subtraktiven (Trennen) Fertigungsverfahren, die durch Informationstechnologie einen neuen Aufschwung erfuhren, konnte die Umformtechnik bisher wenig durch Innovationen bereichert werden. Obwohl die additiven und subtraktiven Prozesse material- und zeitaufwendiger sind, war es aus technischen Gründen einfacher, diese Techniken zu digitalisieren. Beim Umformen müssen unter hohem Aufwand Repetitionen im Herstellungsprozess vorgenommen werden, um schwer berechenbare Materialaspekte einzugrenzen, damit Ergebnisse mit geringer Toleranz erreicht werden können. Deshalb werden diese Verfahren vor allem in der Serien-Fertigung eingesetzt.

Im Vergleich zu den additiven Prozessen wird beim Umformen der Ausgangsstoff in seiner atomaren Beschaffenheit nicht wesentlich verändert. Die zu erreichende geometrische Komplexität bei der Umformtechnik wird beschränkt durch die geometrische und materielle Beschaffenheit des Ausgangswerkstoffs, d.h. das gewünschte Fertigungsstück kann nur unter Berücksichtigung der Umformmöglichkeiten des Ausgangsstoffs erstellt werden. Bei den additiven Verfahren muss eine künstliche Verbindung (z.B. Kleben oder Schmelzen) zwischen den einzelnen Materialschichten erstellt werden, was die Schwachstelle für die Aufnahme von Belastungen darstellt. Bei subtraktiven Verfahren wird zwar der Zusammenhalt des fertigen Produktes auf atomarer Ebene auch nicht verändert, ein hoher Zeit und Materialaufwand ist jedoch bei komplexen Geometrien von großem Nachteil.

Allgemein ist zu den verschiedenen Fertigungsverfahren zu erwähnen, dass diese untereinander nur bedingt austauschbar sind und jede Methode ihre Berechtigung für diverse Produktionsschritte findet. Auch zu erwähnen sei die nötige Fachkenntnis in der Anwendung dieser Verfahren. Es ist für einen Generalisten (vgl. 2.3)

ohne Erfahrung nicht möglich unzureichend automatisierte und vereinfachte Fertigungsverfahren zu überblicken und für seine Gestaltungsfreiheit zu nutzen. Besonders für die Herstellung eines Entwurfs wäre ein Grundverständnis über Verwendung und Limitierung diverser Verfahren für Kreativschaffende von großem Vorteil.

Diese Arbeit beschäftigt sich mit dem Rollbiegen, einem Verfahren der Hauptgruppe Umformen. Es soll versucht werden über die nötigen informationstechnischen Mittel ein Entwurfswerkzeug für die Nutzung dieses Fertigungsverfahrens zu schaffen. Dabei sollen alle verfahrensbezogenen technischen Berechnungen von der Software übernommen werden, um dem Benutzer nur das nötige Grundverständnis für seinen Gestaltungsbereich zugänglich zu machen. Ziel dieser Arbeit ist es auch, dem Kreativen Gestaltungsmöglichkeit zu bieten und in der Fachdiskussion mit Spezialisten über entwurfsrelevante Problematiken diskutieren zu können.

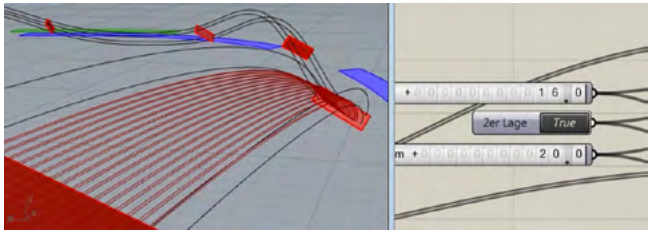


Abb. 2.10: Bewehrungsplanung am Elefantenhaus Zürich mit maßgeschneiderter Software

Umsetzung durch den Autor

Quelle: Kaulquappe GmbH

2.6. Interdisziplinäre Designprozesse für Architekten

Der zunehmend vereinfachte Zugang zu Fachwissen erleichtert ein Arbeiten über Fachbereiche hinaus. Der Einblick des Generalisten in die Spezialisten-Arbeit, aber auch der durch sein technisches Grundverständnis erleichterte Austausch mit dem Fachplaner fordert und fördert interdisziplinäre Designprozesse. Auch profitieren beide Gruppen von einem optimierten Austausch an Information und Knowhow.

Durch Probleme in öffentlichen Großprojekten in den vergangenen Jahren wird nun sogar auf politischer Ebene ein interdisziplinärer Austausch in der Planung gefordert und gesetzlich verankert. In diesem Bezug handelt es sich um softwaretechnischen Planungsdatenaustausch im standardisiertem Format.

»Unter Building Information Modeling (BIM) oder Gebäudedatenmodellierung versteht man die optimierte Planung und Ausführung von Gebäuden mit Hilfe entsprechender Software. BIM ist ein intelligentes digitales Gebäudemodell, das es allen Projektbeteiligten - vom Architekten und Bauherrn über den Haustechniker bis hin zum Facility Manager - ermöglicht, gemeinsam an diesem integralen Modell zu arbeiten und dieses zu realisieren.«

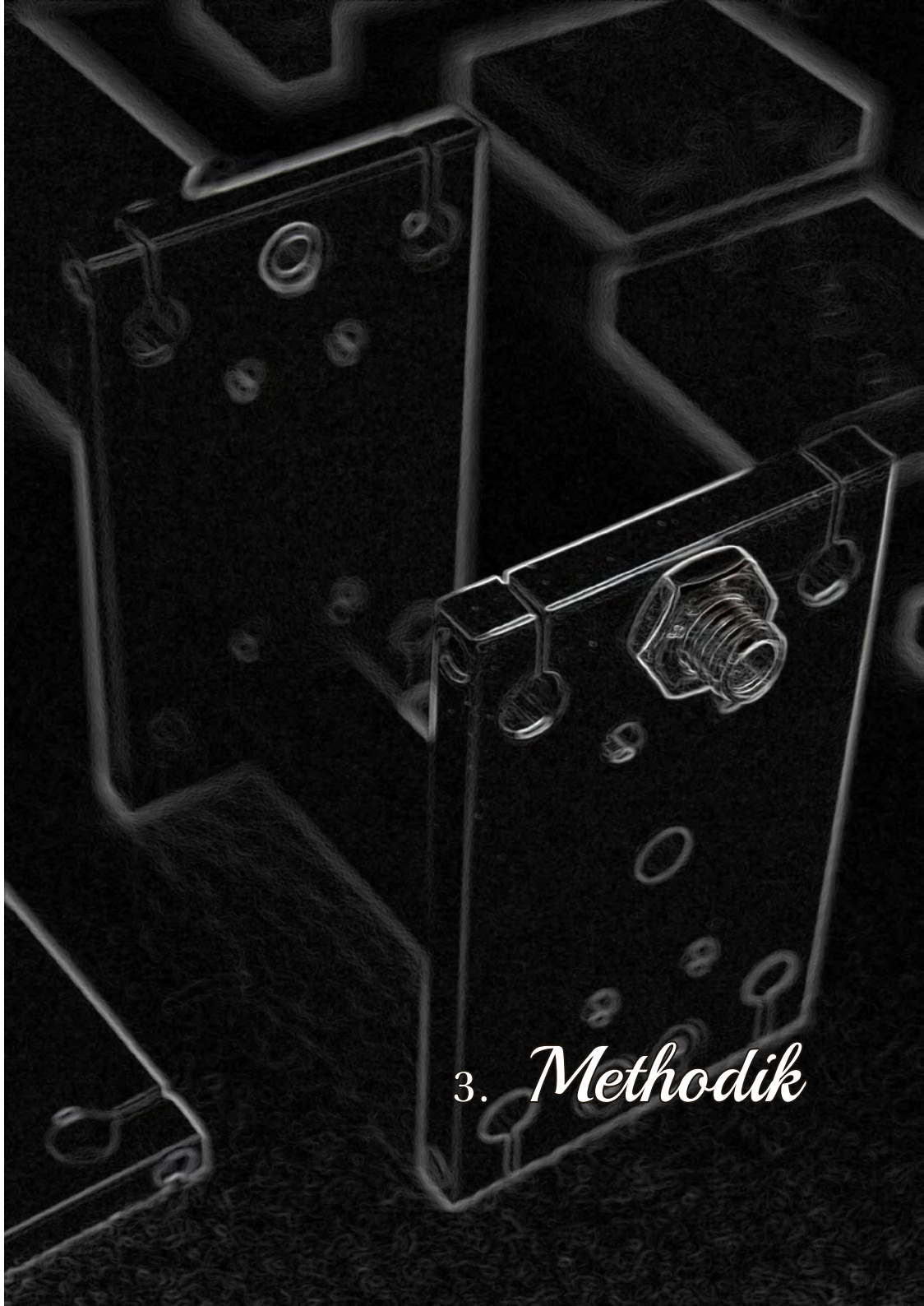
Quelle: BIM - AUSTRIAN STANDARDS

Diese Form der Interdisziplinarität führt aber zu keiner erweiterten Gestaltungsmöglichkeit. Wenn man die Thematik und die technischen Limitierungen, die eine solche Standardisierung mit sich bringen näher betrachtet, so stellt man fest, dass es sogar zu einer Einschränkung der Gestaltungsmöglichkeit führt. Es ist abzuwarten, bis Verbesserungen in diesem Bereich erreicht werden, welche es ermöglichen, komplexe Formgebung und Gestaltungsprozesse mithilfe dieser klassenbasierten Planungstechnik erfassen zu können.

Der branchenübergreifende Austausch ist und wird jedenfalls durch die gestiegene Komplexität der Aufgaben immer wichtiger. Diese Arbeit soll demonstrieren, wie ein Austausch zwischen Disziplinen möglich ist und die Gestaltungs- und Produktionsmöglichkeiten erweitern. Die fachlich fundierte Gestaltung komplexer Formen z.B. aus Beton soll ohne Spezialwissen in Umformtechnik möglich gemacht werden.

Abb. 2.11: BIM von Elbphilharmonie Hamburg

Bildquelle: vdi nachrichten - „Die Angst des Ingenieurs vorm Dirigenten“



3. Methodik

3.1. Umformtechnik-Theorie

Die Recherche der richtigen Umformmethode für diese Arbeit begann mit dem Materialverhalten beim Umformen. Die Thematik Umformung von Metallen ist ein Thema, das eng mit den Materialeigenschaften und den auftretenden Spannungen des Materials verbunden ist. Bei der Umformung jeglicher Materialien entstehen Zug- und Druckkräfte. Abhängig von der Querschnittgeometrie müssen also in den Bereichen von Zug und Druckspannungen mehr oder weniger Material verdichtet bzw. gedehnt werden.

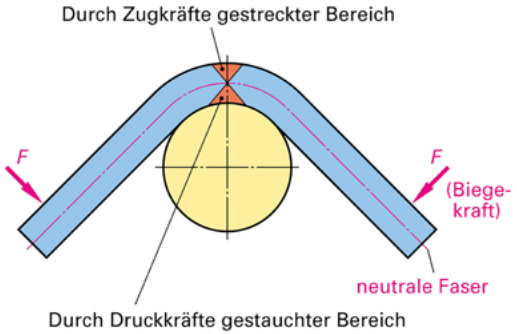


Abb. 3.1: Belastungen im Bereich der Umformung.
Quelle: Bartenschlager, Fachkunde Metall.

Beim Betrachten des Spannungs-Dehnungsdiagramms von Stahl kann man somit feststellen, in welchem Bereich eine bleibende Umformung bzw. eine Biegung stattfindet, die zum Materialversagen führt.

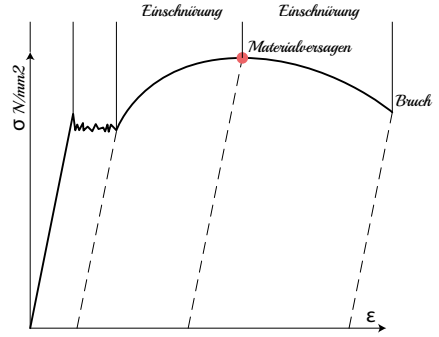


Abb. 3.2: Spannungs-Dehnungsdiagramm von Stahl mit verschiedenen Belastungsbereichen.

Vor allem bei Bewehrungsstahl führt die Umformung über den plastischen Bereich hinaus zu einem Materialversagen. Der Stahl kann hier keine weiteren Kräfte aufnehmen und führt zu einer Längenveränderung bei gleichbleibender Belastung. In einem bewehrten Betonbauteil wird dies ein fatales Versagen des gesamten Bauteils zur Folge haben. Unter Berücksichtigung dieses Verhaltens muss ein Mindestradius eingehalten werden. Der häufigste Fall einer Umformung von Bewehrungsstäben ist das Erstellen von Winkeln. Da das Knicken von Bewehrung zu einem Materialversagen führen würde, ist gerade hier der Mindestradius zu beachten. Dieser ist abhängig von der Legierung und Stahlgüte, sowie vom Durchmesser des Metallstabes.

Tabelle 2: Mindestbiegeradien

Werkstoff	Blech	Rohr
Stahl	1 x Blechdicke	1,5 x Rohr-Ø
Kupfer	1,5 x Blechdicke	1,5 x Rohr-Ø
Aluminium	2 x Blechdicke	2,5 x Rohr-Ø
Cu-Zn-Legierung	2,5 x Blechdicke	2 x Rohr-Ø

Abb. 3.3: Tabelle mit überschlägigem Mindestradius verschiedener Materialien
Quelle: Bartenschlager, Fachkunde Metall.

Um zum Beispiel einen 90° Winkel mit einem Stahlstab zu erreichen, ist ein mit einem Sicherheitsfaktor multiplizierter Mindestradius über Roll- oder Biegestempelumformung herzustellen. Um den exakten Winkel von 90° zu erhalten, muss jedoch die Rückfederung des Materials berücksichtigt und der Überbiegungswinkel gebogen werden, sodass das Werkstück nach der Rückfederung

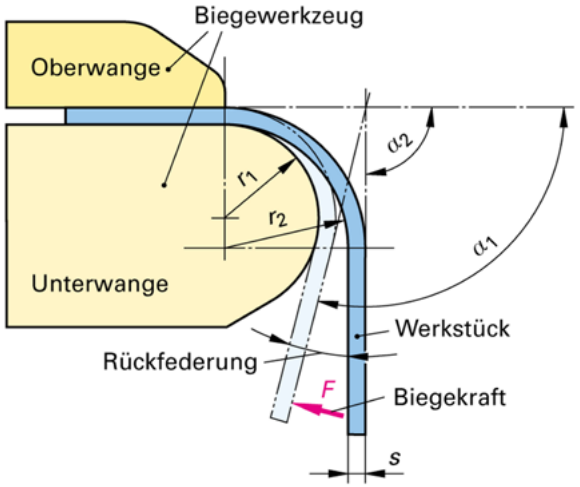


Abb. 3.4: Überbug beim Biegestempelumformen
Quelle: Bartenschlager, Fachkunde Metall.

einen 90° Winkel aufweist. Diese Methode wird auch bei der Biegung von Blechen verwendet. Hierbei wird mithilfe von Tabellen die Überbiegung berechnet. In dieser Arbeit wird die Überbiegung mithilfe der Materialtabelle (siehe Kapitel 3.6.3) kalkuliert und bei der Umformung ein kleinerer Radius produziert, sodass sich nach der Rückfederung der gewünschte Radius ergibt.

3.1.1. Stabbelastungen

Die Abbildung 3.5 zeigt links die Krümmungen eines Stabes beim Verschieben des Auflagers. Die Krümmungen sind äquivalent mit den Spannungen in jedem Bereich des Stabes. Aus diesem Diagramm kann man erkennen, dass die Spannungen sich zur Mitte hin erhöhen und somit ungleichmäßig verteilt sind. Bei der Entwicklung der Methodik für die Umformung spielte diese Beobachtung eine wichtige Rolle. Um eine möglichst gleichmäßige Verteilung der Belastungen im Material zu erreichen muss die gewünschte Form durch Umformungen in Teilbereichen erreicht werden. Als logischer Schritt erschien hier also die Rollumformung als die bestmögliche Methode, um einen konstanten Radius und eine dementsprechende gleichmäßige Spannungsverteilung zu erreichen.

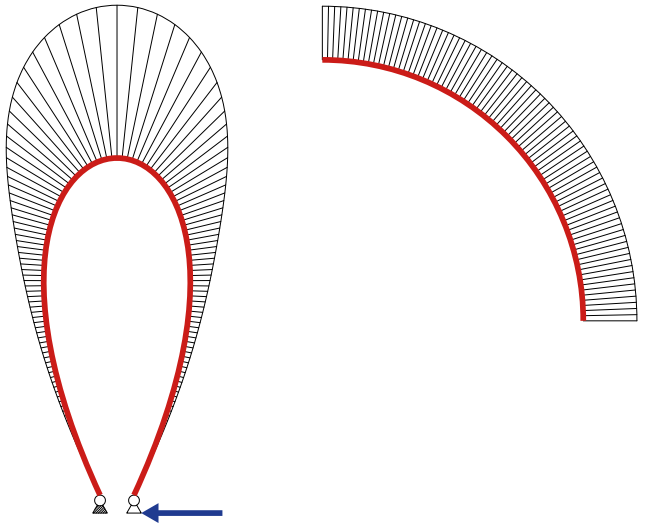


Abb. 3.5: Vergleich Krümmung/Belastung von Stäben. Links: Verformung durch ziehen an den Enden; Rechts: Verformung mit konstantem Radius

3.1.2. Umformmethode

Aus den Recherchen der Theorie zum Umformen von Stahl und den Beobachtungen aus Versuchen ist eine Methode konzipiert worden, welche es möglich macht, die Umformung unter bestmöglicher Verteilung der Belastung zu realisieren. Es soll mit dieser auch eine präzise Kontrolle der Form möglich sein. Zum Vergleich ist Abb. 3.5 eine selbstbildende Form, also nicht kontrolliert. Die entworfene Methode wurde anhand von Computer-Simulation getestet und eine mathematische Berechnung für die Kontrolle erstellt. Die Idee ist es, durch einen konstanten Vorschub des Materials, welches unter konstanter seitlicher Krafteinwirkung am Ende durch gerade angeordnete Düsen verläuft, um eine gleichmäßige Biegung zu erzeugen. Dies würde einen konstanten Radius erzeugen. Durch die Kontrolle der Krafteinwirkung durch Verschiebung des Rollwiderstandes kann so auch der Radius kontrolliert werden. Mit der Möglichkeit den Rollwiderstand rund um den Stab anzuordnen, ist es somit möglich das Material frei im Raum zu biegen und Freiformkurven zu erstellen.

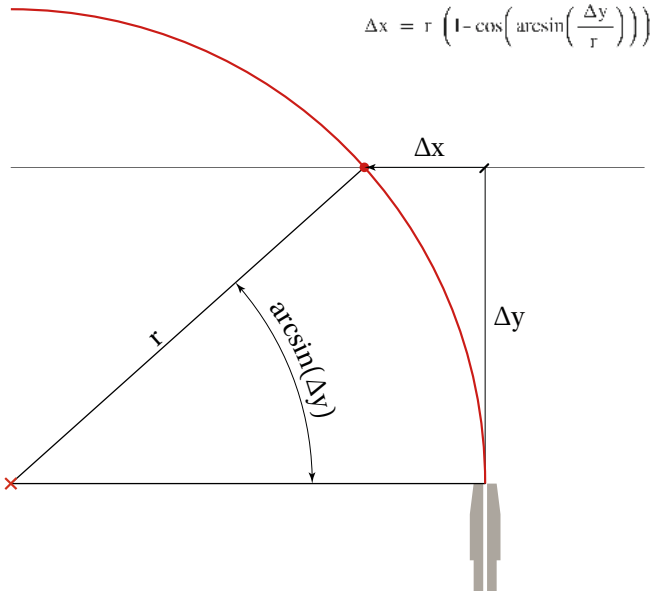


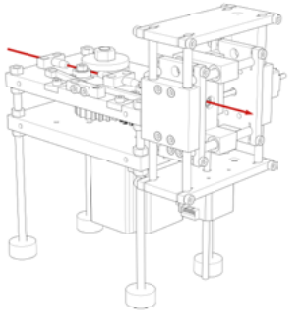
Abb. 3.6: Berechnung der Verschiebung um Delta X für einen gewünschten Radius.

3.2. Konstruktionsplanung

Die konzipierte Umformmethode wird bei der Konstruktionsplanung durch mechanische Elemente gebildet. Der gesamte Prototyp wurde mit der CAD Software Rhinoceros geplant und entworfen. Obwohl es sich bei dieser Software nicht um eine typische Maschinenbau-Software handelt, ermöglicht diese 3D Software jedoch durch eine Parametrisierung (Zusatz Grasshopper3D) eine Simulation der Mechanik, die auf komplexeren Formeln basiert. Aus diesem Grund konnte auch eine Planung in 3D mit anschließender Simulation der komplexen Mechanik, sowie der in 3.1 beschriebenen Umformtechnik vor der Fertigung in der Software getestet werden.

Die Planung des Mechanismus für den Prototyp der Umformmaschine begann vorwiegend unter Berücksichtigung der anzunehmenden Kräfte, die bei der Rollumformung auftreten. Ein weiteres Hauptkriterium war die Größe bzw. Kompaktheit des Geräts in Hinsicht auf die Transportfähigkeit und dem Gewicht. Als erstes Konzept wurde die Mechanik der Umlenkung der Metallstäbe betrachtet und aufgrund der nötigen Präzision, die aus der Analyse der Theorie hervorging, ein gut steuerbarer Mechanismus entworfen. In der zweiten Stufe der Planung wurden die technischen Lösungsmöglichkeiten des Vorschubs betrachtet und als letzten Schritt versucht, diese mechanischen Lösungen in eine möglichst kompakte und gewichteinsparende Gesamtkonstruktion zu bringen. Um die gewählte Obergrenze für das Gewicht von 20kg nicht zu überschreiten, ist als Material Aluminium gewählt worden, nur hoch belastete Teile wie Zahnräder, gelagerte Wellen und Spannprismen werden in Stahl ausgeführt.

Die Berücksichtigung dieser Faktoren in der Planung der Konstruktion ermöglicht überwiegend gleichbleibende Biegeergebnisse und aussagekräftige Resultate in der Messung und der anschließenden Berechnung.



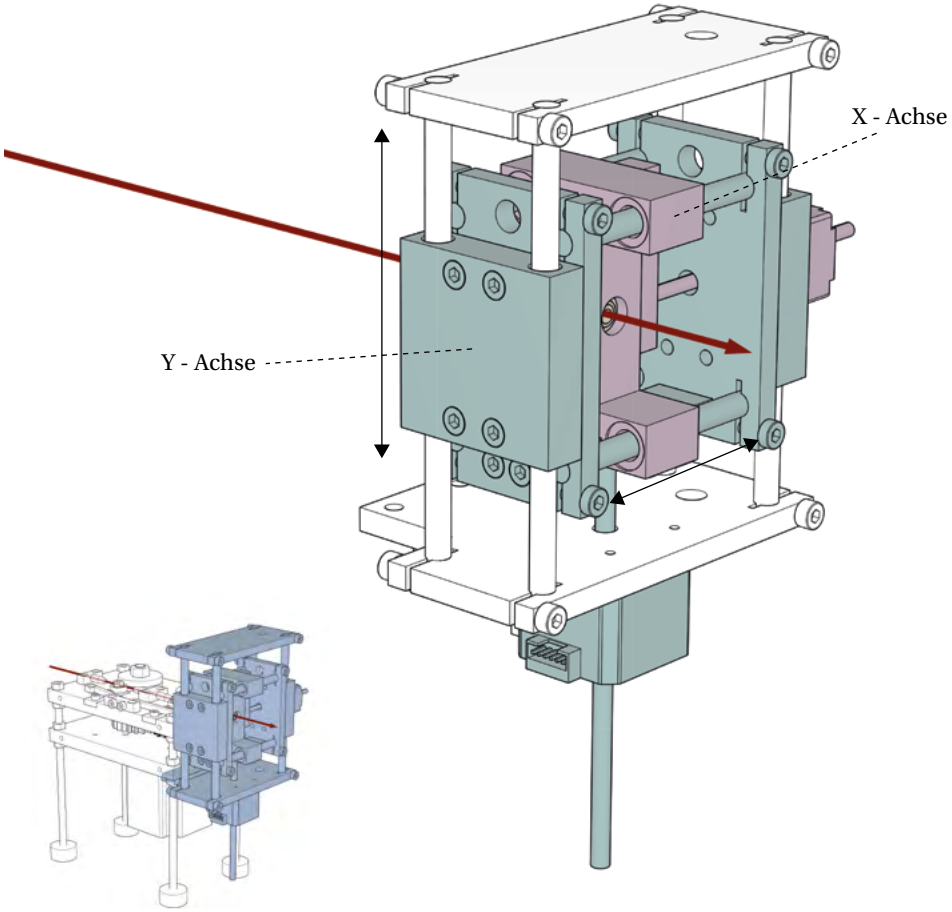
3.2.1. Umlenkkopf

Die Planung der Umlenkmechanik wurde mit dem zentralen Element der Drahtdurchführung gestartet. Diese muss mit einer möglichst niedrigen Reibung funktionieren, um andere Kräfte, außer jene die für die Umformung benötigt werden, auszuschließen. Nach Evaluierung verschiedener Mechanismen kam nur die technische Lösung mittels einer kurzen Düse in Frage. Das zu biegende Material wird durch eine ausreichend lange Düse geführt, welche sich in einem Kugelgelenklager bewegen kann. Durch diese Lagerung der Düse kann ein Freiheitsgrad f=4 bei der Umformung bewerkstelligt werden. Der Freiheitsgrad bezieht sich auf die Rotationsfreiheit (drei Achsen) und der Verschiebung (eine Achse) des Drahtes während der Umformung. Dieser garantiert eine Biegung, die nur durch Kraftweinwirkung mittels Translation auf zwei Achsen (nachfolgende Bezeichnung X und Y-Achse) erfolgt. Als Material für die Düse wurde der weichere Werkstoff Bronze gewählt, welcher diese Düse zum Verschleißteil macht und somit das Lager vor ungewollter Abnutzung schützt.

Der Umlenkkopf als zentrales Element der Umformung soll sich möglichst präzise auf zwei Achsen bewegen können (Freiheitsgrad f=2). Die zwei Achsrichtungen ermöglichen die freie Umformung des Drahtes im Raum. Hier wurde eine technische Lösung gefunden, die es ermöglicht, das Kugelgelenk mit darin gelagerter Düse auf acht linearen Lagern (vier pro Achsrichtung) zu bewegen. Die Achsrichtung für die vertikale Translation (Y-Achse) wird durch vier zylinderförmige Stäbe gebildet. Auf diesen Rundprofilen kann sich über Gleitlager der Wagen mit der darauf befindlichen X-Achse (horizontale Translation) bewegen. Auch dieser wurde unter Gesichtspunkten der Belastungsansprüche mit vier Rundprofil-Stäben und Gleitlagern ausgeführt. Auf der X-Achse wird das vorhin beschriebene Kugelgelenk mit der Düse geführt. Dieses System erlaubt eine präzise Bewegung auf zwei Achsen unter großer Kraftbeanspruchung, welche durch die Kalt-Umformung zu erwarten ist.

Für die elektromechanische Bewegung der beiden Achsen wurde eine Lösung mit sogenannten „non-captive“ Linearaktuatoren gewählt. Diese Linearaktuatoren bestehen aus 2-phasigen Hybrid-Schrittmotoren mit einer Hohlwellen-Mutter und einer Spindel. Durch die Schrittmotortechnik kann eine Verschiebung der Achse mittels Impulssteuerung über einen Mikrocontroller vorgenommen werden (siehe Kapitel 3.2.4). Die Spindel kann sich mit Drehung der Hohlwellen-Mutter durch den Schrittmotor hindurchbewegen.

Dies ermöglicht die Plazierung der Spindel genau im Achszentrum für eine optimale Kraftübertragung. Die somit gleichmäßige Verteilung der Kräfte auf die vier linearen Lager vermeidet das Auftreten von Querbelastungen und reduziert das Risiko für ein Versagen der Achsbewegung durch Lagereinschnürung oder Reibungsblockade. Die unter größerer Belastung arbeitende Y-Achse, die den gesamten Wagen für die X-Achse mitsamt Kugelgelenk und Düse bewegen muss, wurde diese Schrittmotor/Spindel Kombination in einer größeren und stabileren Ausführung gewählt.

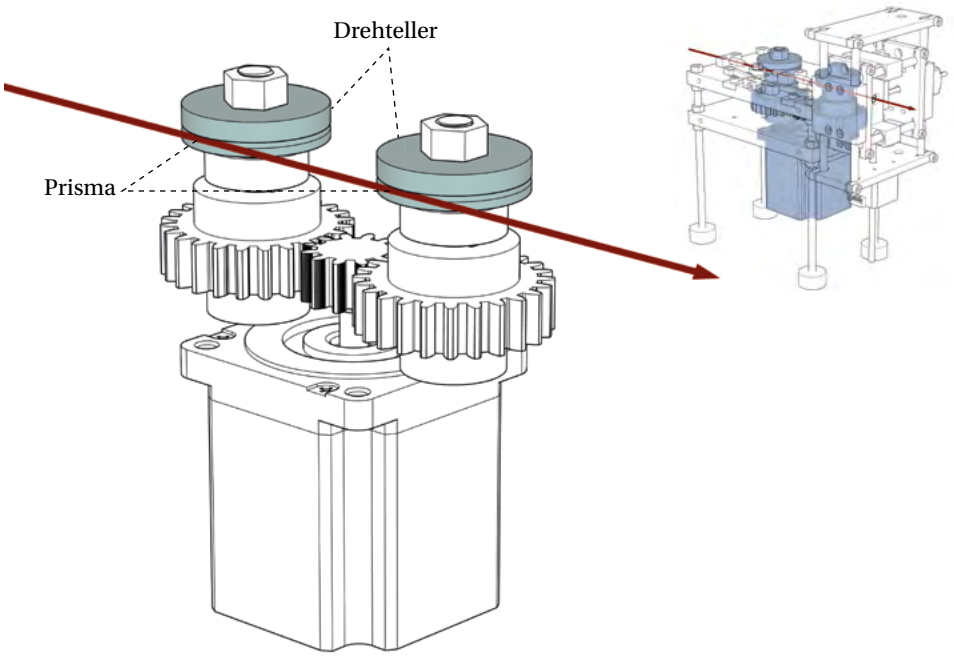


3.2.2. Vorschub

Nach dem Entwurf des Umlenkkopfes mit der Achsmechanik für die X und Y Verschiebung wurde die Funktion und Dimensionierung des Vorschubs (nachfolgend Z-Achse bezeichnet) geplant. Der Ausgangsparameter dieser Planung war die Stärke des Motors für den Vortrieb des Biegematerials. Für eine sehr präzise Ansteuerung ist auch hier eine Lösung mittels Schrittmotor gewählt worden. Nach Evaluierung verschiedener Modelle und Spezifikationen kommt ein Schrittmotor der Baugröße NEMA 34 zum Einsatz, welcher ein Haltemoment von 6,0 Nm bei optimaler Stromzufuhr erreicht. Da das Haltemoment der obere Richtwert für das Drehmoment (Moment im bewegten Zustand) eines Schrittmotors ist, wurde ein möglichst hohes Drehmoment als Hauptkriterium gewählt. Die Baugröße und das Gewicht dieses Motors sind die größten Einflussfaktoren auf die Gesamtmaße und Gewicht des Prototyps. Um nicht eine noch größere Baureihe verwenden zu müssen und die Präzision der elektronischen Ansteuerung noch einmal zu erhöhen, wurde eine Übersetzung von 2:1 mithilfe von Zahnrädern eingeplant.

3.2.3. Einspannmechanik

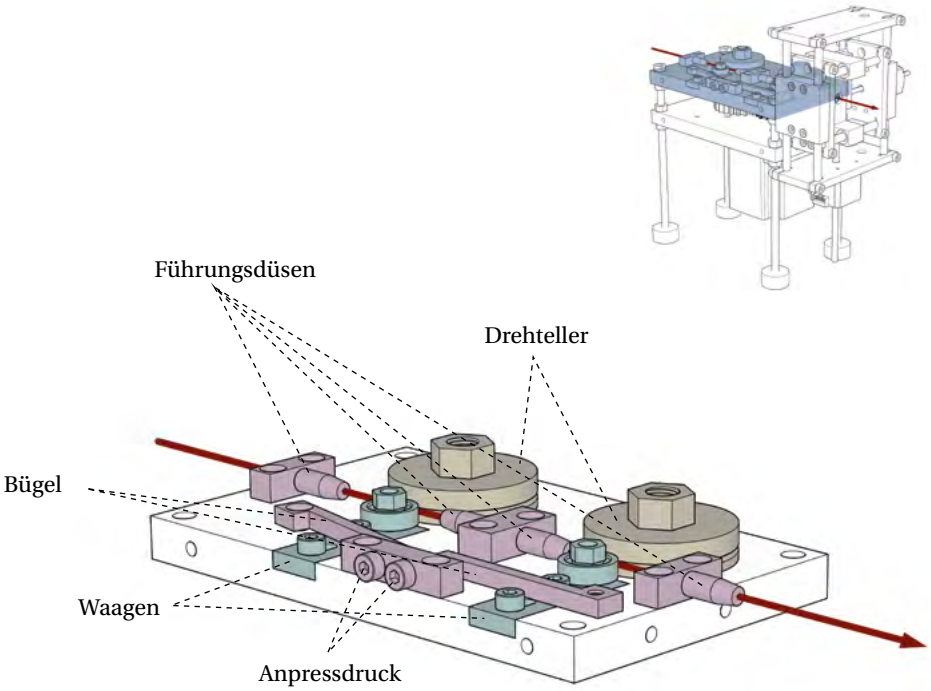
Aufgrund der benötigten hohen Schubkraft für die Förderung des Biegematerials, wurde eine Lösung gewählt, die es erlaubt unter möglichst geringer Materialbeeinträchtigung (Quetschung des Querschnitts) den Vorschub zu realisieren. Auf zwei kreisrunden Tellern wird mittels einem Prisma der Draht fest eingespannt und kann somit nur durch Drehung der Teller nach vorne bzw. nach hinten bewegt werden. Das Prisma wird halbrund und mit dem entsprechenden Durchmesser des Biegematerials ausgeführt. Der Draht, der die Drehteller nur tangiert, kann somit mit möglichst hoher Reibungskraft nach vorne bewegt werden. Die Anzahl der Teller wurde auf zwei Stück beschränkt, um Gewicht und Gesamtgröße des Prototyps gering zu halten, aber dennoch die Geradentreue des Vorschubs sicherzustellen. Der Durchmesser der Teller von 40mm wurde berechnet, um zusammen mit den Zahnrädern eine Übersetzung von 2:1 zu erreichen. Die Stärke der zylinderförmigen Drehteller wurde so gewählt, dass das Anbringen von zwei Prismen für die verschiedenen Drahtstärken möglich ist.



Vor und nach dem Drehteller wurden Führungsdüsen angebracht, um einen geraden Vorschub zu gewährleisten und ein seitliches Ausweichen nach dem Eulerschen Knickfall 3 zu unterbinden. Auch diese Führungsdüsen werden als Verschleißteile betrachtet, um die Haltevorrichtungen für die Düsen und anliegende Teile vor Abnutzung zu schützen. Es sind drei Halteböcke für insgesamt 4 Führungsdüsen in der Konstruktion geplant worden, um jeweils vor und nach dem Drehteller eine gerade Drahtführung zu gewährleisten. Führungsdüsen und die Einspannprismen auf den Drehtellern wurden genauestens aufeinander abgestimmt und je nach Drahtdurchmesser auf ihre axiale Ausrichtung angepasst. Für jeden Drahtdurchmesser kommen somit ein Satz von vier Führungsdüsen und jeweils ein Prisma auf den zwei Drehtellern zum Einsatz. Beim Wechsel vom Durchmesser des Biegematerials müssen die entsprechenden Führungsdüsen eingesetzt werden und durch ein Umdrehen der Drehteller das passende Prisma gewählt werden.

Um den zu biegenden Draht im Prisma des Drehtellers zu halten werden zwei Kugellager des Typs 625 ZZ verwendet. Über einen Wagen, der die Kugellager auf einer Achse lotrecht zum Drehteller führt, können diese das Biegematerial an das Prisma drücken.

Die Anpresskraft kann über drehbare Bügel angepasst werden und über zwei an der Seite zugängliche Schrauben justiert werden. Somit kann die Haftreibungskraft zwischen dem Biegematerial und dem Prisma für den Vorschub eingestellt werden.

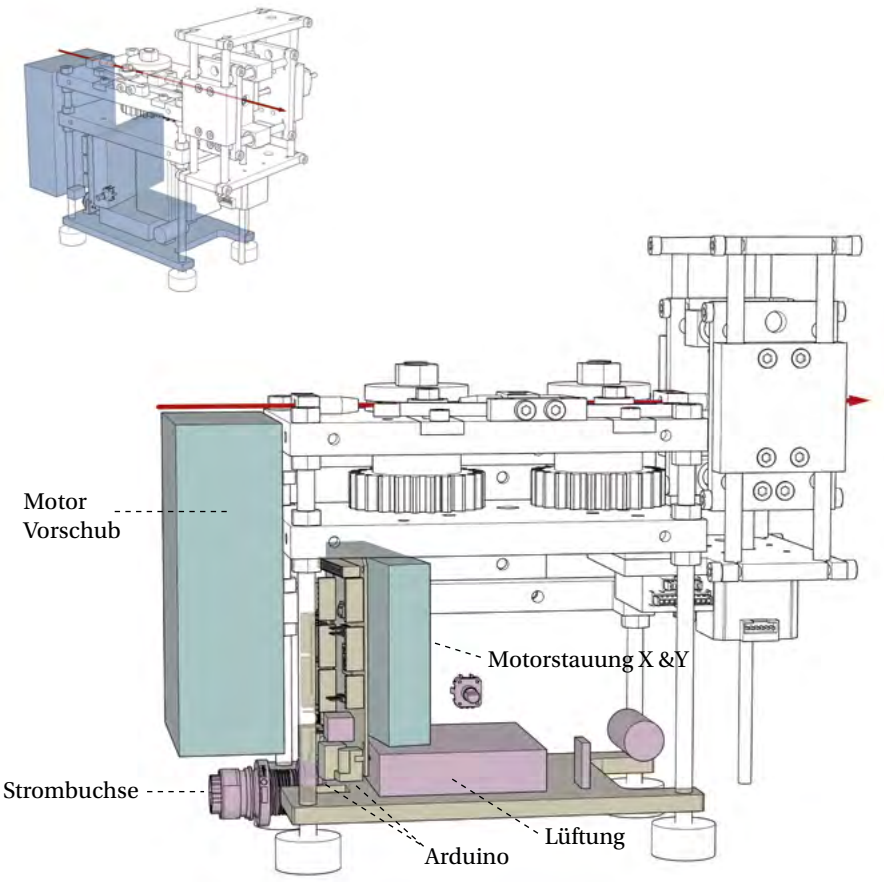


3.2.4. Elektronik

Nachdem der mechanische Teil gelöst wurde, konnten die entsprechenden elektronischen Komponenten für das Funktionieren des Prototypen recherchiert werden. Um einen möglichst hohen Wirkungsgrad der Motoren zu erzielen, wurden Schrittmotoren-Steuergeräte, sogenannte „microstep driver“ gewählt. Diese sind in verschiedenen Ausführen und Betriebsspannungen erhältlich und können mit Schritt und Richtungsimpulsen gesteuert werden. Für den großen Schrittmotor an der Vorschubmechanik wurde eine leistungsstarke Steuerung gewählt, die aufgrund ihrer Größe an der Außenseite des Gehäuses angebracht werden muss. Die beiden Motoren für die X und Y-Achse werden durch zwei kleine Steuerungen gespeist, welche platzsparend im Gehäuse untergebracht werden

können. Für die Bedienung des Prototypen wurde ein 2,8“ großer Monitor mit Touch Funktionalität, ähnlich einem Smartphone verwendet. Ein beleuchteter Taster dient als Ein-/Ausschaltknopf und gibt zudem Rückmeldung über den Status der Maschine (Ein/Aus/Standby). Neben der Touch-Funktion des Displays wird auch ein Drehencoder für die Bedienung eingesetzt. Dieser erleichtert das Navigieren in den Befehlsdateien und die Auswahl von Parametern des Prototyps. Für die Kommunikation mit dem PC ist eine USB Buchse vom Typ B verwendet, die typisch für diese Geräteklasse (07h Drucker) ist. Um eine genügende Kühlung des großen Schrittmotors zu gewährleisten und damit auch die im Gehäuse befindliche Elektronik zu schützen, wird ein 12V betriebener Lüfter an der Unterseite des Gehäuses angebracht. Als wichtiges Sicherheitselement werden bei der X und Y-Achse jeweils zwei mechanische Endschalter angebracht. Diese unterbrechen beim Betätigen den Stromkreis und verhindern so ein Überdrehen der Motoren, wenn das Ende der Achsbewegung (beide Richtungen) erreicht ist. So kann sichergestellt werden, dass bei Softwarefehlern während der Entwicklungsphase die teure Hardware nicht zerstört wird.

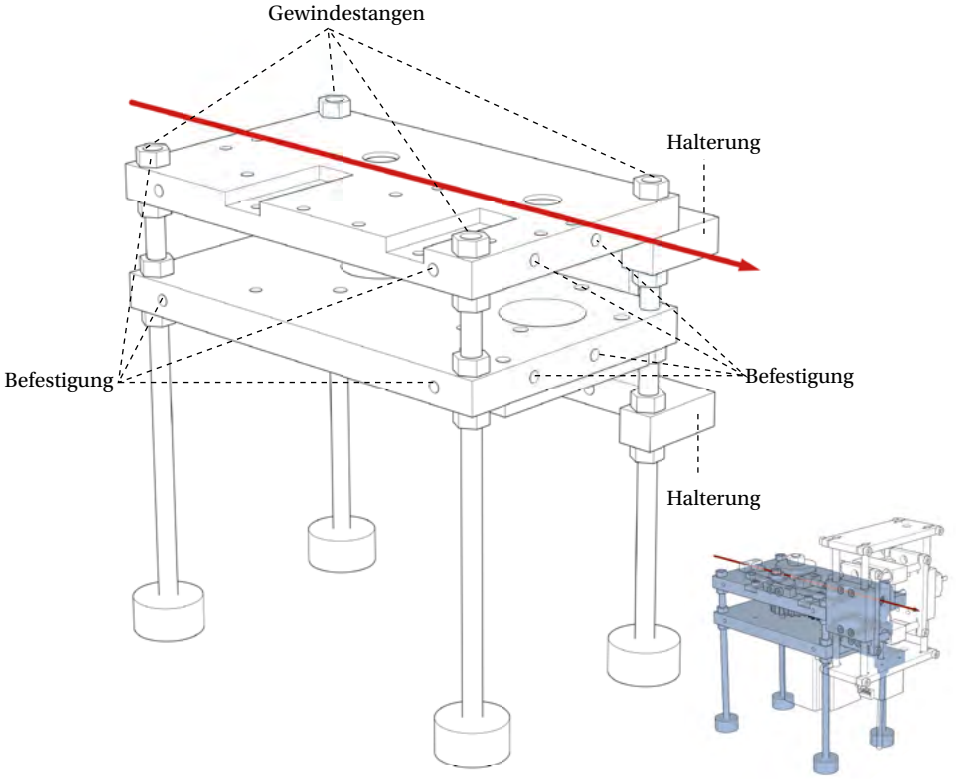
Das Herzstück und zentrale Verknüpfung der Elektronik bilden zwei Mikrocontroller der Arduino Familie. Ein Arduino vom Typ UNO R3 das mit den Steuergeräten der Motoren verbunden ist und ein Arduino Mega 2560 R3, welches für die Bereitstellung der Benutzeroberfläche auf dem Display und die Datenübertragungen zwischen PC, Maschine und dem Arduino Uno (Motorsteuerung) zuständig ist. Beide Steuergeräte für die X und Y-Achse, sowie die zwei Mikrocontroller werden im Gehäuse hinter dem NEMA 34 Motor für den Vorschub angebracht.



3.2.5. *Gehäuse*

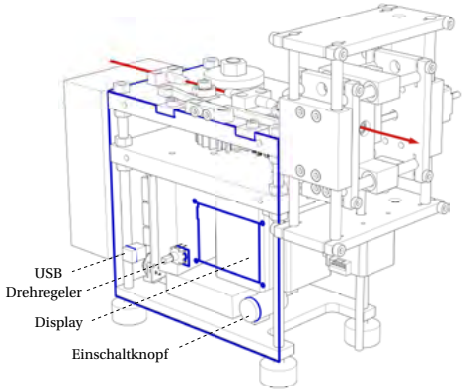
Im letzten Schritt der Planung wurde versucht alle bisher beschriebenen Komponenten möglichst platzsparend anzubringen. In diversen Iterationen wurden Abstände und Maße der in Kapitel 3.2.1 bis 3.2.3 beschriebenen Komponenten verändert und optimiert, um eine möglichst kompakte Bauform zu erhalten. So wurde etwa die X und Y-Achse außerhalb der Belastungszone (Vorschub) angeordnet und mit einer einfachen Verschraubung an das Gerät angehängt. Die Anordnung der Drehteller und des Einspannmechanismus wurde so gewählt, dass diese mit den berechneten Zahnrädern

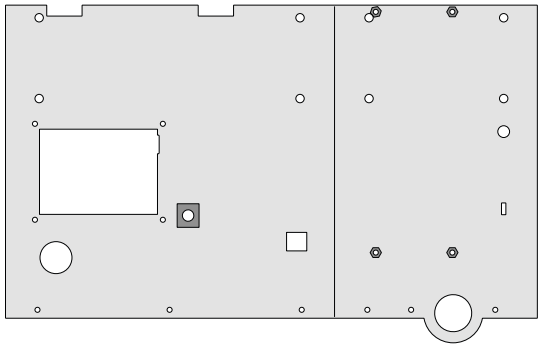
(drei Stück) und dem großen Schrittmotor passend zusammenlaufen. Aufgrund der zu erwartenden Kräfte, die beim Vorschub auftreten, wurden zwei massive Bauteile für das Anbringen der kugelgelagerten Wellen geplant. Diese Wellen sind fest mit den Zahnrädern verbunden, übertragen die Drehbewegung auf die Drehteller und sind zwischen den beiden massiven Bauteilen mittels Lagersitzen befestigt. Um die gesamte Konstruktion aufzuständern und eine Torsion um die Wellenachsen zu unterbinden wurde eine Lösung mit Stäben mit ausreichendem Durchmesser gewählt. Diese sollen später als Fein-Gewindestangen ausgeführt werden, um ein einfaches Justieren zu ermöglichen. An der Rückseite (Display = Vorderseite) wird eine massive Halterung angebracht, welche das Gesamtgewicht der Maschine aufnehmen kann und eine eventuelle Montage auf einem Industrieroboter o.ä. möglich macht. Seitlich an den massiven Bauteilen sind jeweils sechs Gewindebohrungen für die Befestigung der Abdeckung vorgesehen.



3.2.6. *Abdeckung*

Zur Unterbringung von Bedienelementen (siehe 3.2.4) und Verkabelung, sowie als Schutzvorrichtung der Elektronikkomponenten wurde eine Abdeckung geplant. Diese kann an den massiven Rahmen der Konstruktion befestigt werden und hüllt den gesamten Bereich des Vorschubs ein. In der Planung wurde eine Abwicklung der Außenkanten der massiven Teile vorgenommen und in 2D die Bedienelemente platziert. In wiederholender Rückprüfung mit dem 3D Modell wurde so eine Anordnung gewählt, die möglichst viel Freiraum für die spätere Verkabelung bietet. Die Abdeckung soll in Acrylglas ausgeführt werden, da dieses durch Laserschneidetechnik einfach zu bearbeiten ist und durch die Transparenz einen Einblick auf alle innen befindlichen Elektronikkomponenten zulässt. Die Überwachung dieser Komponenten ist bei einem Prototyp von großer Wichtigkeit, um bei eventueller Überlastung oder Fehlfunktion sofort Maßnahmen einleiten zu können.





3.3. Simulation

Nach Fertigstellung und während der Planung wurde die Konstruktion immer wieder mittels Simulation in der Software überprüft. Mithilfe einer Parametrisierung der Geometriedaten konnten so alle Bewegungsabläufe getestet und der Umformmechanismus evaluiert werden. Mithilfe der Funktionen, die die verwendete Software Rhinoceros mit dem Zusatz-Modul Grasshopper für die Parametrisierung bietet, konnten so Auswirkungen der Veränderungen an den Komponenten auf die Effizienz der Umformtechnik beobachtet werden.

Die Parametrisierung des 3D-Modells wurde mit den in Kapitel 3.1 beschriebenen Formeln bewerkstelligt. Dabei wird die X-Achse mit der kugelgelagerten Düse parametrisch so verschoben, dass aus dem errechneten Abstand zur letzten Führungsdüse der Radius der Umformung berechnet werden kann. Die Geometriedaten des 3D-Modells werden hier verwendet, um die relevanten Koordinaten für diese Berechnung zu erhalten. Bei jeder Veränderung der Geometrie wird so automatisch eine Neuberechnung gestartet, was sich bei der Planung als sehr hilfreiche Funktion erwies.

Der Simulationsknoten, in welchem die Berechnung stattfindet, erhält als Eingabeparameter den gewünschten Radius (r), die Richtung (Vorzeichen Radius + Winkel) des Radius und den Abstand der letzen Führungsdüse zur Düse im Kugelgelenk (Delta Z). Mithilfe der Formel (Erklärung 3.1).

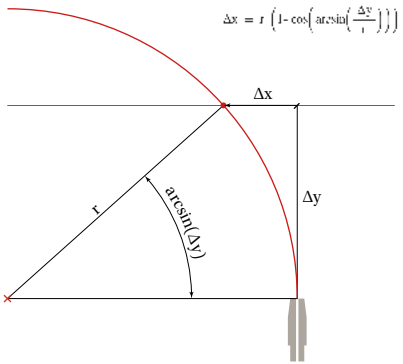


Abb. 3.6: Berechnung der Verschiebung um Delta X für einen gewünschten Radius.

kann so die Position der X und Y-Achse berechnet werden (Delta X und Delta Y). Jede Verschiebung dieser Achsen wird als Relativkoordinate vom Schnittpunkt der Achsen mit der Vorschubachse gesehen. Wenn also Delta X und Delta Y gleich 0 gesetzt werden ,ergibt das einen unendlich großen Radius, also keine Umformung des Drahtes. Bei Delta X ungleich 0 entsteht eine entsprechende Umformung in der Ebene. Wird auch Delta Y ungleich 0, ergibt sich eine helix-förmige Umformung des Biegematerials. Durch Veränderung des Vorzeichens des Radius wird die Richtung der Biegung geändert. Ein negativer Radius entspricht einer linksgerichteten Biegung in Vorschubrichtung, ein positiver Radius folglich eine Biegung nach rechts in Richtung der Drahtförderung. Durch die Angabe der vertikalen Verschiebung kann eine Helix simuliert werden, die einen Radius entsprechend der Position der X-Achse aufweist und eine Steigung (engl. pitch) entsprechend der Y-Position annimmt.

Nach Anpassung aller Komponenten in der Planung mithilfe der Simulationsdurchgänge konnte die Organisation der Fertigung angegangen werden. Die Simulation wird sich erst später in der Entwicklung der System-Architektur als geeignetes Hilfsmittel erweisen.

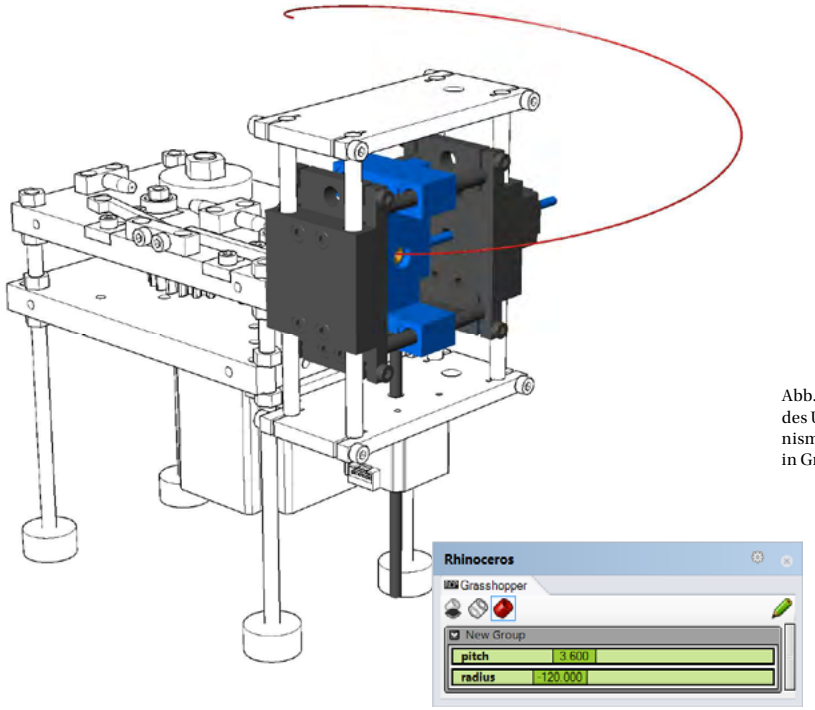


Abb. 3.7: Simulation des Umformmechanismus am 3D-Modell in Grasshopper

3.4. Fertigung

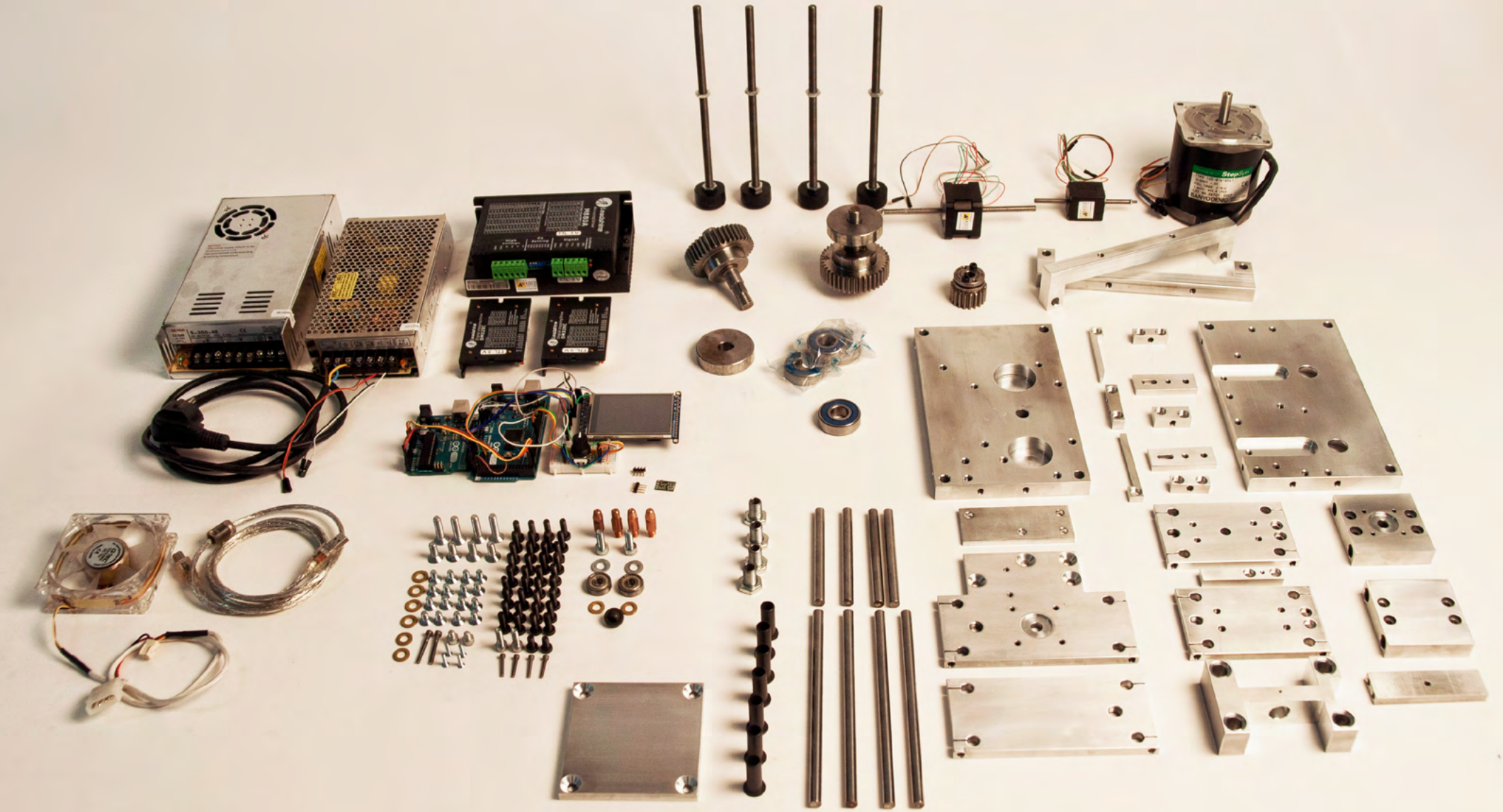


Abb. 3.7: Übersicht aller gefertigten Teile und verwendeten Komponenten.

Schon während der Planung wurden immer wieder Recherchen im Bezug auf die Fertigung getätigt. Es wurde versucht so viele Standardkomponenten wie möglich in der Konstruktion zu verwenden. Teile, die eine aufwendige Fertigung benötigen, wurden so gewählt, dass diese von Händlern bezogen werden konnten. Die restlichen Komponenten mussten selbst gefertigt und der dementsprechende Aufwand abgeschätzt werden. Obwohl es sich bei den verwendeten Produkten um keine Spezialanfertigungen handelt gestaltete sich der Bezug dieser Waren nicht einfach. Über diverse Händler mussten die Komponenten bestellt werden. Die Linearaktuatoren und darauf abgestimmte Steuergeräte wurden beim deutschen Hersteller für elektrische Antriebe „Koco Motion“ bestellt. Zahnräder vom Fachhändler Norelem. Die Schaltnetzteile für die externe Stromzufuhr wurde bei CNC-Profi bezogen. Für die Verbindung vom großen Motor mit dem Zahnrad wurde ein mechanisches Spannelement vom Typ ETP-Mini von Lenze Selection bezogen. Die Mikrocontroller, das Touch-Display und diverse Elektronikkomponenten wurden bei Flikto in Köln bestellt. Endschalter, Ein-/Ausschalter, Verkabelung, Stecker und diverse andere Elektronikkomponenten wurden beim Händler RS-Components gekauft. Die speziellen Rundsteckverbinder aus der Militärtechnik-Klasse wurden beim polnischen Händler Transfer Multisort Elektronik bestellt. Diverse Kleinteile und Metallwaren konnten im lokalen Fachhandel gekauft werden.

3.4.1. *Aluminiumteile*

Durch die freundliche Unterstützung des Südtiroler Betriebes Windegger Maschinen GmbH konnten die geplanten Aluminiumteile aus Rohmaterial gefertigt werden. Der Betrieb stellte einen Mitarbeiter und eine 5-Achs CNC-Fräse für Metallbearbeitung zur Verfügung. In insgesamt 3 Arbeitstagen wurde das Rohmaterial zugeschnitten und vorbereitet, die aus den Planungsdaten erhobenen Daten auf der Computer-Steuerung der Maschine eingegeben und Stahlteile, wie Welle und Drehteller auf der digitalen Drehbank gefertigt. Für die Fertigung mit der CNC-Fräse wurden vom 3D-Modell Maschinenbaupläne erstellt, die alle Bemaßungen, Gewindebezeichnungen, Toleranzen und Frästaschen angeben. Daraufhin konnten die Daten in die Steuerung der Fräse eingegeben werden. Beim Erstellen der Steuerungsdateien wurden alle Werkzeugwechsel der Fräsmaschine einprogrammiert, sodass in einem Durchgang alle Fertigungsschritte am Ausgangsmaterial vorgenommen

werden konnten. Im Fachjargon wird diese Vorgangsweise als Fertigung unter einer Einspannung bezeichnet und verhindert Toleranzen, die bei einer erneuten Einspannung auftreten. Vor dem Frässtart wird das Ausgangsmaterial auf die dritte Kommastelle vermessen um eine sehr hohe Bearbeitungsgenauigkeit zu erreichen. Sobald der Fräsvorgang gestartet wird, kann das Kühlungsmittel manuell justiert werden. Dieses kühlt die gefrästen Stellen während des Vorgangs und verhindert so einen Materialverzug und dementsprechende Abweichungen des Werkstücks. Nach vollendetem Fräsvorgang werden die Werkstücke optisch überprüft und nachgemessen. In einem letzten Schritt werden von Hand alle scharfen Kanten, Auffaserungen und Splitter mit einem sogenannten Entgratwerkzeug entfernt.

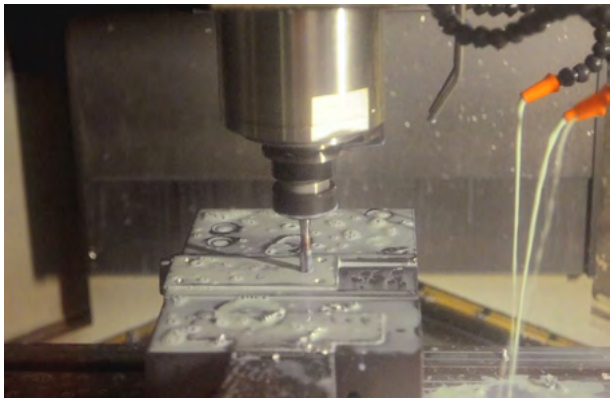


Abb. 3.8: CNC-Fräsung des Oberteils für die Y-Achse

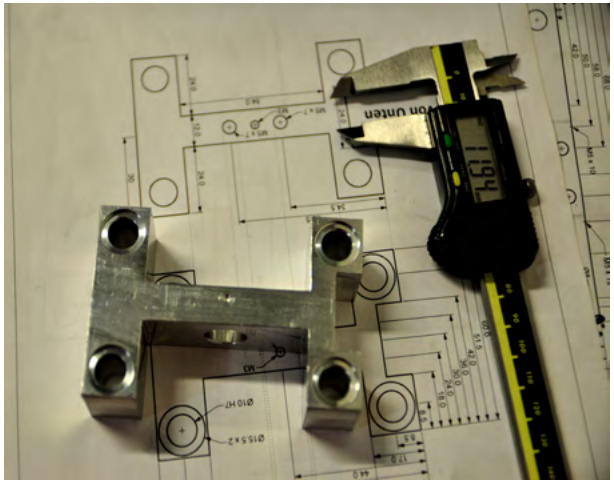


Abb. 3.9: Überprüfung des gefrästen Teiles mithilfe des 1:1 Plans und einer digitalen Schieblehre

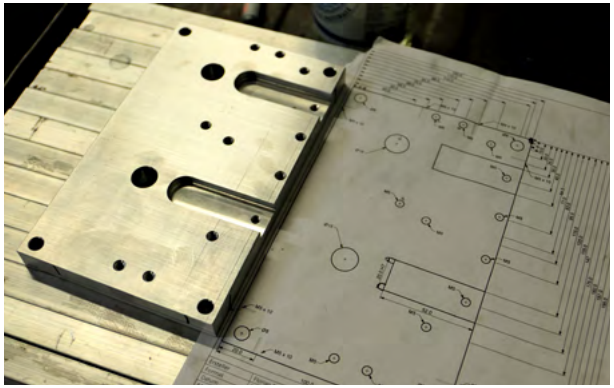


Abb. 3.10: Gegenüberstellung des fertigen Werkstücks und dem Plan.

Abb. 3.11: Eingabe der Koordinaten und Befehle für die CNC anhand der Pläne.

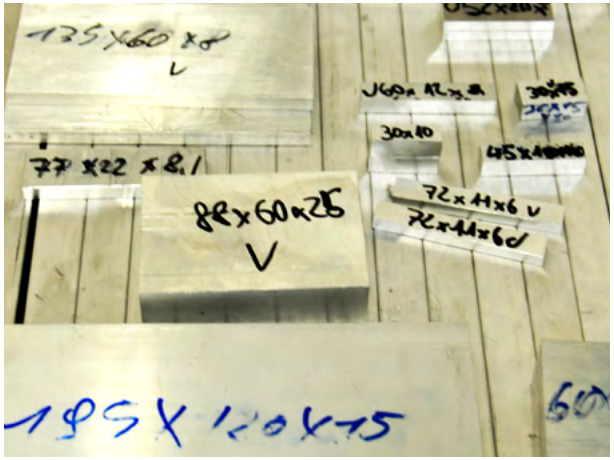


Abb. 3.12: Vorbereitung des Ausgangswerkstoffs Aluminium.

Abb. 3.13: Eloxierbecken der Firma S&W Eloxal in Kirchbichl

Abb. 3.14: Aufsteigender Wasserstoff beim Eloxiervorgang

Abb. 3.15: Aufgehängte und fertig eloxierte Teile auf einem Gerüst aus nichtleitendem Titan.

3.4.2. Eloxieren

Für eine Verbesserung der Oberfläche der gefrästen Teile wurde die Eloxal-Technik gewählt. Da Aluminium ein relativ weiches Material ist, kann durch einer oxidischen Schutzschicht die Oberfläche widerstandsfähiger gemacht werden und so die unteren Schichten vor Korrosion schützen. Beim Eloxieren wird, im Gegensatz zu den galvanischen Verfahren, die Schutzschicht nicht auf dem Werkstück aufgetragen, sondern durch eine anodische Oxidation die obersten Metallschichten umgewandelt. Mit der freundlichen und kompetenten Unterstützung der Firma S&W Eloxal aus Kirchbichl konnte eine schwarzes Eloxal auf allen aus Aluminium gefertigten Prototypkomponenten angebracht werden.

3.4.3. Abdeckung

Die in Kapitel 3.2.6 beschriebene Abdeckung für das Gerät wurde mit einem CNC-Lasercutter aus Acryl gefertigt. Da es sich dabei um die Abwicklung der Abdeckung handelt musste diese später von Hand und unter Zufuhr von Wärme mit einem Heißluftföhn gebogen werden. Hierbei wurde die gelaserte Acrylplatte in einem Schraubstock zwischen zwei Holzelementen eingeklemmt und mit einem dritten Holzelement und unter der Wärmeeinwirkung langsam zu einem 90° Bogen geformt. Um die exakte Position der Biegung zu bestimmen wurde mit dem Laser eine Linie an der zu erstellenden Kante eingraviert, die Biegung und die Übereinstimmung der Befestigungslöcher am zusammengebauten Hauptteil überprüft und nachträglich kleine Anpassungen unter Wärmeeinwirkung vorgenommen. Anschließend konnte der An-/Aus Schalter, die USB-Buchse, der Drehregler und das Display auf der Abdeckung montiert und diese dann am Hauptteil mittels der Befestigungsschrauben fixiert werden. Für die Halterung der beiden kleinen Motorsteuergeräte und der zwei Mikrocontroller wurde ebenfalls Acrylglas verwendet und diese ebenfalls unter Hitzeeinwirkung so verformt, dass die Elemente mit genügend Abstand am unteren Massivteil des Prototyps befestigt werden konnten. Im letzten Schritt wurde eine untere Abdeckung gelasert, welche für die Halterung des Lüfters dient und als Träger für die spätere Verkabelung dienen soll.

Abb. 3.16 Links: Einspannung; Mitte: Erhitzen mit 220°; Rechts: Fertiges und eingepasste Abdeckung.

Abb. 3.17: Biegung des erhitzten Acrylglases mit einem Holz.

3.4.4. Montage

Im ersten Schritt wurden alle mechanischen Bauteile zusammengebaut, beginnend mit dem Vorschub (Hauptteil) des Prototyps und anschließender Verbindung der X und Y-Achse mit dem Gehäuse. Als erster Schritt wurden hier die Wellen mit Zahnrad und Kugellagern verbunden. Anschließend wurde das obere Massivteil mit allen Komponenten der Einspannung und des Vorschubs bestückt und die zusammengesetzten Wellen von unten eingesteckt. Abschließend konnten die obenliegenden Drehteller und das untere Massivteil zur Stabilisierung der Wellen montiert werden. Am Ende wurden das obere und das untere Massivteil mit den Gewindestangen verbunden und am unteren Ende Pfropfen aus Kunststoff als Füße montiert. Der Zusammenbau der Wagen für X und Y-Achse gestaltete sich etwas aufwendiger. Zuerst wurden die Gleitlager in die Sitze der Wagen gepresst. Die jeweils 4 Rundprofile wurden in die dafür vorgesehenen Bohrungen gehämmert und anschließend auf die dritte Kommastelle genau vermessen, um beide Wangen des

Führungssystems genau parallel anzuordnen. Nach mehrmaligem Justieren konnte der X-Achse Waagen in die Stäbe der Y-Achse eingeführt werden. Abschließend wurde die obenliegende Wange der Y-Achse montiert und wiederum genau vermessen und justiert. Die Überprüfung des Freigangs der Führungen zeigte, dass sich die X-Achse nicht flüssig auf den Rundprofilen bewegen ließ. Aus diesem Grund wurden Metallplättchen (Stärke 0,1 und 0,2mm) hinter die Wangen eingelegt und die Bewegungen wieder kontrolliert. Sobald auch der gesamte Umlenkkopf mit den beiden Achsen komplettiert wurde, konnte dieser über vier Verschraubungen mit dem Hauptteil verbunden werden. Mit der Anbringung der drei Schrittmotoren an die entsprechenden Sitze wurde die Montage der mechanischen Bauteile beendet.

Abb. 3.18: Ausgelegte eloxierte Teile mit Schrauben und Montagematerial.

Abb. 3.19: Teilweise zusammengesetzte X-Achse

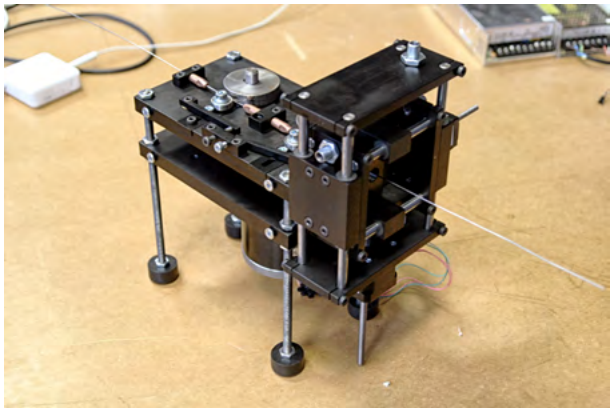


Abb. 3.20: Fertige Montage der Mechanik. Sicht von Vorne.

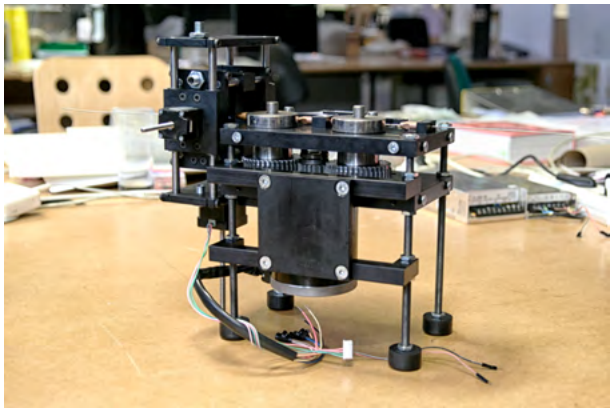
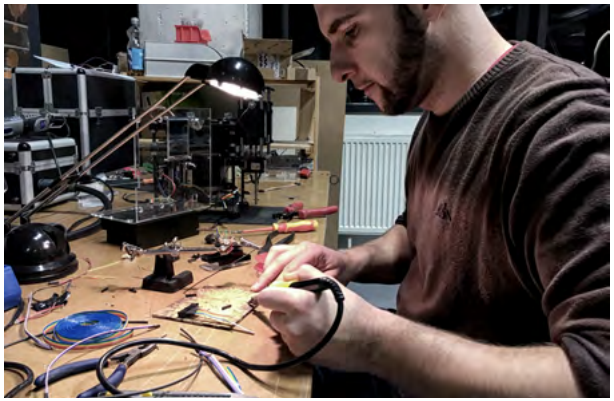
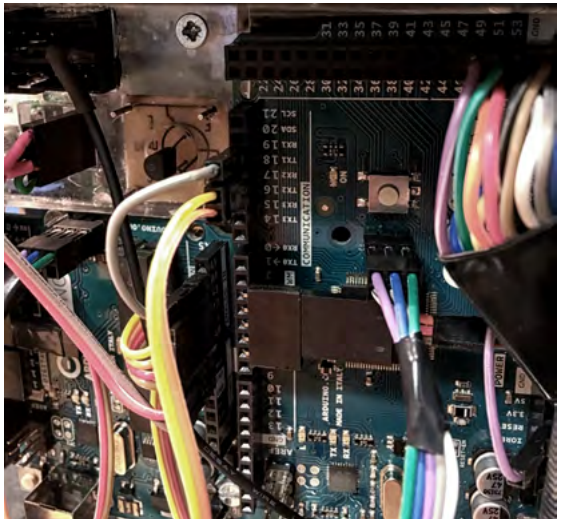


Abb. 3.21: Fertige Montage der Mechanik. Sicht von Hinten.

3.4.5. *Verkabelung*

Nachdem nun alle mechanischen und elektronischen Komponenten des Prototyps montiert wurden, konnte mit der Herstellung der Verkabelung begonnen werden. Als erstes wurden alle nötigen Verbindungen aufgelistet und ein Schaltplan entwickelt. Die beiden Mikrocontroller werden über die ICSP Schnittstelle verkabelt, teilen sich somit die Stromversorgung und können über die SPI (Serial Peripheral Interface) Leitungen dieser Schnittstelle miteinander kommunizieren. Die ICSP Schnittstelle wird mit 5 Leitungen (nachfolgend als Pin bezeichnet) verbunden. Der erste Pin wird als Clock (Zeitgeber) bezeichnet und ist für die Taktung der seriellen Signale zwischen den beiden Mikrocontrollern verantwortlich. Der zwei Pins MISO (Master In Slave Out) und MOSI (Master Out Slave In) werden für die Datenübertragung zwischen dem Master (Arduino Mega) und dem Slave (Arduino Uno) genutzt. Die Pins VCC (Positive Spannung) und GND (Masse) dienen der Stromversorgung. Das Arduino Mega wird ebenfalls über SPI (4 Pins) mit dem Touch-Display verbunden. Die Beleuchtung des Displays wird über einen weiteren Pin gesteuert. Dabei kann die Helligkeit mit PWM (Pulsweitenmodulation) über diese Leitung geändert werden. Vier Pins sind für die Eingangssignale der Touch-Funktion vorgesehen. Weitere vier Pins werden für den Drehregler verwendet, wobei die Masseleitung mit dem Display geteilt wird, um die Anzahl der Leitungen zu reduzieren. Am Arduino Uno werden jeweils zwei Pins für jedes der drei Steuergeräte für die Schrittmotoren verwendet, jeweils ein Pin für den Impuls und ein Pin für die Richtung der Motorendrehung. Ein Pin wird für den Ein-/Ausknopf, ein Pin für die Kontrollleuchte des Kopfes, ein Pin für die Warnleuchte und zwei Pins für mögliche externe Signale verwendet. Vier Pins werden für die Endschalter der X und Y-Achse verwendet. Damit diese die Motorenbewegung sofort abbrechen wurden Interrupt Pins für diese Endschalter gewählt. Um Kabellänge zu sparen, wurden alle GND (Masse) Leitungen für diese Endschalter zu einem Kabel vereint und bis ganz nach vorne geführt. In einer Schaltbox zwischen dem Umlenkkopf und dem Hauptteil werden alle Pins in die zwei Richtungen der Achsen verteilt. Hier laufen alle Kabel für die Motoren (jeweils 4) und zwei Kabel pro Endschalter (2x2 = 4 pro Achse) zusammen. Die gesamte Verkabelung wurde galvanisch getrennt, d.h. alle strombetriebenen Komponenten auf der Motorenseite werden über eine eigene Stromquelle versorgt und sind somit von den Stromkreisen der Mikrocontrollerseite und der damit verbunden Elektronik getrennt. Aus diesem Grund wurde auch die Stromzufuhr für die Motoren

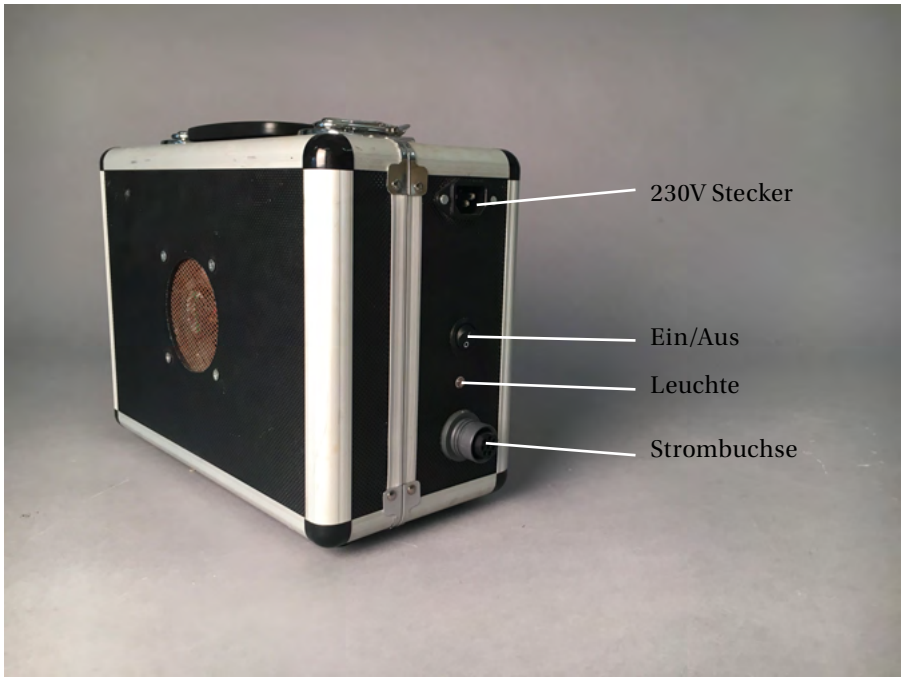




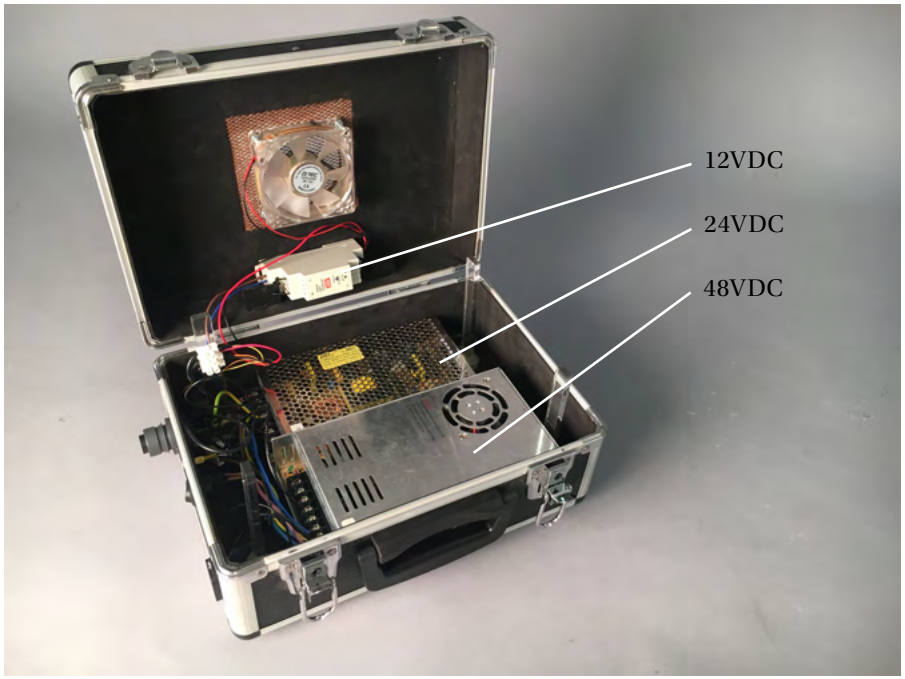

mit einer Masse (GND) und für die 12VDC Versorgung der Mikrocontrollerseite mit einer anderen Masse ausgestattet (siehe 3.3.2).

Die Stromkabelbuchse wurden an der Abdeckung angebracht und die Verkabelung auf dem Boden der Abdeckung fortgeführt. An der Vorderseite wurden alle Versorgungsleitungen für die Maschine an Lüsterklemmen mit diesen Stromleitungen verbunden. Daraufhin wurde das flexible Multicore-Kabel (8 Adern) mit den Steckern versehen und das große Steuergerät an der Hinterseite der Abdeckung angebracht, womit die Montage der Maschine beendet wurde.

3.4.6. *Stromzufuhr*

Weil die Schaltnetzteile nicht in der Planung am 3D-Modell berücksichtigt wurden, wurde ein geeigneter Werkzeugkoffer aus dem Baumarkt als Gehäuse umfunktioniert. Wieder wurde mit Acrylglas und dem Laser gearbeitet und eine geeignete Basishalterung für die Schaltnetzteile im Koffer hergestellt. Für den Kaltgerätestecker, den Ein-/Ausschalter, die Kontrollleuchte, die Belüftung und die Stromkabelbuchse wurden Öffnungen in den Werkzeugkoffer gefräst und diese dann fest mit dem Gehäuse verbunden. Im Inneren des Koffers wurden alle Verkabelungen auf die entsprechenden Anschlüsse der Schaltnetzgeräte gelegt und mit der 230VAC Stromversorgung, dem Ein-/Ausschalter und der Kontrollleuchte verbunden.





3.5. System-Architektur (Software Maschine)

Der wohl aufwendigste und langwierigste Teil der Entwicklung des Prototyps stellte die Entwicklung der Software für die Mikrocontroller dar. Deshalb wurde zunächst mit der Konzeption der System-Architektur begonnen. Ein solches Konzept ist wichtig, um alle Komponenten der Software eingliedern zu können und die jeweiligen Funktionen und Interaktionen untereinander vorauszuplanen. Eine skizzenhafte Auflistung aller geplanten Module wurde angefertigt und diente als weitere Stütze für die Programmierung. In dieser Übersicht wurden zuerst alle Motorsteuerungs-relevanten Prozesse von der Benutzeroberfläche getrennt. Deshalb wurden auch im Vorfeld zwei Mikrocontroller eingeplant. Die Auftrennung der Programmierung auf zwei Mikrocontroller ist zwar aufwendiger, stellt aber sicher, dass die Steuerung der Motoren sehr exakt und flüssig abläuft, was für die wissenschaftliche Auswertung der Resultate notwendig ist. Eine Programmierung beider Prozesse auf einem Kontroller hätte eine Beeinträchtigung der Motorsteuerung, bei Interaktion mit der Benutzeroberfläche des Displays zur Folge.

Bei der eigentlichen Programmierung der Mikrocontroller stellte sich sehr schnell heraus, dass die Arbeit mit der von Arduino mitgelieferten Entwicklungsumgebung nur sehr mühselig durchführbar wäre und zu Fehlern in der Programmierung führen würde. Aus diesem Grund wurde die Software Atmel Studio verwendet, die vom Hersteller des Mikro-Chips der Arduino Boards stammt. Diese Entwicklungsumgebung erlaubt es eine professionelle Projektstruktur aufzubauen und erleichtert die objektorientierte Programmierung in C++. Es wurden für jedes Arduino Board ein Projekt angelegt, welche über die Ordnerstruktur in verschiedene Programmkomponenten aufgegliedert wurden. Beide Projekte weisen einen Aufbau nach demselben Prinzip auf, um eine gute Orientierung beim Wechsel zwischen den beiden Projekten zu ermöglichen. Beide Projekte enthalten zwei Kernelemente, die für Mikrocontroller essenziell sind. Der sogenannte Loop stellt die Laufzeitfunktion des Kontrollers dar, das sogenannte Setup legt die Startparameter des Programms bei Aktivieren des Arduino Boards fest. Auf Basis dieser beiden Grundelemente wurden sukzessive alle Funktionen und Klassen erstellt und logisch in den beiden Strukturen eingeordnet. Das Arduino Mega beinhaltet im Loop u.a. wichtige Funktionsaufrufe für: die Kommunikation mit dem Arduino Uno, der Anzeige auf dem Touch-Display, das Lesen und Schreiben auf der integrierten SD-Karte, der Betätigung des Drehreglers und der seriellen

Kommunikation zwischen PC und der Maschine.

Das Arduino Uno beinhaltet u.a. Funktionen für die Kommunikation mit dem Arduino Mega, der Motorsteuerung, den vier Endschaltern (Not-Aus) und für die Rückmeldung des Zustands über die Kontrollleuchten.

Für die Übersichtlichkeit werden in diesem Kapitel nur die wichtigsten programmiertechnischen Lösungen behandelt, im Appendix (WireBender2 = Arduino Mega und StepperDriver = Arduino Uno) kann der Programmiercode für die beiden Mikrocontroller eingesehen werden.

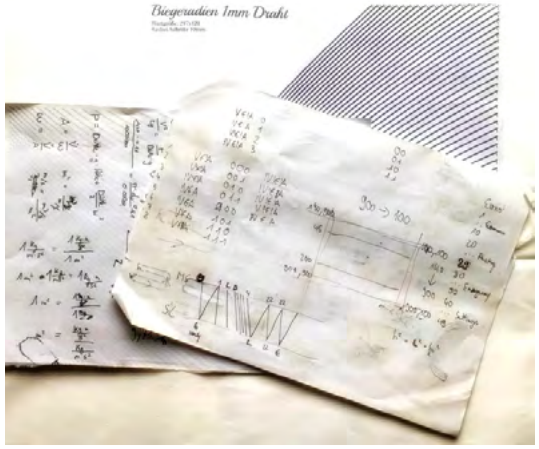


Abb. 3.22: Gefertigte Skizzen für die System-Architektur und Planung der Signalverarbeitung

3.5.1. Display (WireBender2 > 03_TFT)

Die gesamte Benutzeroberfläche des Displays wurde am PC mithilfe von Illustrator entworfen und dient später als Hilfestellung für die Programmierung. Alle grafischen Elemente wurden aus dem Code generiert, nur komplexe Grafikelemente wurden als Bilddatei auf der SD-Karte gespeichert, um eine möglichst schnelle Darstellung zu erreichen. Um eine möglichst effektive Programmierung und Speicherauslastung zu erzielen, wurden alle Schaltflächen auf dem Bildschirm über eine flexible Klasse erstellt. Auch wiederholende grafische Elemente, wie die Achsposition-Darstellung auf dem Heim-Bildschirm wurden als ein Klassenobjekt konzipiert. Alle anderen Schaltflächenfunktionen wurden im aufrufenden Code (Jogging, Browser, Viewer, Settings) strikt nach dem Muster Init (Initialisierung und Rückstellung), Draw (grafische Darstel-

lung), Click (Funktionsaufruf beim Berühren) in verschiedene Dateien abgelegt. Alle Funktionen für das Zeichnen und der Aktion beim Berühren der Schaltfläche wurden in der Klasse als sogenannte Callbacks abgelegt. Diese ermöglichen das Platzieren des eigentlichen Codes für jede Instanz der Schaltfläche außerhalb der Klasse. Beim Betätigen des Buttons wird nur der Verweis auf die Funktion aufgerufen (Callback). Somit werden alle für Schaltflächenobjekte relevanten Funktionen in der Klasse abgehandelt und die für jede spezifische Schaltfläche relevanten Aufrufe außerhalb der Klasse platziert. Dies spart zum einen Arbeitsspeicher und erhöht die Programmiereffizienz. Alle weiteren Erklärungen zu den Programmierlösungen bzgl. der Darstellung am Display würden den angestrebten Umfang dieses Kapitels sprengen, bei Interesse können weitere Informationen dem Appendix WireBender (Kapitel 6.1) entnommen werden.

Abb. 3.23: In Illustrator entworfene Benutzeroberfläche.

3.5.2. *Kommunikation* (WireBender2/StepperDriver > 02_Communication)

In der Funktionsgruppe Kommunikation befindet sich der Code für zwei verschiedene Kommunikationssysteme. Die Kommunikation über die serielle Schnittstelle (USB) für die Datenübertragung mit dem PC und die SPI Schnittstelle für den Befehlsaustausch mit dem jeweils anderen Mikrocontroller. Während das Projekt StepperDriver in der seriellen Schnittstelle nur Funktionen für das Debugging (Testen) während der Entwicklungsphase beinhaltet, stellt das Projekt WireBender fast alle wichtigen Befehle für die Kommunikation mit dem PC zur Verfügung. Es ist somit möglich über den PC mit einem USB Kabel eine Verbindung über den entsprechenden COM-Port (baud= 19200) zur Drahtumformmaschine aufzunehmen und komplett fernzusteuern. Diese Schnittstelle wird aber hauptsächlich für die Übertragung der Befehlsdateien vom PC bzw. Grasshopper genutzt.

Die Kommunikation zwischen den beiden Mikrocontrollern wurde über SPI mit einer relativ hohen Taktrate konzipiert. Durch elektromagnetische Interferenzen mit den anderen Leitungen im Gehäuse können fehlerhaft übertragene Bits auftreten. Um die Korrektheit der übertragenen Daten gewährleisten zu können, wird bei dieser Kommunikation eine sogenannte Prüfsumme (genauer Fletcher Algorithmus) vor der eigentlichen Übertragung an den anderen Mikrocontroller übertragen. Nach der Übermittlung des Datenpakets wird dieser 16 bit lange Schlüssel mit dem aus den erhaltenen Daten errechnete Schlüssel verglichen. Falls diese nicht übereinstimmen, wird ein Übertragungsfehler an das Gerät zurückgegeben und eine erneute Übertragung ausgelöst. Um eine Endlosschleife zu vermeiden wird die Übertragung auf 5 Versuche begrenzt, ehe eine Fehlermeldung auf dem Bildschirm für den Benützer erscheint.

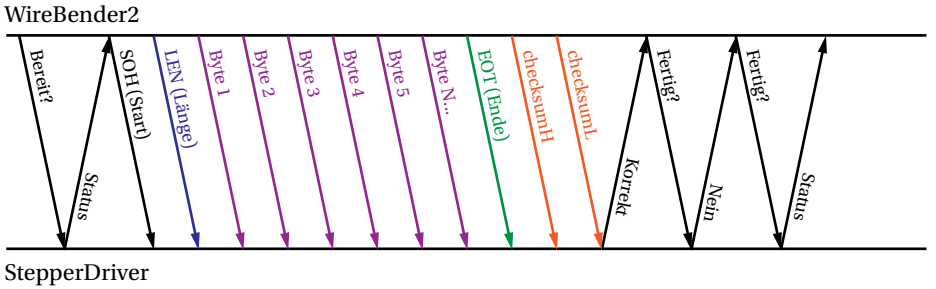


Abb. 3.24: Diagrammatischer Kommunikationsablaufes zwischen den beiden Mikrocontrollern.

Eine Prüfsumme, auch Checksum genannt, ist ein Wert mit dem die Integrität von Daten überprüft werden kann. Die Prüfsumme wird aus allen Zeichen des zu übertragenen Datensatzes berechnet und ergibt eine 16bit Zahl (max. 65.535) Diese wird über zwei bytes (8bit) übertragen. Der Empfänger berechnet eine Prüfsumme mit dem Fletcher Algorithmus aus den empfangenen Daten. Nun kann die übertragene Prüfsumme mit der errechneten verglichen werden. Unterscheiden sich die Checksums, ist ein Fehler in der Übertragung vorhanden.

Alle weiteren Erklärungen zu den Programmierlösungen bzgl. Kommunikation würden den angestrebten Umfang dieses Kapitels sprengen, bei Interesse können weitere Informationen dem Appendix WireBender (Kapitel 6.1) entnommen werden.

3.5.3. *Einstellungen* (WireBender2 > 10_Settings)

Alle permanenten Einstellungen, wie Grundposition, Geschwindigkeit und Beschleunigung, letzte Position vor Ausschalten des Gerätes und letztes geladenes Programm werden in der „Settings.s“ Datei im Hauptverzeichnis der SD-Karte gespeichert. Beim Start der Drahtumformmaschine wird diese Datei ausgelesen und alle Werte an den StepperDriver übermittelt. Die Grundeinstellungen können durch Aufrufen der Einstellungsseite am Display geändert werden. Dazu wurde der Drehregler als Eingabeinstrument für die Werte eingesetzt. Bei jeder Speicherung des geänderten Wertes wird der Wert automatisch über SPI an den StepperDriver übertragen.

3.5.4. *Motorsteuerung* (StepperDriver > 03_Stepper)

Die Motorsteuerung mittels Impulsen wurde mithilfe einer im Loop platzierten Funktion getätigt. Wenn eine Positionsänderung vorliegt, werden so lange Impulse an das Steuergerät des Schrittmotors gesendet, bis diese erreicht ist. Falls es zuvor zu einem Auslösen des Endschalters bzw. Not Aus Tasters kommt, weil die Achse ihr Ende erreicht hat, so werden die Impulse abrupt geändert, um ein Weiterlaufen der Motoren zu unterbinden.

3.6. Vereinfachte Schnittstelle

Das Ziel dieser Arbeit ist es, dem Benutzer, dem Architekten oder Designer die Möglichkeit zu bieten, diese Technologie zugänglich zu machen und für seine Gestaltungsmöglichkeit zu nutzen, ohne sich ein dafür notwendiges Fachwissen aneignen zu müssen. Um dieses Ziel zu erreichen wurde eine vereinfachte Schnittstelle zu der beschriebenen Umformtechnik entworfen und in Form einer Softwareerweiterung für die CAD Anwendung Rhinoceros (McNeel) erstellt.

Die Hauptaufgabe dieser vereinfachten Schnittstelle ist es, die vom Benutzer entworfenen Kurven im CAD Programm in geeignete Maschinenbefehle umzuwandeln und durch Berechnung der Überbiegungswinkel und der Positionen der Umlenkungsachsen eine Rückmeldung über mögliche Probleme in der Fertigung zu geben. Mit dieser Feedback-Loop ist es dem Benutzer möglich Entscheidungen über sein Design zu überdenken, ggf. zu ändern und somit ungewollte Resultate in der Fertigung zu vermeiden. Die Informationen sollen in verschiedenen Detailstufen abrufbar sein, damit der Benutzer auch über die Detailinformationen des spezifischen Problems verfügt. Mit diesen Informationen kann er seine Arbeitsweise ständig überprüfen und läuft nicht Gefahr bei Vollendung des Entwurfs diesen völlig neu gestalten zu müssen, weil Teilprobleme in der Umsetzung das Gesamtkonzept in Gefahr bringen würden.

Abb. 3.25: Auflistung aller Komponenten der vereinfachten Schnittstelle für den WireBender.

3.6.1. NURBS Approximation (CurveToArcs)

Die Eingabe von NURBS-Kurven wird von dieser Schnittstelle übernommen und in für die Maschine steuerbare Befehle umgewandelt. Dabei müssen NURBS-Kurven aufgrund ihrer geometrischen Eigenschaften mit Kurven, die nur aus Bögen (sog. Arcs) bestehen, angenähert werden. NURBS (Non-Uniform Rational Bezier-Splines) sind Kurven, die mathematisch definiert sind. Gewählte Kontrollpunkte dienen als Parameter für die Errechnung dieser Kurven und die exakte Form kann durch die vom Grad (Degree) bestimmte algebraische Funktion bestimmt werden. Die meisten aktuellen CAD Programme unterstützen diese NURBS Technologie und erlauben es mit sehr einfachen Mitteln Freiform-Kurven und Flächen zu erstellen. Die in der Mitte der 90er Jahren von der Autoindustrie (Mathematiker de Casteljau von Citroën) entwickelten Methode für die digitale Darstellung von Freiform-Geometrie hat entscheidenden Einfluss auf die aktuelle Gestaltung in Architektur und Design. Durch die mathematische Bestimmtheit ist diese Darstellungsform von großem Wert für die Fabrikation, weil aus dem entworfenem Modell u.a. geometrische Überprüfungen durchgeführt, fertigungsrelevante Daten gewonnen und mathematische Näherungsverfahren angewendet werden können. Letzteres spielte in dieser Arbeit, bei der Umsetzung für die Annäherung der NURBS-Kurven durch Bögen, eine wichtige Rolle.

Beim Näherungsverfahren (engl. Approximation) für diese Kurven wurde auf einen bekannten Algorithmus zurückgegriffen. Es wurde auf Basis der im „Journal of Computation and Applied Mathematics 59“ (1995) von D.J. Walton und D.S. Meek veröffentlichten wissenschaftlichen Arbeit programmiert. Die Arbeit von Walton & Meek beschreibt im Allgemeinen, wie die algebraische Beschreibung einer quadratischen Bezier-Kurve mit Bögen angenähert werden kann. Unter quadratischen Bezier-Kurven versteht man eine aus drei Kontrollpunkten bestehende Kurve. Diese kann durch zwei kreisförmige Bögen, die Endpunkte und deren Tangenten teilen, angenähert werden. Um eine möglichst gutes Resultat zu erhalten, zeigen Walton & Meek, wie der optimale Radius der beiden Kreisbögen berechnet werden kann. Aus den Koordinaten der Kontrollpunkte kann unter Anwendung folgender Formel eine Tangente ermittelt werden, aus der sich die Kreisbögen bestimmen lassen.

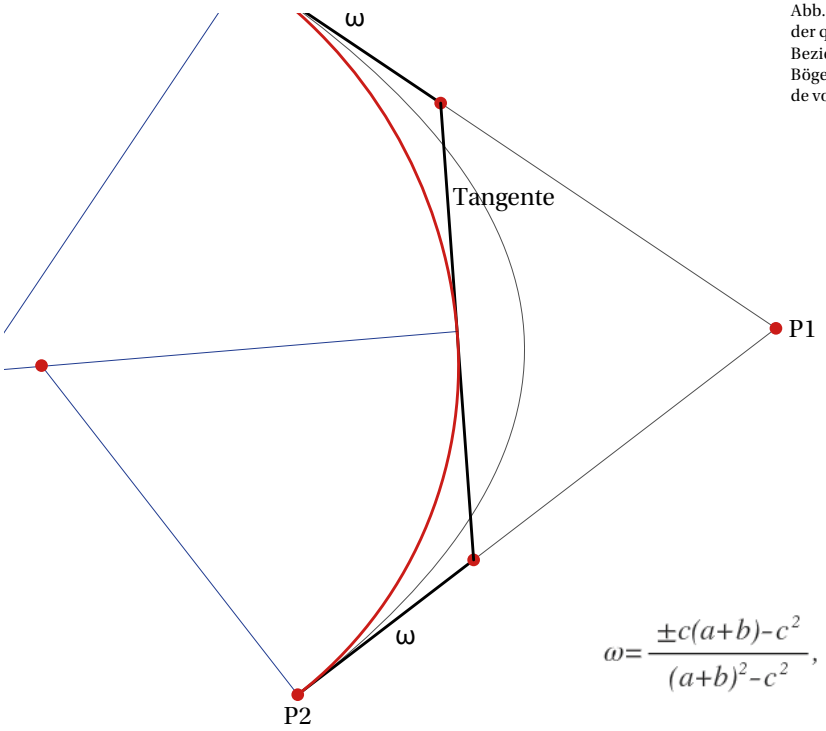


Abb. 3.26: Näherung der quadratischen Bezier-Kurve mit zwei Bögen, nach der Methode von Walton & Meek.

$$\omega = \frac{\pm c(a+b) - c^2}{(a+b)^2 - c^2}, \quad 0 \leq \omega \leq$$

Da NURBS Kurven aus Bezier-Splines bestehen, konnte in dieser Arbeit ein Algorithmus programmiert werden, welcher über folgende Schritte eine im CAD gezeichnete räumliche NURBS-Kurve mit Bögen annähert:

-Überprüfung eines Grades (Degree) von 2 für die beschriebene Kurve und ggf. Rückmeldung an den Benutzer.

-Ermittlung der kubischen Beziers für die NURBS-Kurve.

-Berechnung der quadratischen Beziers aus den kubischen Beziers

-Annäherung der kubischen Beziers mit Biarcs (zwei Bögen) nach Walton & Meek

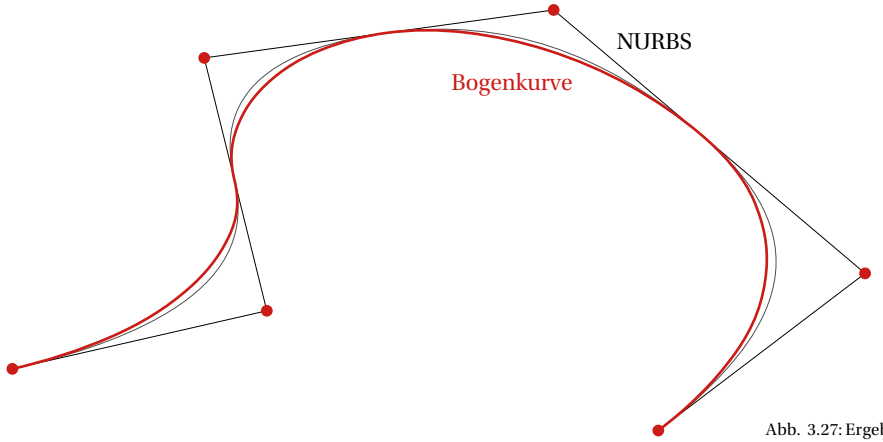


Abb. 3.27: Ergebnis des erstellten Algorithmus für die Näherung von NURBS Kurven.

3.6.2. *Postprozessor & Feedback* (WireCommand)

Bei der Programmierung des Feedback-Loops für den Benutzer wurde die Berechnung der Maschinenbefehle genutzt, um über die Umsetzbarkeit des Designs zu informieren. Der Benutzer erhält im CAD Programm sofort eine farbliche Rückmeldung über seine eingegebene Kurve. Diese stellt dar, ob es in einem Abschnitt einen zu kleinen Radius gibt, welcher von der Maschine nicht herstellbar ist, oder ob die Länge des Materials nicht ausreichen wird, um die gewünschte Kurve in einem Stück umformen zu können. So kann der Benutzer die Krümmungen seiner Kurven berichtigen oder durch Kappen die Länge und somit Überlappungspunkte der später gebogenen Metallstäbe selbst wählen. Die Farbe Rot wird wegen ihrer Signalwirkung verwendet, um einen zu kleinen Radius der an der Stelle errechneten Bögen darzustellen. Verschiedene andere Farben werden abwechselnd dafür angezeigt, wenn eine Kurve länger als das Ausgangsmaterial für die Umformung ist und deshalb die Kurve mit mehreren Stäben erstellt werden muss. Der Benutzer kann für ein schnelleres Arbeiten zwischen verschiedenen Stufen der Rückmeldung wählen und somit zusätzliche Informationen ein und ausblenden. So kann er sich zusätzlich zur Farbdarstellung auch eine Nummerierung der einzelnen Bögen anzeigen lassen. Mithilfe dieser Nummerierung und der Textausgabe des Postprozessors kann er den Betrag des Radius an einer bestimmten Stelle einsehen und damit den Grad der benötigten Justierung einschät-

zen. Auch ist es dadurch möglich abzuschätzen, wo eine bestmögliche Trennung der Kurve erfolgen kann, um das Design durch die Bruchstellen nicht ästhetisch zu beeinträchtigen.

Im Postprozessor werden zur selben Zeit auch alle Maschinenbefehle erzeugt. Diese können nach zufriedenstellendem Gestalten der Kurven mithilfe der Kommunikationskomponenten (siehe Kapitel 3.6.4) direkt an die Umformmaschine gesendet werden. Als Grundlage für die Berechnung des kleinstmöglichen Radius und der benötigten Überbiegung wird die Materialtabelle genutzt.

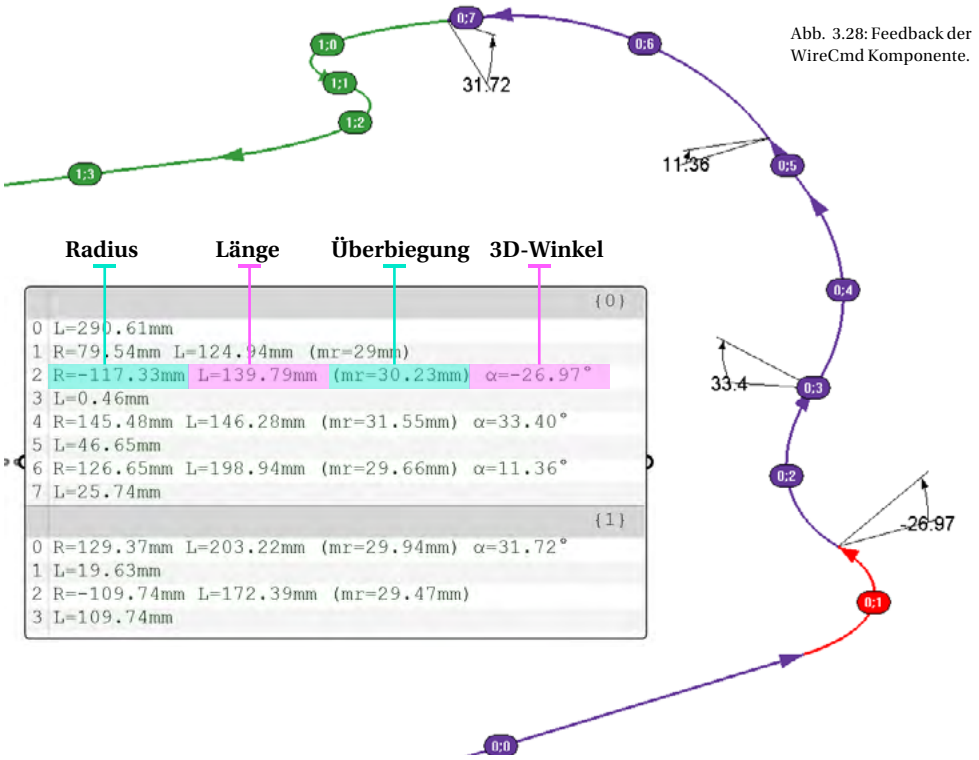


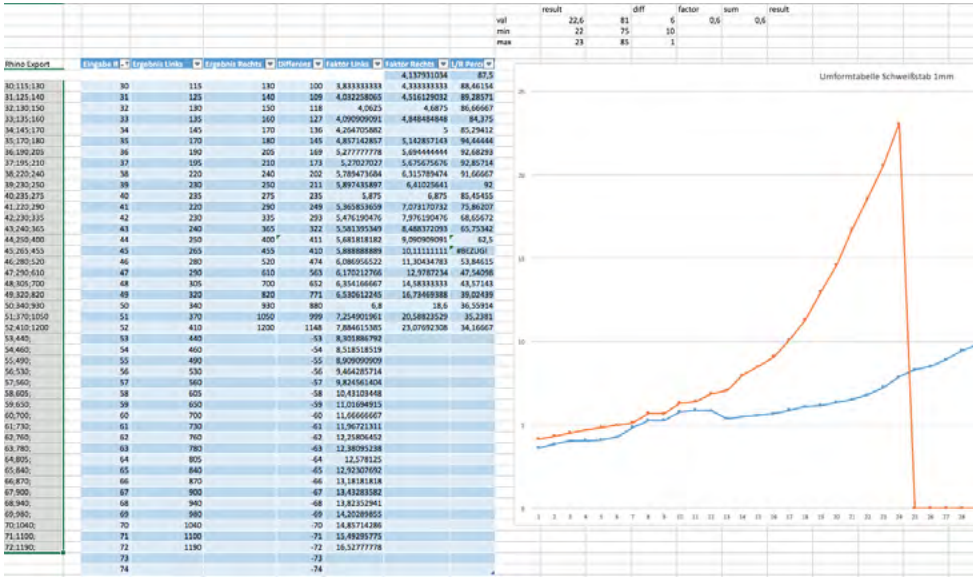
Abb. 3.28: Feedback der WireCmd Komponente.

3.6.3. *Materialtabelle* (MaterialTable)

Die Erstellung der Materialtabelle wurde nach der empirischen Methode vorgenommen. Für jedes Material und jeden Durchmesser muss eine eigene Tabelle erstellt werden. Dabei wird schrittweise ein Radius manuell an der Drahtumformmaschine eingestellt und

ein Bogen mit genügender Länge erzeugt. Danach kann der produzierte Bogen mithilfe des Messblattes vermessen werden. Mit dem händisch gemessenen Radius und dem Radius der in der Maschine eingegeben wurde, kann die Tabelle gefüllt werden. Schrittweise ist nun der Radius zu verringern, bis der kleinstmögliche Radius für die Umformmaschine erreicht ist. Dieser Radius ergibt sich sobald der Radius so klein ist, dass eine Kollision mit der Konstruktion des Umlenkkopfes auftritt. Ein Weiterbiegen unter Kollision würde ein verfälschtes Biegeergebnis liefern. Sobald alle Daten nach der frei gewählten Auflösung (im besten Fall 1mm Schritte) vorgenommen wurde und die Tabelle nach eigenem Ermessen komplett ist, kann diese in den Postprozessor (siehe Kapitel 3.6.2) gespeist werden. Sobald eine Materialtabelle vorhanden ist, wird der Postprozessor anhand der Materialtabelle für alle Radien die Überbiegung berechnen und das Resultat in der Befehlsdatei ausgeben. Die Berechnung der Überbiegung geschieht anhand der Tabelle, durch eine Interpolation können Werte zwischen den Tabellenzeilen errechnet werden.

Abb. 3.29: In Excel aufgezeichnete Werte nach der Messung verschiedener Materialien.



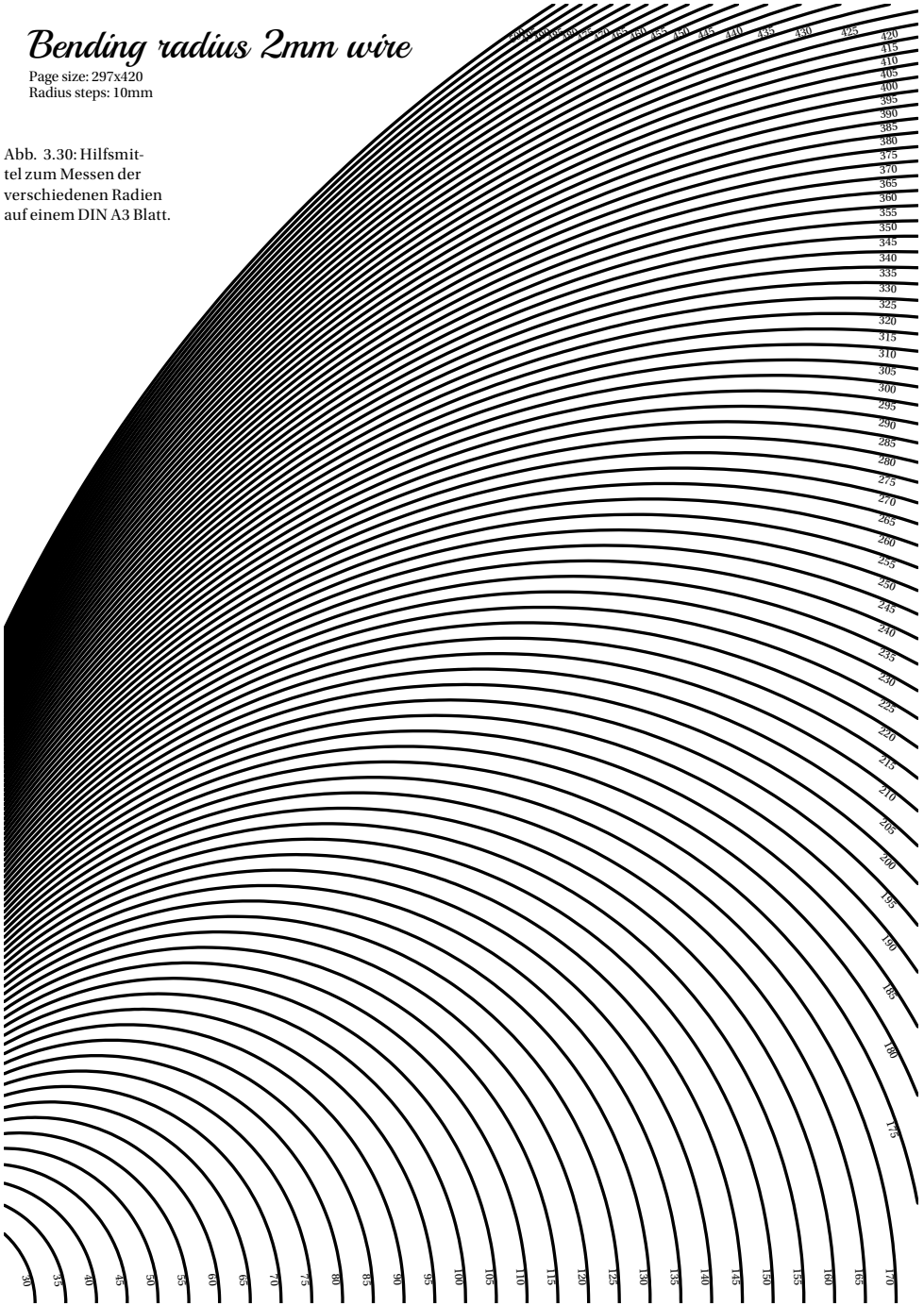
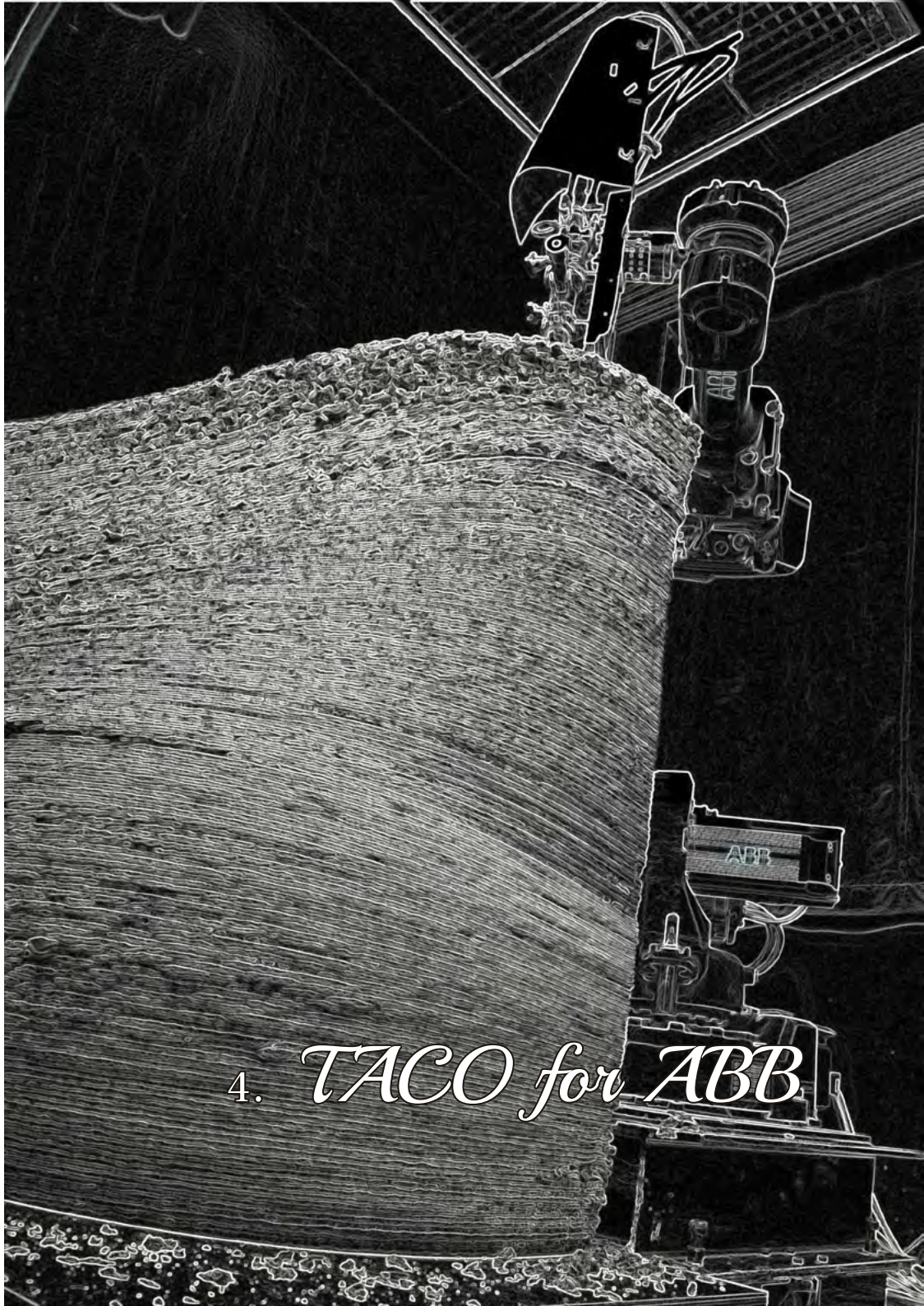


Abb. 3.30: Hilfsmittel zum Messen der verschiedenen Radien auf einem DIN A3 Blatt.

3.6.4. Kommunikation (Connect & Upload)

Für eine benutzerfreundliche Bedienung wird die Kommunikation und Datenübertragung vom Postprozessor getrennt. Dies ermöglicht es eine permanente Verbindung mit dem Gerät aufzunehmen und über eine kleine Fläche am linken oberen Bildschirmrand den Status der Verbindung und die Rückmeldungen der Umformmaschine zu erhalten. Die aktive Verbindung kann so auch von anderen Komponenten (z.B ManualCommand), wie dem Senden von manuellen Befehlen, verwendet werden.

Die Komponente für die Datenübertragung (WireBenderUpload) ermöglicht dem Benutzer die berechneten Befehle aus dem Postprozessor (WireCommand) an die Umformmaschine zu senden. Mit der Angabe des Projektnamens und der Eingabe der Befehlsliste kann diese per Knopfdruck an das Gerät gesendet werden. Hierbei wird die Verbindung über USB genutzt und Zeile für Zeile auf der SD-Karte der Umformmaschine als Datei gespeichert. Es können mehrere Dateien gleichzeitig übertragen werden. In diesem Fall wird mit dem angegebenen Projektnamen ein Ordner auf der SD-Karte erstellt und die einzelnen Dateien, die jeweils einem Stahlstab entsprechen, in diesem abgelegt. Es ist so möglich ein Biegeprojekt in einer gewünschten Ordnerstruktur zu organisieren. Diese Datenübertragung nützt auch die Checksum Technik, um die Vollständigkeit der Übertragung zu überprüfen (siehe Kapitel 3.5.2)



4. TACO for ABB

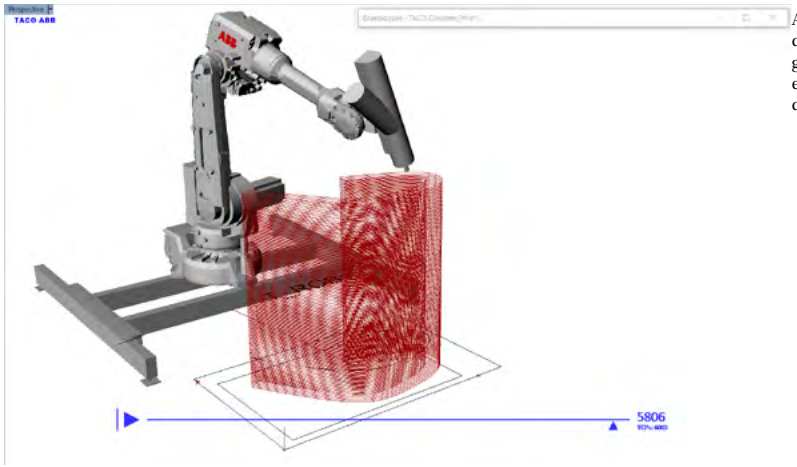


Abb. 4.1: Simulation der Roboterbewegung anhand des erstellten Pfades für das Betondrucken.

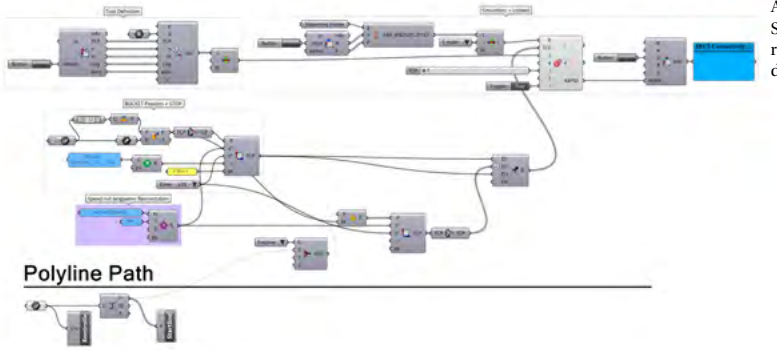


Abb. 4.2: Script für die Simulation und Kontrolle diverser Abläufe der Roboterbewegung.

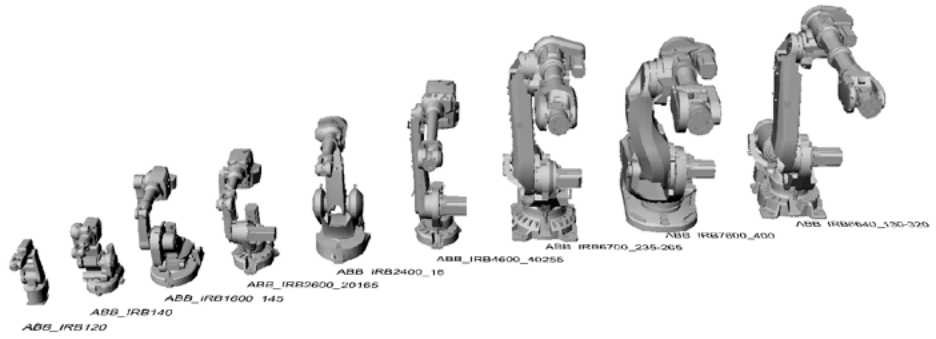


Abb. 4.3: Aufstellung aller zurzeit verfügbaren Robotermodelle in TACO.

Der Einsatz von industriellen Robotern in verschiedenen Bereichen der Forschung gewinnt zunehmend an Bedeutung. Auch im Forschungsbereich Architektur finden Industrieroboter Einzug. Das erforderliche technische Fachwissen für die Bedienung von industriellen Robotern ist relativ hoch und fordert von den Forschern viel Zeit für die Einarbeitung in die Materie. Neben ihrer eigentlichen Forschungsarbeit muss viel Aufwand an Vorbereitung und die entsprechende Schulung aufgebracht werden, dass ein Arbeiten mit diesen Maschinen möglich ist. Auch im Zuge dieser Arbeit wurde an den Robotern getestet und gearbeitet. Im Zusammenhang mit der Zielsetzung der Arbeit „Wirement“ war es deshalb schlüssig sich auch gerade in diesem Bereich für eine Bereicherung der vereinfachten Bedienung und Erweiterung der Gestaltungsmöglichkeiten zu widmen.

Taco für ABB ist nicht Teil dieser Arbeit. Es wird jedoch mit dieser Arbeit in Verbindung gesetzt, da es in der selben Zeit entstanden ist und dieselben Ziele verfolgt. Taco ist als Projekt im Zuge einer Lehrveranstaltung an der Universität Innsbruck am Institut für Experimentelle Architektur / Hochbau (Univ.-Prof. Marjan Colletti, PhD) in Kooperation mit der Media x Design Lab EPFL (Prof. Jeffrey Huang) entstanden und wird von den Entwicklern Florian Frank (Autor dieser Arbeit), Shih-Yuan Wang und Yu-Ting Sheng ständig weiterentwickelt. Das Ziel unserer Arbeit war es auch hier mithilfe von programmierten Erweiterungen für die CAD Anwendung durch Simulation, die Steuerung/Bedienung von komplexen Roboterbewegungen zu erleichtern und diese Software kostenlos für jeden zur Verfügung zu stellen. Durch das Beschäftigen mit dieser Thematik und das gezielte Erlernen von technischen Kenntnissen wollen wir durch Taco diese Technologie für möglichst viele Forscher und Interessierte zugänglich machen, aber auch die Arbeit für Spezialisten erleichtern. Da grundlegende Unterschiede in der Softwaretechnologie zwischen den verschiedenen Herstellern von Industrie-Roboterarmen liegen, ist Taco vorerst nur für den Hersteller ABB (Asea Brown Boveri) erstellt worden.

Grundlegende Ziele dieser Software sind die Simulation der Inversen Kinematik, die Überprüfung von Kollisionen und eventuelle Singularitäten (Problemposition für Roboter) über grafische Feedback-Loops für die Benutzer vereinfacht zugänglich zu machen. Somit ist es möglich ohne fachliche Kenntnis und nur mittels grafischer Eingabe eine Roboterbewegung zu erstellen und umfangreich zu simulieren. Dies ermöglicht Korrekturen und beschleunigt das Arbeiten mit den Robotern. Bei fertiggestelltem Bewegungs-

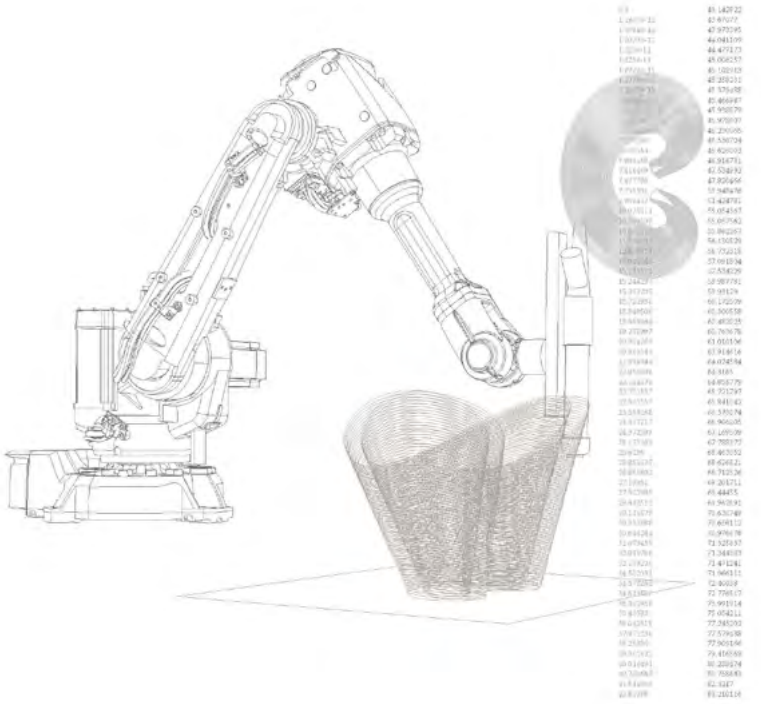
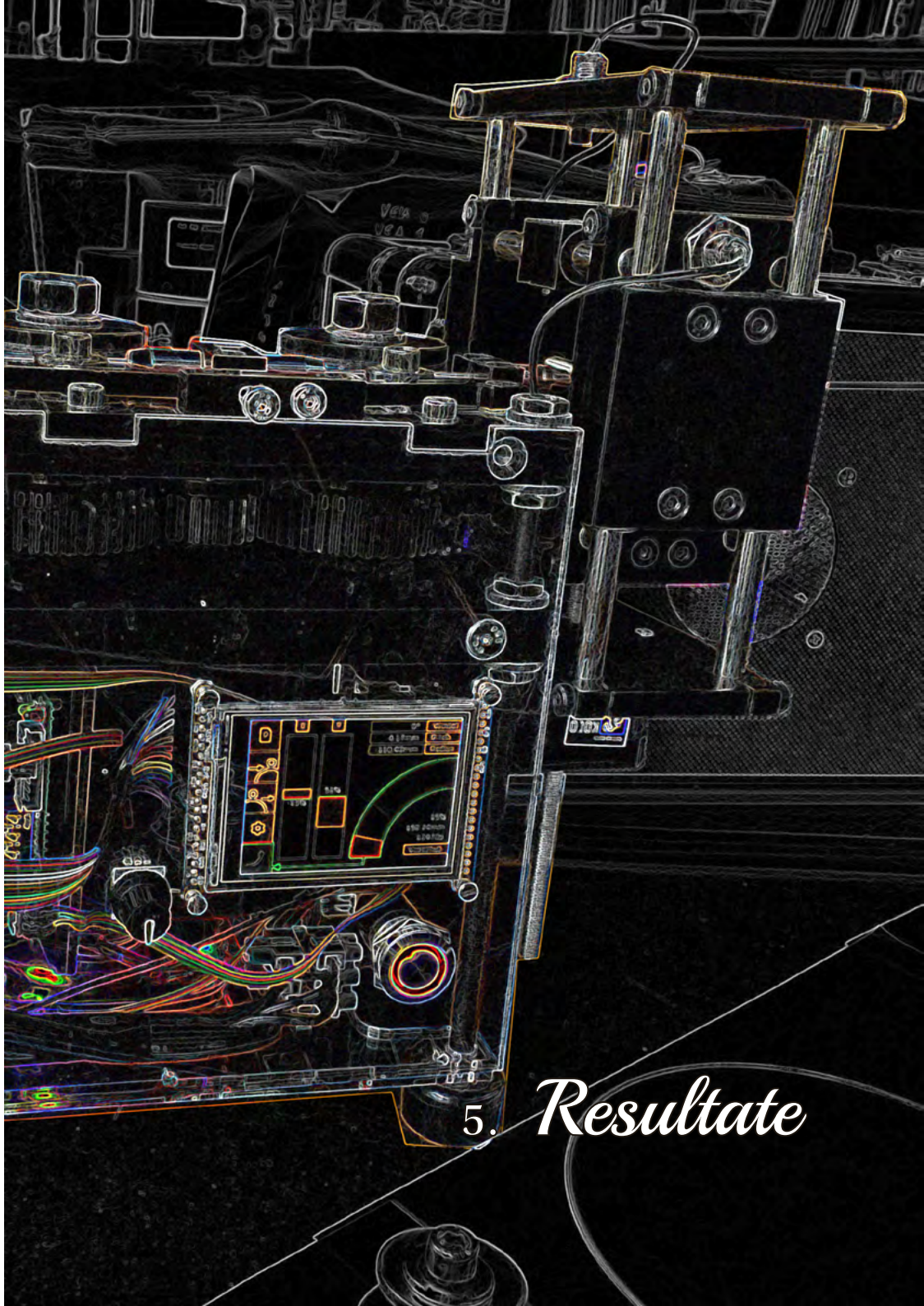


Abb. 4.4: Einsatz von Taco beim 3D-Betondruck an der YunTech

ablauf können die durch Taco generierten Befehle per Knopfdruck direkt an den Roboter-Kontroller gesendet werden. Es wurden die vom Hersteller zur Verfügung gestellten Programmierschnittstellen verwendet, um eine direkte Verbindung zum Robotercontroller aufnehmen zu können und am Roboter selbst erstellte Daten vom Kontroller zu laden. Nach der Veröffentlichung von Taco für ABB im Internet, kam die Nachfrage für die Unterstützung verschiedener ABB Robotermodelle von verschiedenen Institutionen. Im Laufe der Zeit wurden Funktionen erweitert und es wird laufend die Unterstützung von neuen Robotertypen aktualisiert.

Alle Erklärungen zu den Funktionen von Taco für ABB würden den angestrebten Umfang dieses Kapitels sprengen und aufgrund der fortlaufenden Aktualisierung obsolet werden. Bei Interesse finden sich weitere Informationen, Erklärungen und Beispiele auf der Webseite http://taco.blickfeld7.com. Die aktuellste Version dieser Software kann dort heruntergeladen werden. Im Kapitel 5.3 werden Resultate des 3D-Druckens von Beton unter Verwendung von Taco und einem ABB Industrieroboter gezeigt.



5. Resultate

Das Umformen von Draht gestaltete sich anfangs recht mühsam. Im Laufe der ersten Resultate wurden Verbesserungen an der Maschinensoftware und der vereinfachten Schnittstelle gemacht. Die in dieser Arbeit präsentierten Resultate zeigen die verbesserte Methode.

Als Erstes wurden Stahlstäbe mit einem Durchmesser von 1mm umgeformt. Nach anfänglicher Eingabe der Radien von Hand wurde der Radius schrittweise um 1mm verkleinert, bis eine Kollision mit der Konstruktion festzustellen war. Alle Ergebnisse wurden aufgezeichnet und in einer Excel Tabelle vermerkt. Das erstaunliche Ergebnis der Auswertung dieser Tabelle war, dass der Faktor zwischen eingegebenen Radius und Resultat (Radius nach Rückfederung) ein fast lineares Verhalten aufweist. Es ist davon auszugehen, dass minimale Messfehler und kleinste Ungenauigkeiten in der Methodik geringe Abweichungen in der Auswertung der Resultate ergeben. Da in Fertigungsverfahren immer mit Toleranzen zu rechnen ist und die Abweichungen in einem sehr geringen Bereich liegen, wird dieses erste Resultat als positiv gewertet.

Mit dem Ergebnis und der ersten Version der vereinfachten Schnittstelle wurden erste Biegeversuche erstellt, um die Verlässlichkeit

	Eingabe	Ergebnis	Differenz	Faktor
1	22	75	53	3,409090909
2	23	80	57	3,47826087
3	24	95	71	3,958333333
4	25	115	90	4,6
5	26	125	99	4,807692308
6	27	135	108	5
7	28	150	122	5,357142857
8	29	175	146	6,034482759
9	30	185	155	6,166666667
10	31	235	204	7,580645161
11	32	260	228	8,125
12	33	285	252	8,636363636
13	34	320	286	9,411764706
14	35	350	315	10
15	36	380	344	10,55555556
16	37	445	408	12,02702703
17	38	500	462	13,15789474
18	39	570	531	14,61538462

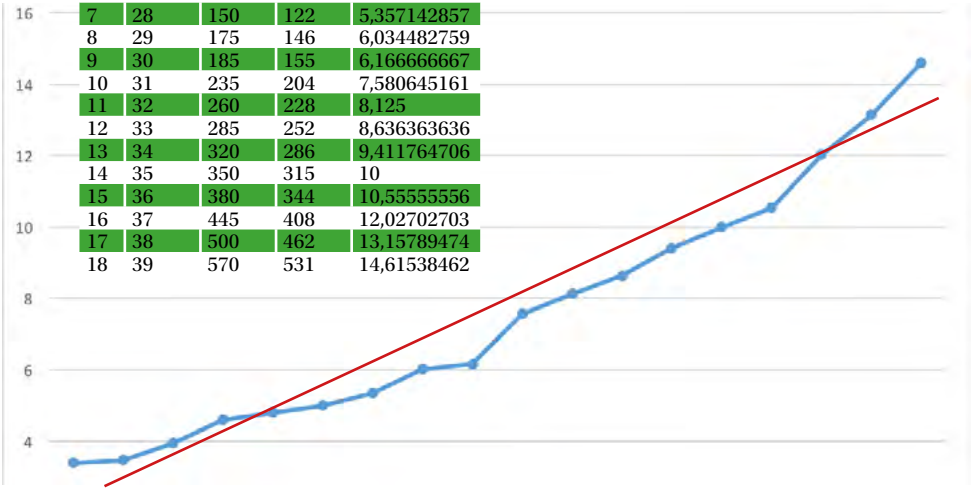


Abb. 5.1: Auswertung der aufgezeichneten Daten nach der Biegung von Schweißdraht.

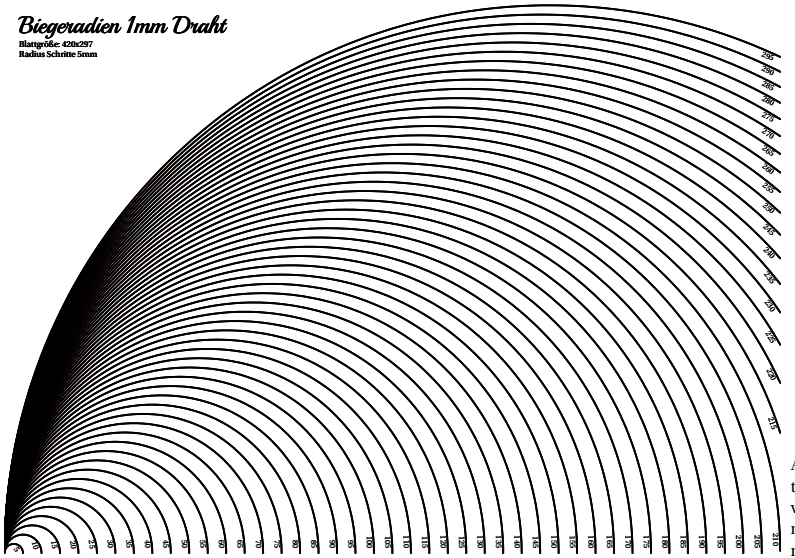


Abb. 5.2: Hilfsmittel für die Messung von Schweißdraht mit einem Druchmesser von 1mm

und Präzision der Umformung bei Wiederholung zu testen. Begonnen wurde mit der Erstellung von geometrischen Grundformen, diese sollen zeigen, wie Radien und Längen vom Prototypen richtig produziert werden und mit der entworfenen Form im CAD Programm übereinstimmen. Es wurden ein Quadrat mit einem Eckradius von 75mm, ein Dreieck mit 85mm Eckradius und eine Endlosschleife mit Radien von 75mm und 130mm als Testobjekte ertellt. Nach Eingabe der aus der Excel Tabelle errechneten Werte für den Radius mit Überbiegung wurde eine Befehlsliste auf die SD-Karte des Prototypen gespeichert und mit der Umformung der drei Versuchsobjekte begonnen. Anschließend wurde die Genauigkeit der Ergebnisse überprüft und die Übereinstimmung der beiden Enden mit geringer Toleranz als positives Resultat gewertet. In weiterführenden Tests wurde versucht durch wechselnde Richtung der Radien eine Welle zu produzieren, um eine eventuelle Krümmung des Gesamtobjektes zu erkennen.

Nach den grundlegenden Tests in der Umformung zu ebenen Objekten wurden erste Versuche für die 3D Verformung gemacht. Durch Angabe von Winkeln zwischen den Bogenstücken wird eine Krümmung im Raum erstellt. Auch diese Resultate erwiesen sich als zufriedenstellend und die weitere Vereinfachung der Schnittstelle für Grasshopper erfolgte mit den erworbenen Erkenntnissen aus der Bedienung des bisherigen Fertigungsprozesses.

Als letzter Test wurde versucht eine möglichst komplexe Formge-

bung für die Stäbe zu finden. Nach Überlegungen über eine gute Demonstration der Resultate wurde die Nachbildung eines Schriftzugs als zielführendes Objekt gewählt. Hierbei sollte der Titel dieser Arbeit „Wirement“ mit dem 1mm Draht geformt werden. Der Entwurf wurde mit der verbesserten Schnittstelle im CAD gezeichnet und mit der in Kapitel 3.6.2 beschriebenen Feedback-Loop einer ständigen Überprüfung und Korrektur, bis zur gewünschten Formgebung, unterworfen.

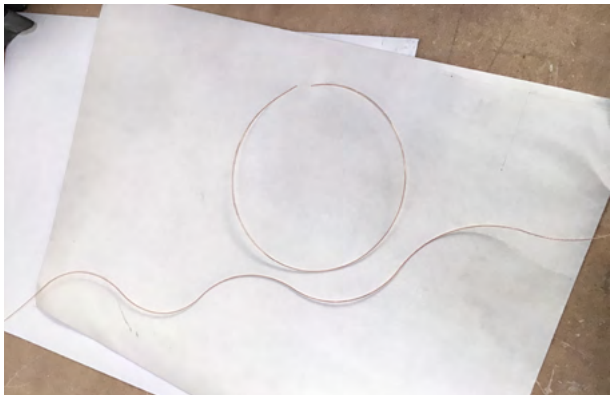


Abb. 5.3: Geschlossener und gleichmäßig geformter Kreis. Welle mit gleichen Radien.

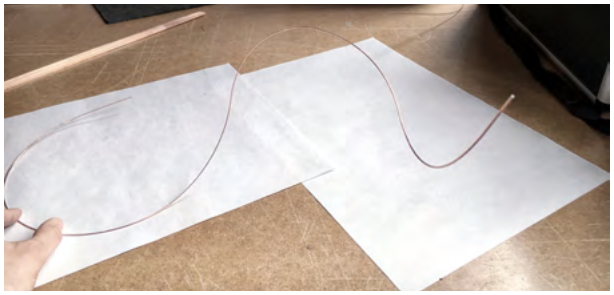


Abb. 5.4: Erste Umformung in 3D.

Abb. 5.5: Erstellter Schriftzug „Wirement“ mit 3D-Umformung der 1mm Stäbe.

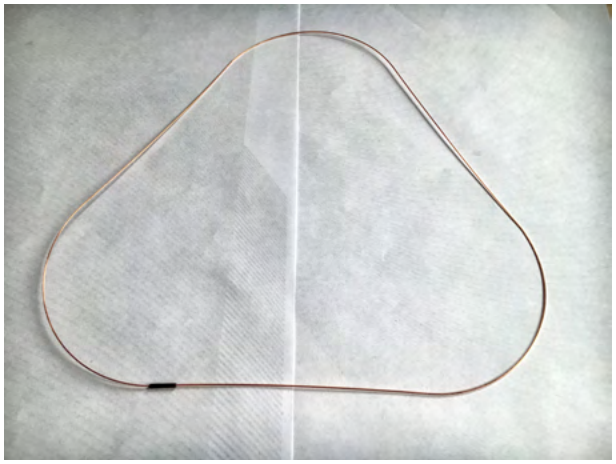


Abb. 5.5: Geschlossenes und gleichförmiges Dreieck

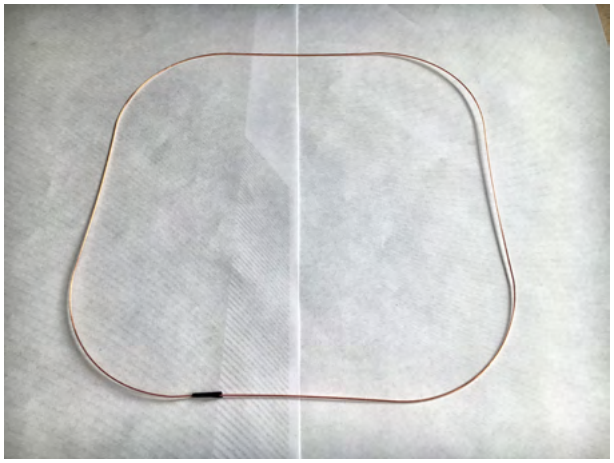


Abb. 5.6: Geschlossenes und gleichförmiges Viereck

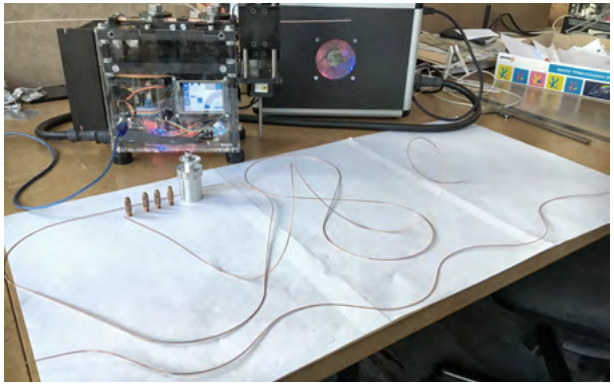


Abb. 5.7: Übersicht aller Testversuche vor dem fertigen WireBender Prototyp.

5.1. *WireBending Resonance*

Im Rahmen einer Einladung zur National Yunlin University of Science and Technology (Taiwan) für die Lehre am internationalen Workshop „Resonance“ konnten die Ziele dieser Arbeit überprüft werden. Der 10-tägige Workshop (22-31 August 2016) wurde von Professoren und Forschern vom MIT (USA), Curtin University (Australien), EPFL Lausanne(Schweiz), Arup (East Asia) und dem Kyoto Institute of Technology besucht, welche ca. 100 Studenten aus diesen Ländern in Arbeitsgruppen betreut haben. In einer dieser Arbeitsgruppen wurden unter Mitwirken des Autors mehrere komplexe Formen aus Beton hergestellt, sowie eine 7 Meter lange und doppelt gekrümmte Mauer errichtet (siehe Kapitel 5.3). Das 3D-Umformen von Metallstäben für die zukünftige Einsatzmöglichkeit als Bewehrung von gekrümmten Formen aus Beton konnte so im Zuge dieser Arbeit ebenfalls getestet werden.

Das Konzept dieser Arbeit, Fachwissen durch eine vereinfachte Schnittstelle zugänglich zu machen, konnte so mithilfe des Prototypen geprüft werden. Das für Taiwan bestellte Biegematerial wurde nach der beschriebenen Methode (siehe Kapitel 3.6.3) für die einfache Verwendung mit der Drahtbiegemaschine vorbereitet. Es wurden Edelstahl und mit Kupfer überzogener Stahl, jeweils mit einem Durchmesser von 2mm, verwendet. Alle Radien wurden durch manuelle Eingabe wieder schrittweise gebogen, bis der kleinstmögliche Radius erreicht wurde. Die eingegebenen Radien und die gemessenen Ergebnisse wurden in einer Exceltabelle ausgewertet und kontrolliert und später als Datensatz in der vereinfachte Schnittstelle in Grasshopper für die Verwendung durch die Studenten bereitgestellt.

In zwei intensiven Tagen wurde die Software Rhinoceros mit Grasshopper und die vereinfachte Schnittstelle WireBender unterrichtet, um ein Basiswissen zu schaffen. Anhand dieser Tools sollten die Studenten ihre ersten Konzepte und Ideen entwerfen, um diese später in einem Design mittels der Drathumformmaschine zu produzieren. Insgesamt wurden sechs kleine Teams gebildet und während der Gruppendiskussionen begleitet. Die Studenten illustrierten ihre ersten Ideen auf Papier und führten später diese Idee in Rhinoceros und Grasshopper in ein 3D-Modell am PC über. Mithilfe der vereinfachten Schnittstelle konnten sie schrittweise ihre Entwürfe überprüfen und anpassen und anschließend die entworfenen Kurven mithilfe der Maschine fabrizieren.

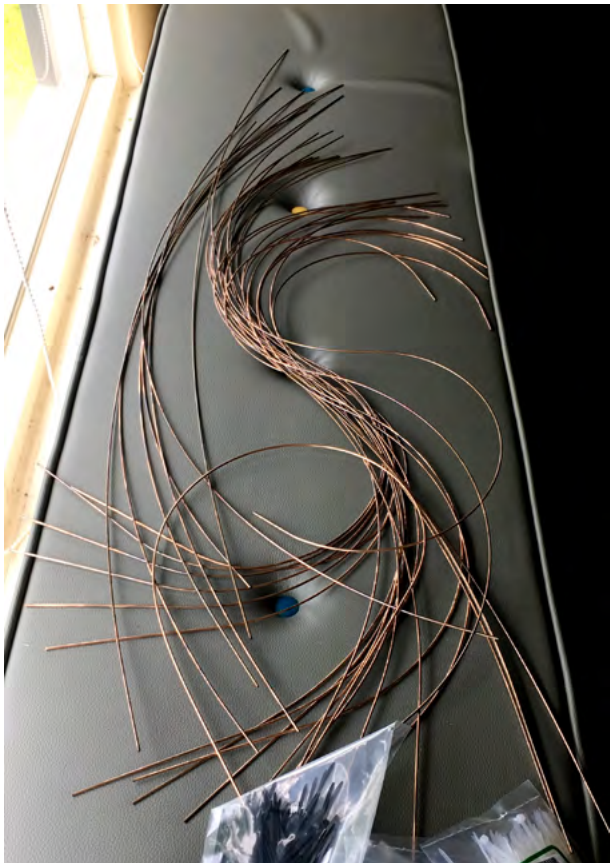
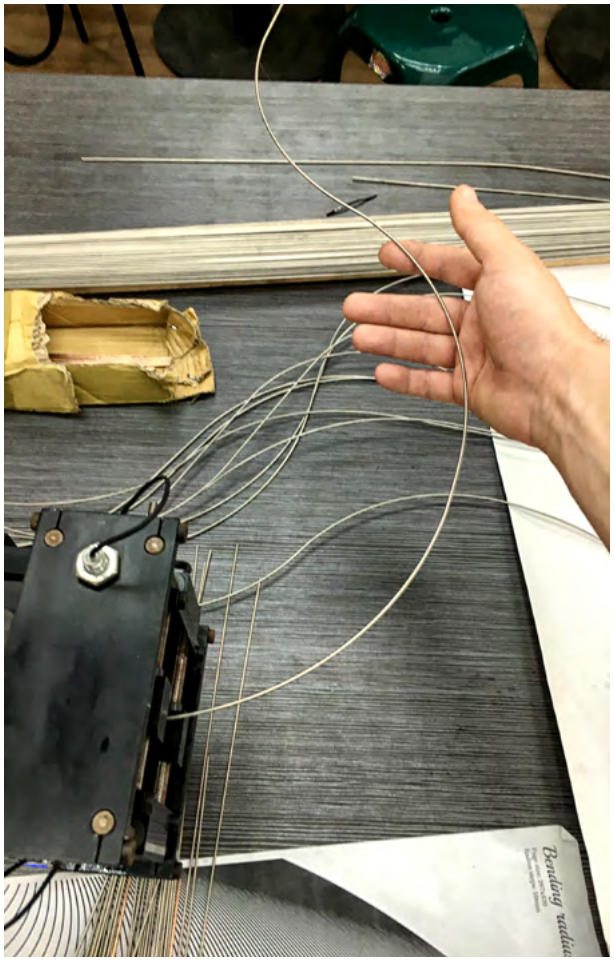


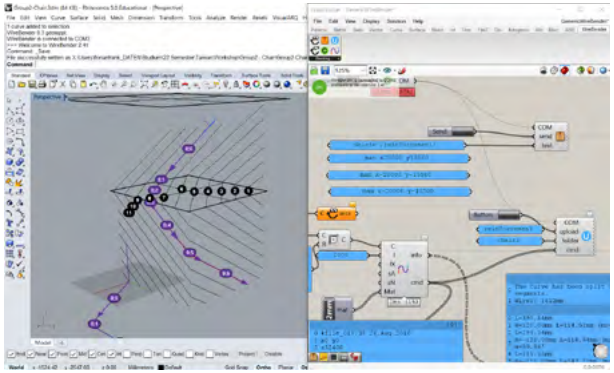
Abb. 5.8: Gebogene Radien der 2mm Schweißstäbe für die Vermessung und Eingabe in die Materialtabelle.

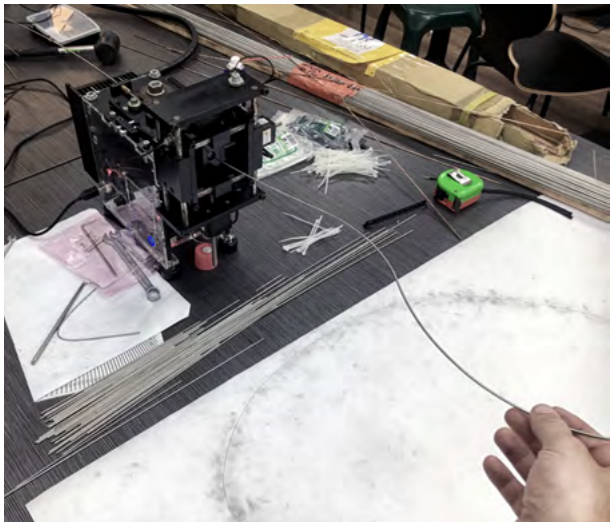
Abb. 5.9: Studenten betrachten und analysieren die gebogenen Stäbe als Inspiration für ihre kommenden Entwürfe,

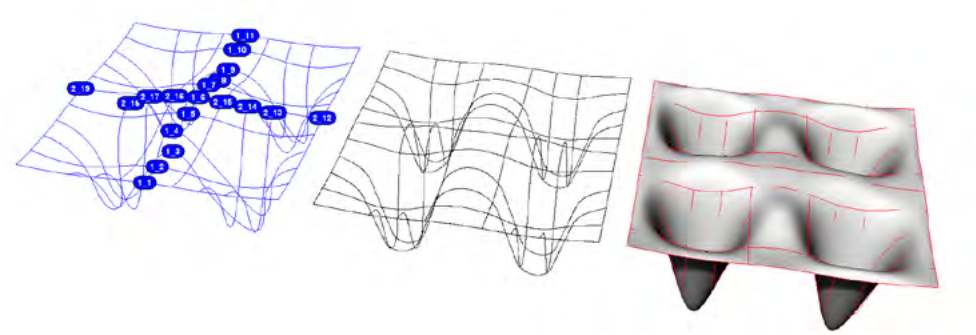
5.1.1. *Team 1*

Die erste Gruppe versuchte die Form eines Sitzmöbels mithilfe von dreidimensionaler Konturbildung als minimalistische Skulptur auszubilden. Es wurde eine Form entwickelt, die dem Referenzobjekt folgt und mithilfe von Grasshopper die Konturen im Raum zugeordnet. Diese wurden anschließend mithilfe der WireBender Komponenten überprüft und angepasst. Nach mehrfacher Iteration wurde das Modell fertiggestellt und alle Kurven mithilfe von RhinoCount nummeriert und in einem 3D-Plan für den Aufbau vorbereitet. Stab für Stab wurde gebogen und mit Nummern versehen, welche jenen des 3D-Modells entsprechen. Anschließend wurde die Abwicklung des Rahmen für die Skulptur aus Acrylglas geschnitten und mit Erhitzung und einer aus MDF gelaserten Schablone geformt. Der Zusammenbau gestaltete sich etwas schwierig, da die Edelstahlstäbe sehr weich sind und durch ihre Länge und der weit entfernten Fixierung nicht genug halt bieten. Zur Stabilisierung mussten deshalb mittels einer dünnen Schnur die Konturen in Position gehalten werden.







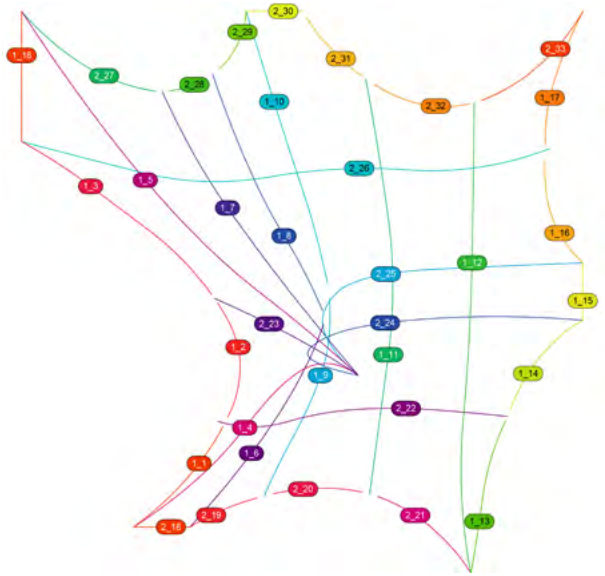


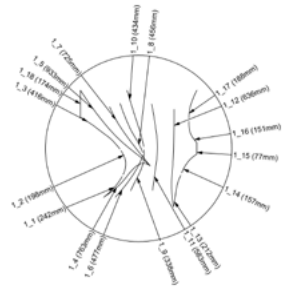
5.1.2. *Team 2*

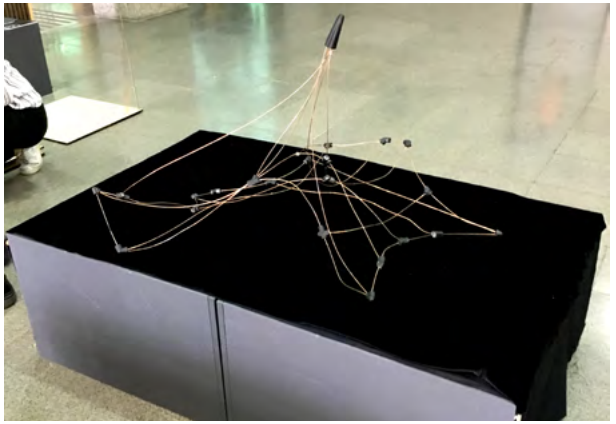
Die Studenten dieser Gruppe haben mithilfe von Rhinoceros eine Freiformfläche entworfen, die später durch gekreuzte Überlappung der gebogenen Stäbe gebildet werden sollte. Ausgangspunkt war eine ebene Fläche, welche im Raum so gekrümmt wurde, dass sich vier Erhebungen ausbilden. Diese dienen als Füße für die entworfene Skulptur. Nachdem die Fläche im ihrer gewünschten Größe und Form im CAD vorlag, machte sich die Gruppe daran, die Fläche durch Kurven nachzubilden. Mithilfe von Grasshopper wurden die Iso-Kurven der Fläche erzeugt und anschließend mit dem WireBender Plugin auf ihre Machbarkeit getestet. Das Design der Fläche wurde angepasst und mit der Feedback-Loop immer wieder überprüft. Nach Beendigung dieses Vorgangs wurden alle Kurven nummeriert und für die Fabrikation ein Plan erstellt und gedruckt. Dieser diente nach erfolgtem Umformen als Hilfe für das spätere Zusammenfügen der geformten Stäbe.

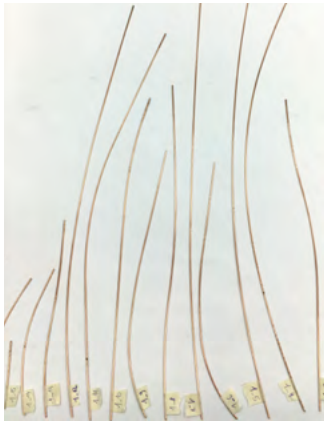
5.1.3. *Team 3*

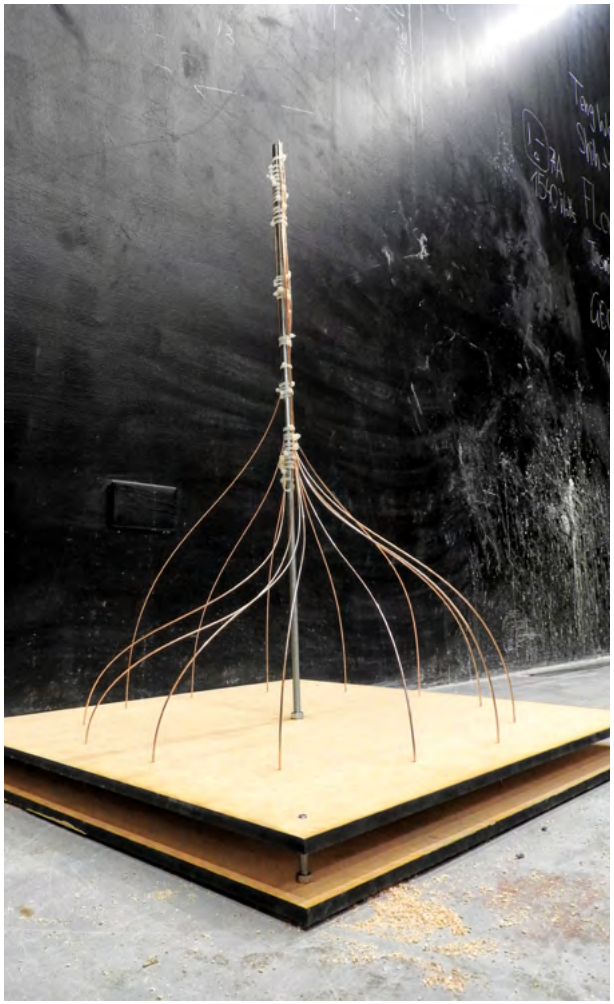
Die Idee dieser Gruppe war es, mit einem Formfindungsverfahren am PC eine Form zu generieren, die später wiederum durch Metallstäbe nachgebildet wurde. Mithilfe von Kangaroo, einem Zusatzmodul für Grasshopper, wurde eine Fläche mit dem sogenannten „Relaxation“ Verfahren zu einer Membrane geformt. Diese bildet eine zeltähnliche Fläche. Ziel dieser Idee war es, eine möglichst natürliche Form, wie sie beim Bespannen eines Gerüstes entsteht, durch die Stäbe selbst nachzuempfinden. Die komplex gekrümmte Geometrie wurde im CAD erstellt und anschließend durch die Kanten der Faltung verlaufenden Kurven nachgebildet. Die Kurven wurden wieder mit der WireBender Schnittstelle geprüft und mit Kangaroo weiter editiert, bis die Biegungen oberhalb des Mindestradius liegen. Sobald der Verlauf der Kurven für die Fabrikation bereit war, wurde ein Farbplan für den komplexen Zusammenbau erstellt. Weiters wurden alle Kurven nummeriert, um später die umgeformten Stäbe im Plan wiederzufinden.











5.1.4. *Team 4*

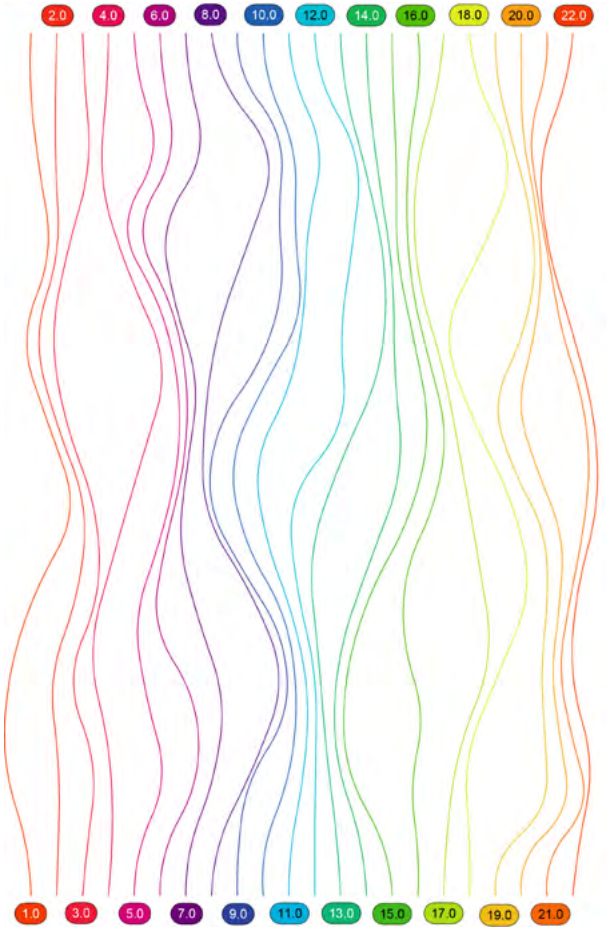
Diese Gruppe konnte unter dem Einsatz der Industrieroboter von YunTech das Umformen von Metallstäben erforschen. Mithilfe von TACO konnten die Studenten innerhalb kürzester Zeit die Roboter selbstständig programmieren und bedienen. Es wurde ein Greifer (engl. Gripper) für diese Gruppe auf dem Roboterarm montiert und die wichtigsten Funktionen erklärt. Ziel dieser Gruppe war es, durch die sehr exakten Bewegungen des Roboters herauszufinden, welchen Einfluss die Länge zwichen den Einspannungen eines Drahtes Auswirkung auf die Form der Biegung haben. Es wurde ein Kreis von geraden Stäben gebildet, welche der Roboter nacheinander in gleichmäßig unterteilten Schritten sukzessive kürzer und kürzer werdend an den im Zentrum des Kreises befestigten Stab biegt. Von Hand wurden die Stäbe nacheinander befestigt und mittels des mit TACO programmierten Ablaufes alle Drähte nacheinander gefertigt. Der Versuch zeigt die unterschiedlichen Krümmungsbelastungen je kürzer der Stab wird. Es handelt sich um eine selbstbildende Form und nicht um eine permanente bzw. plastische Verformung, wie es mit dem WireBender möglich ist. Dieser Versuch ist allerdings eine gute Demonstration dafür, dass das Umformen alleine durch Biegen eine schlechte Verteilung der Belastung im Material zur Folge hätte (siehe Kapitel 3.1). Das Ziel dieser Gruppe war es, eine ästhetische, selbstbildende Form im elastischen Bereich der Verformung zu erstellen, was durchaus gelungen ist.

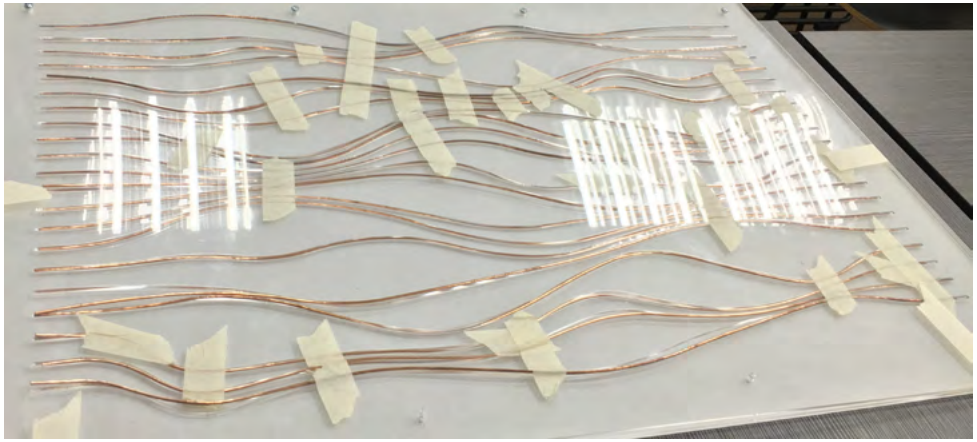
5.1.5. *Team 5*

Bei diesem Entwurf wollte die Studentin eine bereits bestehende Entwurfsmethode, jene der sogenannten Streamlines, an der Umformung von Metall testen. Ein in Python geschriebenes Script wurde von der Studentin adaptiert und an die vom WireBender gemeldeten Problemstellungen angepasst. Die Kurven sollen durch ihre Bewegung ein irreguläres Muster auf der Fläche bilden und durch die programmierte Logik in einer Wechselwirkung gegenseitige An- und Abstoßung erfahren. Die vom Script produzierten Krümmungen erwiesen sich als nicht optimal für die Radien, welche mit dem Material gekrümmt werden konnten. Mithilfe des erstellten Farb- und Nummerierungsplan wurde auch hier versucht das Modell fertigzustellen.

Team 6

Die Studenten dieses Teams verfolgten das Ziel, eine Bewehrung für eine bereits mit den Industrierobotern gedruckte Form aus Beton zu erstellen. Die Schwierigkeit in dieser Aufgabe lag darin, die bestehende Form zu vermessen und eine geeignete Profil-Kurve für die Bewehrung im CAD zu entwerfen. Nachdem die Betonform vermessen wurde, konnte in Rhinoceros eine NURBS-Kurve für das Profil erstellt werden. Mithilfe der WireBender Schnittstelle wurde diese überprüft und adaptiert, bis ein entsprechendes Profil mit der Umformmaschine erstellt werden konnte. Die Kurve musste an der bestehenden Form geprüft werden und konnte nach zwei Durchgängen mit dem Biegeapparat erstellt werden. Eine Serie an Profilkurven wurde erstellt und die Endstücke verlötet. Um eine möglichst stabile Form zu erhalten und den Beton bestmöglich zu armieren, wurden Querstäbe eingesetzt, welche durch eine wellenförmige Aussteifung in Form gehalten wird. Diese dient als statisches Element um die Position der Profilkurven zu erhalten. Auch diese wurden an den Schnittstellen verlötet, um zu einem belastbaren und steifen Element zu werden.





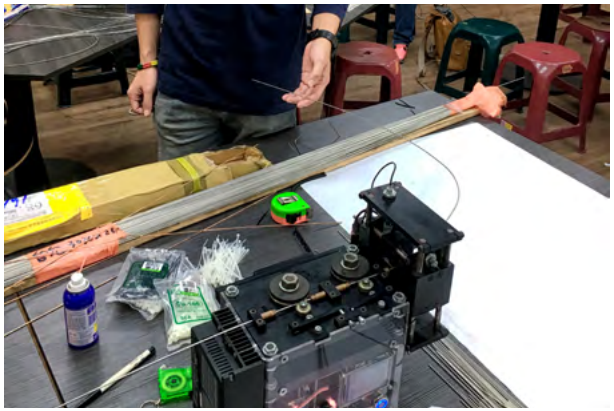


Abb. 5.10: Umformen der Edelstahlstäbe für die Profilkurve der Bewehrung

Abb. 5.11: Herstellung der Aussteifungen

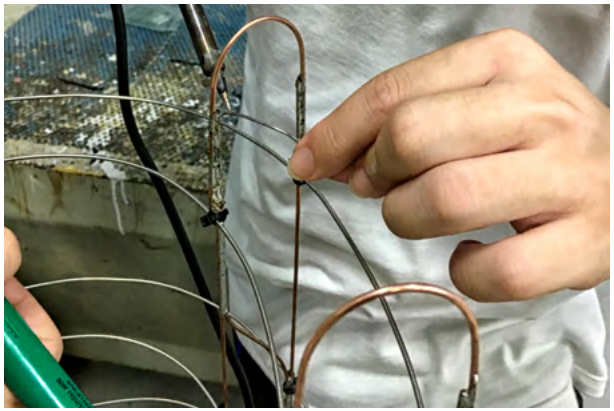


Abb. 5.12: Verlöten der Überlappungen

Abb. 5.13: Gießen des Betons in das Bewehrte Betonteil.

Abb. 5.14: Aus dem Betonteil herausragende Bewehrung im Demonstrationsobjekt.

5.2. TACO für ABB

Die Resultate, die mit der Erweiterung Taco für ABB erzielt wurden können am Besten durch die Verbreitung der Software erklärt werden. Im Vergleich zu anderen erhältlichen ähnlichen Softwarelösungen stellt Taco eine stark vereinfachte Bedienung für die Komplexität der Robotersteuerung zur Verfügung. Aus diesem Grund wurde es schon kurz nach der ersten Veröffentlichung mehrfach heruntergeladen. Bis zum jetzigen Zeitpunkt (6 Monate nach Veröffentlichung) wurden insgesamt ca. 630 Downloads gezählt, die sich aus Bezugsquellen der Software zusammensetzen. Taco wird vor allem an Universitäten eingesetzt. Durch die Verwendung im REX-Lab, dem Robotic experimental laboratory, der Universität Innsbruck, konnte das Plugin auch ständigen Tests unterworfen werden und Verbesserungen bzw. Funktionserweiterungen vorgenommen werden. Die Rückmeldung durch die Studenten, welche diese Software nun in verschiedenen Lehrveranstaltungen an der Fakultät für Architektur Innsbruck nutzen, waren durchwegs positiv und anregend. Vor allem die vergleichende Kritik mit anderen Softwarelösungen bestätigt das Erreichen des Ziels einen vereinfachten Zugang zu dieser Technologie geschaffen zu haben. Die Studenten berichten von einer sehr vereinfachten Arbeitsweise und der Möglichkeit, ihre Ideen in kürzester Zeit umsetzen zu können. Auch sei das Verständnis für diverse fachliche Problematiken durch die Arbeit mit Taco schnell gewachsen.

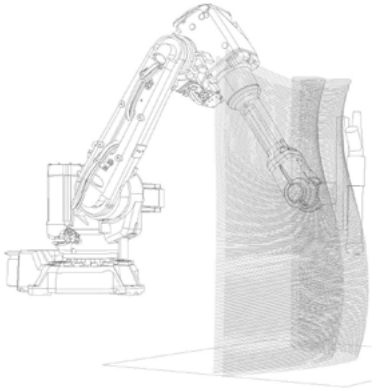
Neben der Nutzung durch viele Studenten an der Universität Innsbruck konnte die Nutzung der Software auch während des Workshop „Resonance“ in Taiwan beobachtet werden. Nach einer kurzen Einführung für die Studenten in die grundlegenden Funktionen von Taco konnten diese innerhalb des Workshops komplexe Roboterbewegungen innerhalb kurzer Zeit programmieren, korrigieren und analysieren. Vor allem die Gruppe mit dem 3D-Betondruck konnte sich innerhalb der 10-Tage tief in die Thematik einarbeiten und umständliche und zeitraubende Vorgehensweisen in der Programmierung der Industrie-Roboter durch Taco umgehen.

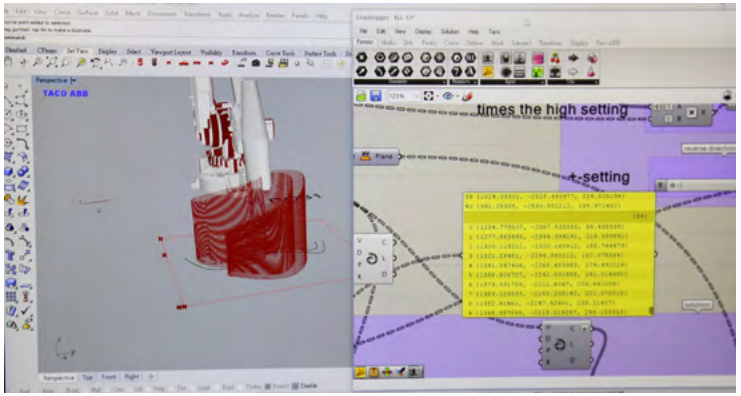
Innerhalb der Monate seit Veröffentlichung und dem Zeitpunkt der Verfassung dieser Arbeit kamen auch diverse Rückmeldungen von Instituten aus verschiedenen Ländern über die Verwendung dieser Software. Durch Rückmeldungen von Forschern und Studenten von Sci-Arc Shanghai (China), EPFL (Schweiz), ETH (Schweiz), FabLab Venedig (Italien), Virginia Tech (USA), Princeton University (USA), GSD Harvard (USA), TU Graz (Österreich), University of Calgary (Kanada), University of Melbourne (Australien) und sogar dem Softwareentwickler von Rhinoceros selbst (McNeel/USA) kann Rückschluss auf das Einsatzgebiet von Taco für Forschungszwecke gemacht werden.

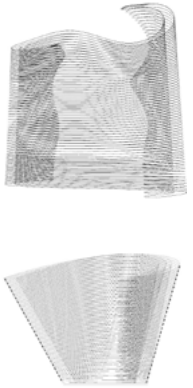
5.3. 3D-Betondruck Resonance

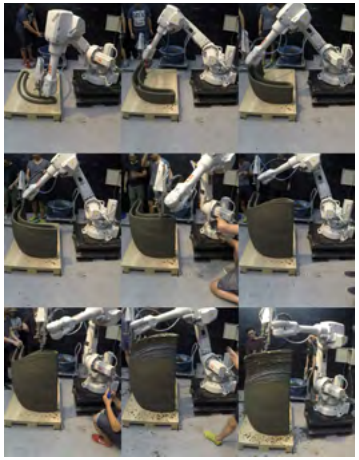
Im Zuge der Entwicklung des Workshop in Taiwan am Anfang dieses Jahres wurde von den Organisatoren der Wunsch geäußert, den Einsatz von Robotern mithilfe von einem neuartigen Betondruckverfahren zu demonstrieren. Mit dieser Methode soll es möglich sein die Roboterbewegung zu nutzen, um Schicht für Schicht Beton durch eine schnelle Aussteifung aufzutragen. Aufgrund des nötigen Fachwissens in Betonkunde erfolgte eine Teambildung innerhalb der Bereiche Bauingenieurwesen und Architektur. Die Beteiligung des Experten Georg Breitenberger in diesem Projekt erwies sich als gelungene Symbiose zwischen Softwareentwicklung, Materialkunde und Verfahrensentwicklung. In einer zweimonatigen Vorbereitungsphase wurden verschiedene Verfahrensmöglichkeiten besprochen und diverse Betonprodukte für Tests bestellt. Die Aufgabe des Studenten im Bauingenieurwesen war es, die Betonprodukte für ein schnelles Auftragen zu optimieren und durch Analyse des Aussteifungsprozesse im Beton eine Kombination von Chemikalien bereitzustellen, mit welcher die gewünschten Ergebnisse erzielt werden konnten. Im Verlauf der Vorbereitungen wurden in Zusammenarbeit verschiedene Werkzeuge für die Montage am Roboterarm gebaut und mit den Produkten getestet. Als Grundlage diente hier das Verfahren von Spritzbeton, welches auf die gewünschten Anforderungen angepasst wurde. Die ersten Ergebnisse wurden analysiert und für die weitere Verwendung geprüft. Eine Kombination von Pumpensystemen und Materialforschung, sowie ein genauer Prozessablauf war das Ergebnis dieser Arbeit. Einige Komponenten des Werkzeugs mussten schon während dieser Vorbereitungsphase in teilweise aufwendigen Fertigungsweisen erstellt werden und für die Verwendung am Workshop in Taiwan vorbereitet werden. Alle weiteren Komponenten konnten durch eine detaillierte Planung während der 8-tägigen Vorbereitungsphase des Workshop, in Taiwan selbst aufgebaut und optimiert werden. Es wurden Standard-Betonprodukte eingekauft und mit den bestellten Chemikalien an der entwickelten Anlage getestet. Der Einsatz von Taco in Verbindung mit den Robotern an der Universität in Taiwan konnte mit diesem Verfahren einige Tage vor dem Start des Workshops getestet und noch einmal verbessert werden. Auch das Werkzeug, welches auf dem Roboterarm montiert wurde, konnte auf die Verwendung optimiert werden und mit dem Einsatz von Farbpigmenten beim Mischen der Betonrezeptur getestet werden.

Zu Beginn des Workshops wurde zusammen mit den Studenten der Umformung von Metallstäben mit dem WireBender Basiswissen über Rhinoceros und Grasshopper unterrichtet. Nach dieser intensiven Unterrichtsphase wurden die Gruppen gebildet und am Entwurf von ersten verschiedenen Versuchsformen beauftragt. Diese konnten mit dem entwickelten Verfahren im Roboterlabor der Yun-Tech produziert werden und auf die Einsatzmöglichkeit dieser Versuche in einem größeren Objekt analysiert werden. Es wurde die Idee stark, die vom Material aufnehmbaren Toleranzen beim Auftragen der Schichten zu nutzen, um eine Musterung des Schichtbildung zu erzielen. Die Studenten entwickelten den Entwurf einer sieben Meter langen Wand, an der die Reliefartige Musterung getestet werden konnte. Die verschiedenen Studentengruppen an Studenten entwarfen und programmierten insgesamt sieben Teile, die zu einem Gesamtobjekt zusammengefügt werden konnten. Die Reliefbildung sollte sich dabei über die gesamte Wand ziehen und die entworfenen Elemente der jeweiligen Gruppen somit als Gesamtbild wirken. Mithilfe von TACO und Rhinoceros konnten die Studenten die Roboterpfade untereinander abstimmen und so zu ihren jeweiligen Objekten einen Gesamtentwurf erzielen. Nacheinander wurden die Elemente mit den von den Studenten erstellten Bewegungen mit Beton gedruckt und am Ende des Workshops zu einer Wand zusammengesetzt. In verschiedenen Phasen des Drucks wurden Farbpigmente in den Beton gemischt, um die Reliefs unter verschiedener Farbgebung zu demonstrieren. Durch den Einsatz des erleichterten Betondruckverfahren und der leichten Bedieung von Taco war es den Studenten möglich, zusammen mit den Instruktoren diese aufwendigen Ziele in wenigen Tagen zu realisieren.




TACO ABB
times the high setting
+setting





Quellverzeichnis

Abb 2.4 071022-Tuecken_der_Luecken.pdf, n.d.

Bartenschlager, J. (Ed.), 2013. Fachkunde Metall, 57., neu bearb. Aufl. ed, Europa-Fachbuchreihe für metalltechnische Berufe. Verl. Europa-Lehrmittel, Haan-Gruiten.

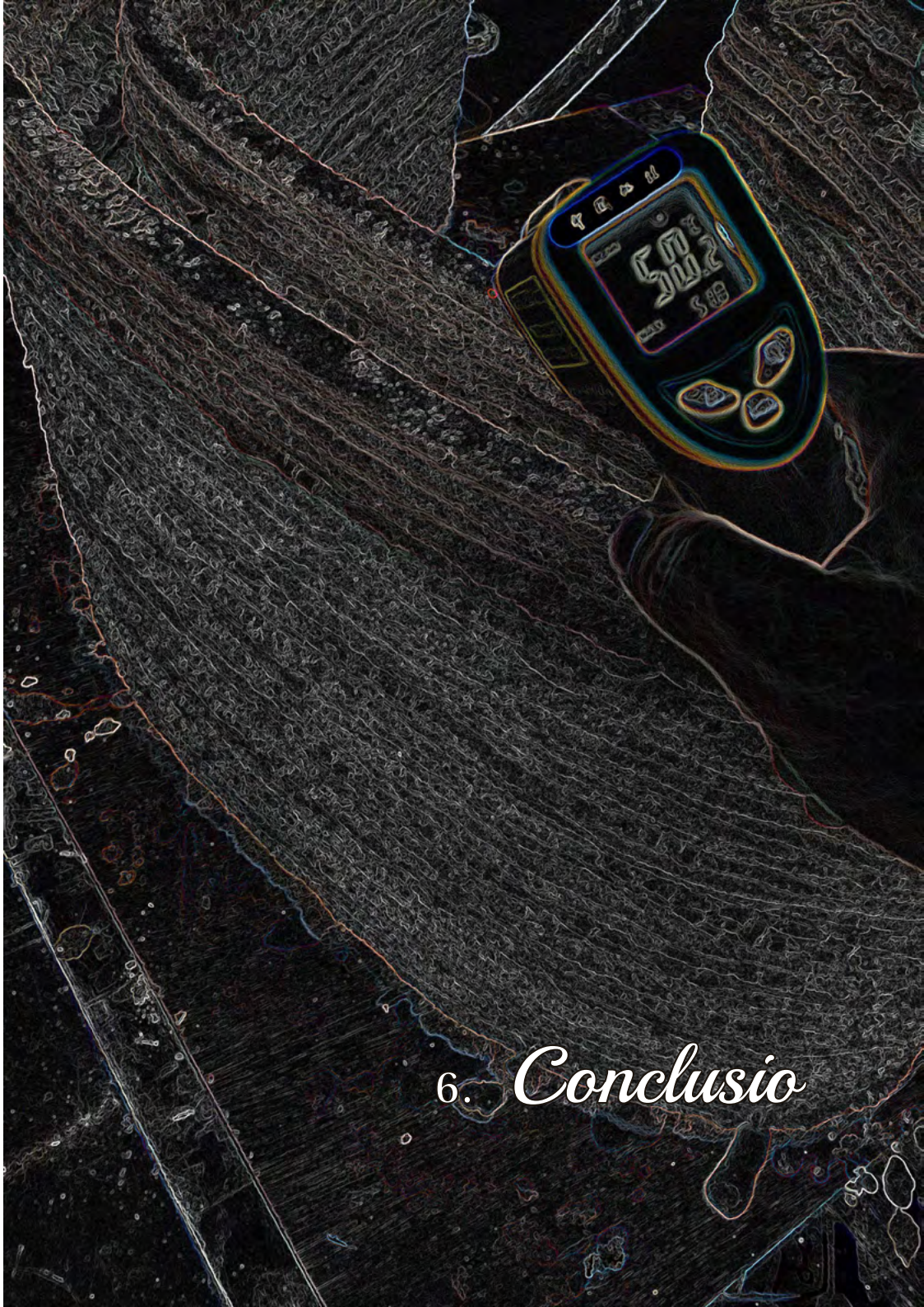
Building Information Modeling (BIM) - AUSTRIAN STANDARDS [WWW Document], n.d. URL https://www.austrian-standards.at/infopedia-themencenter/infopedia-artikel/building-information-modeling-bim/ (accessed 9.11.16).

DIN 8580:2003-09 [WWW Document], n.d. URL http://www.din.de/de/mitwirken/normenausschuesse/natg/normen/wdc-beuth:din21:65031153 (accessed 9.11.16).

Hadid-Projekt bestätigt Architektur-Niveau [WWW Document], n.d. URL https://www.wien.gv.at/rk/msg/2004/0325/023.html (accessed 9.10.16).

Meek, D.S., Walton, D.J., 1995. Approximating smooth planar curves by arc splines. Journal of Computational and Applied Mathematics 59, 221–231.

Mesch, M., 2016. Automatisierung und Beschäftigung: Ein Rückblick aufs 20. Jahrhundert. blog.arbeit-wirtschaft.at.



6. Conclusio

Ziel der Arbeit ist es, durch Verknüpfung von Technologien, (WireBending, Softwareentwicklung) den Gestaltungsspielraum für Architekten und Designer zu erweitern. Die traditionelle Methode des Architekten, durch Verknüpfung von Wissen und Gewerken zu neuen Gestaltungsmöglichkeiten zu gelangen, wurde hier neu interpretiert und erweitert.

Die Umsetzung der Methode in Gestalt des Prototypen „WireBender“ wurde zu einer erfolgreichen Schnittstelle für kreative Ideen. Durch die digital gesteuerte Umlenkung beim Vorschub eines eingespannten Metallstabes können Freiform-Kurven erzeugt werden. Die Umformung mit konstanten Radien führte zu einer exakten Kontrolle der Formgebung und ermöglichte eine gleichmäßige Verteilung der Belastungen im Material. Die Besonderheit dieser Arbeit ist die Verknüpfung der physikalischen Umformung mit der digitalen Entwicklungs- und Simulationsumgebung im 3D Programm zur direkten Kontrolle und in der Gestaltung von Entwürfen. Die persönliche Sicht- und Herangehensweise als Generalist in der Beschäftigung mit der fachlichen Thematik ermöglichte es, einen vereinfachten Zugang zu dieser Technologie zu schaffen. Die Studienarbeiten der Teilnehmer am Workshop „Resonance“ zeigen, dass eine vereinfachte digitale Kontrolle über den Herstellungsprozess den kreativen Gestaltungsraum in der Umsetzung erweitert. Dieser Ansatz ermöglichte den Teilnehmern bereits vorhandene digitale Fertigkeiten in der Umsetzung mit den neuen Methoden zu verknüpfen und ihren individuellen Gestaltungsraum zu erweitern.

Die Anforderungen an die Präzision in der Fertigung des Prototypen wurden anfänglich unterschätzt und resultierten in einer Ungenauigkeit der Umformung. Diese konnte durch die Anpassung der Steuerungselektronik verringert werden. Die Erwartungen vor Beginn dieser Arbeit mussten im Verlauf der Grundlagenforschung eingeschränkt werden, da sich aus technischen und zeitlichen Gründen nicht alle Vorhaben mit der gewünschten Sorgfältigkeit umsetzen lassen würden. Diese Arbeit zukünftig im Team mit Experten fortführen zu können, wäre sicherlich von großem Vorteil. Es könnten Verbesserungen am bestehenden Resultat, sowie weitere Verknüpfungs- und Einsatzmöglichkeiten gefunden werden. Ein interdisziplinäres Projekt könnte durch die Erfahrungen von Experten bereichert werden und die Aneignung von Fachwissen ersetzen, welches Voraussetzung zur Realisierung eines derartigen Projektes durch einen Generalisten wäre. Dies führt schneller zu präzisen Ergebnissen, die ohne Zweifel in eine technologisch verbesserte Gestaltungsmöglichkeit münden und den bisherigen Stand der Technik vielseitig erweitern.

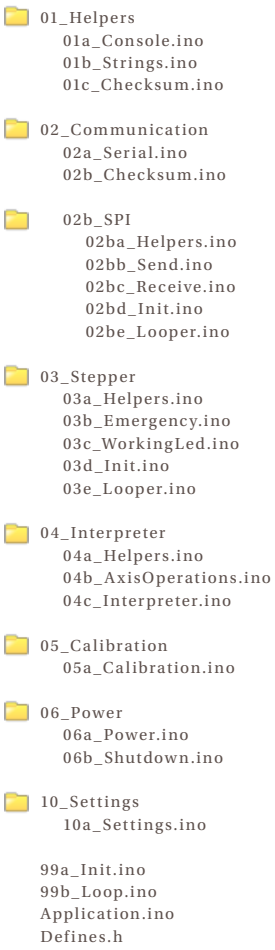
Aus dieser Arbeit resultierende Erfahrungen motivieren zu weiterem Bestreben in der Verknüpfung von Technologien. Zukünftige Projekte sind bereits geplant und sollen zu einer noch weitreichenderen Vernetzung (Wirement) der Rolle des Architekten im kulturellen sowie technologischen Kontext führen.

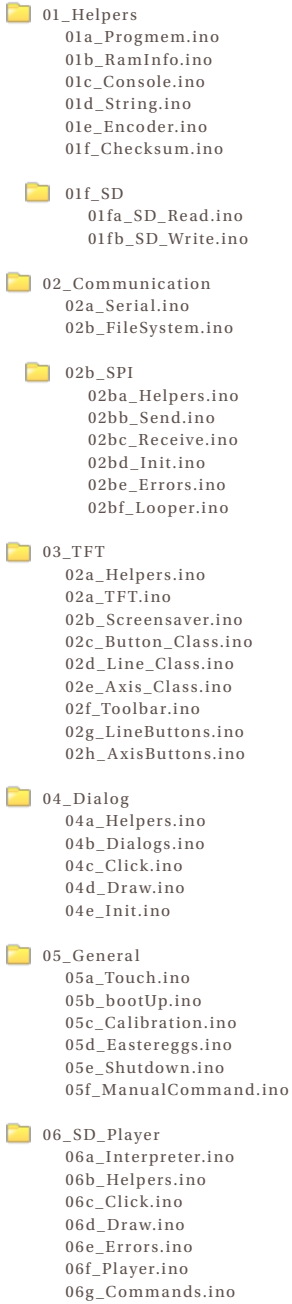
7. Appendix

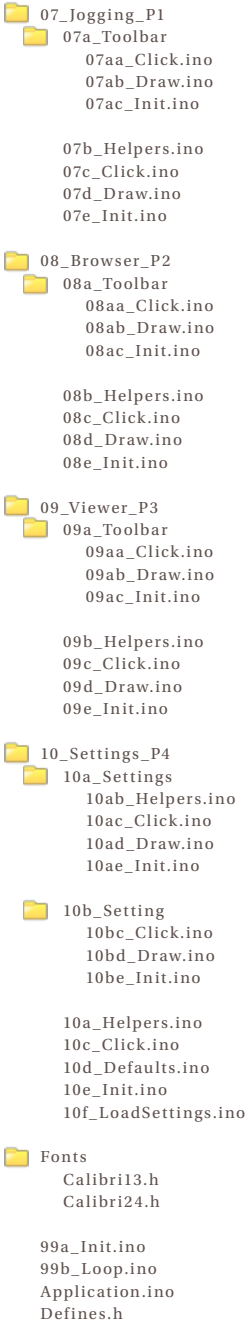
Alle in dieser Arbeit erworbenen Erkenntnisse werden für die Weiterverwendung unter der Creative Commons „by“ (Namensnennung) Lizenz freigegeben.

Die Inhalte des Appendix können nach Veröffentlichung dieser Arbeit auf der Webseite http://wirement.blickfeld7.com heruntergeladen und eingesehen werden.

Have Fun!







7.1. Projekt StepperDriver

01a_Console.ino

StepperDriver\01_Helpers\01a_Console.ino

```
001 void SuspendDebug()
002 {
003 config.debugCache = config.debug;
004 config.debug = false;
005 }
006
007 void ResumeDebug()
008 {
009 config.debug = config.debugCache;
010 }
011
012 size_t Log(String message, const char* v1 = NULL, const char* v2 = NULL)
013 {
014 if(v1 != NULL)
015   Serial.print(message);
016 else
017   return Serial.println(message);
018
019 if(v2 != NULL)
020   Serial.print(v1);
021 else
022   return Serial.println(v1);
023
024 if(v2 != NULL)
025   Serial.println(v2);
026
027 return 0;
028 }
029
030 size_t Debug(String message, const char* v1 = NULL, const char* v2 = NULL)
031 {
032 if(config.debug != 1)
033   return 0;
034
035 return Log(message, v1, v2);
036 }
```

01b_Strings.ino

StepperDriver\01_Helpers\01b_Strings.ino

```
037 bool isInt(String text){ return isNumber(text, false); }
038
039 bool isFloat(String text){ return isNumber(text, true); }
040
041 bool isNumber(String text, bool floating)
042 {
043 if(text.length() < 1)
044   return false;
045
046 // comparer string
047 char comp[text.length()];
048 bool skip = true;
049 int j = 0;
050 for(int i = 0; i < text.length(); i++)
051 {
052   if(text[i] != 48)
053     skip = false;
054
055   if(!skip)
056     comp[j++] = text[i];
057 }
058 comp[j] = 0;
059 String numStr = String(comp);
060 numStr.trim();
061
062 //Debug(String(F("Clean:"))+numStr);
063
064 if(floating)
065 {
066   String comparer = numStr;
067
068   if( numStr.indexOf(DOT) < 0 )
069     comparer = comparer+String(F(".00"));
070
071   if( comparer.indexOf(DOT) == 0 )
072     comparer = String(ZERO)+comparer;
073
074   if( comparer.indexOf(DOT) == comparer.length() - 1 )
075     comparer = comparer + String(F("00"));
076
077   if( comparer.indexOf(DOT) == comparer.length() - 2 )
078     comparer = comparer + String(ZERO);
079
080   float num = text.toFloat();
081   //Debug("isFloat:"+String(num)+String(SPACE)+comparer);
082   if( comparer.length() > 0 && String(num) != comparer )
083     return false;
084 }
085 else
086 {
087   String comparer = (numStr.length() < 1) ? String(ZERO) : numStr;
088   int32_t num = text.toInt();
089   //Debug("isInt:"+String(num)+String(SPACE)+comparer);
090   if( comparer.length() > 0 && String(num) != comparer )
091     return false;
092 }
093
094 return true;
095 }
```

01c_Checksum.ino

StepperDriver\01_Helpers\01c_Checksum.ino

```
096 // Fletcher's 16 Checksum
097
098 uint16_t checksum( uint8_t const *data, size_t bytes )
099 {
100 uint16_t sum1 = 0xff, sum2 = 0xff;
101 size_t tlen;
102
103 while (bytes) {
104   tlen = bytes >= 20 ? 20 : bytes;
105   bytes -= tlen;
106   do {
107     sum2 += sum1 += *data++;
108   } while (--tlen);
109   sum1 = (sum1 & 0xff) + (sum1 >> 8);
110   sum2 = (sum2 & 0xff) + (sum2 >> 8);
111 }
112 /* Second reduction step to reduce sums to 8 bits */
113 sum1 = (sum1 & 0xff) + (sum1 >> 8);
114 sum2 = (sum2 & 0xff) + (sum2 >> 8);
115 return sum2 << 8 | sum1;
116 }
117
118 uint16_t checksum(const char* data, size_t bytes)
119 {
120 return checksum(reinterpret_cast<const uint8_t*>(data), bytes);
121 }
122
123 uint16_t checksum( String text )
124 {
125 const uint8_t* data = reinterpret_cast<const uint8_t*>(text.c_str());
126 size_t bytes = text.length();
127 return checksum(data, bytes);
128 }
```

02a_Serial.ino

StepperDriver\02_Communication\02a_Serial.ino

```
129 void serialLooper()
130 {
131 if(!Serial.available())
132   return;
133
134 delay(128);
135
136 String msg = Serial.readStringUntil('\n');
137
138 //Debug(String(msg.length())+String(F(",bytes received over Serial.")));
139
140 if(msg[0] == -1)
141 {
142   Log(String(F("Welcome to WireDriver ,"))+String(VERSION));
143   Serial.flush();
144   return;
145 }
146
147 msg.trim();
148
149 if(msg == F("on"))
150   PowerOn();
151
152 else if(msg == F("off"))
153   Shutdown();
154
155 else if(msg == F("reset"))
156   resetStepperPost();
157
158 else if(msg == F("status"))
159   DebugPosition();
160
161 else if(msg == F("calibrate"))
162   Calibrate(true);
163
164 else if(msg.startsWith(F("test:")))
165   Debug(F("Checksum:"), String(checksum(msg.substring(5))).c_str());
166
167 else if(msg.startsWith(F("out ,")))
168   Post(msg.substring(4, msg.length()));
169
170 else if(config.isPlaying)
171   Debug(F("Driver is busy"));
172
173 else if(msg.startsWith(F("set:")))
174   Setting(msg);
175
176 else
177 {
178   msg.toLowerCase();
179   Interprete(msg);
180 }
181
182 Serial.flush();
183 }
```

02b_Checksum.ino

StepperDriver\02_Communication\02b_Checksum.ino

```
184 uint16_t fletcher16( String text, size_t bytes )
185 {
186 const uint8_t* data = reinterpret_cast<const uint8_t*>(text.c_str());
187 uint16_t sum1 = 0xff, sum2 = 0xff;
188 size_t tlen;
189
190 while (bytes) {
191   tlen = bytes >= 20 ? 20 : bytes;
192   bytes -= tlen;
193   do {
194     sum2 += sum1 += *data++;
195   } while (--tlen);
196   sum1 = (sum1 & 0xff) + (sum1 >> 8);
197   sum2 = (sum2 & 0xff) + (sum2 >> 8);
198 }
199 /* Second reduction step to reduce sums to 8 bits */
200 sum1 = (sum1 & 0xff) + (sum1 >> 8);
201 sum2 = (sum2 & 0xff) + (sum2 >> 8);
202 return sum2 << 8 | sum1;
203 }
```


02ba_Helpers.ino

StepperDriver\02_Communication\02b_SPI\02ba_Helpers.ino

```
204 bool isReady()
205 {
206 return !spiState.processIt && !spiState.reading && !spiState.sending;
207 }
208
209 bool PostError(uint8_t errorNumber)
210 {
211 errno = errorNumber;
212 resetSPI();
213 delayMicroseconds(2000);
214
215 if(errorNumber > 0)
216   Debug(„PostError:"+String(errorNumber));
217
218 delayMicroseconds(2000);
219 return errorNumber % 10 < 2;
220 }
221
222 bool PostError(uint8_t errorNumber, String text)
223 {
224 PostError(errorNumber);
225
226 if(text.length() > 0)
227   Post(text);
228
229 return errorNumber % 10 < 2;
230 }
231
232 bool PostPosition()
233 {
234 String pos =   String(F(„POS:x"))+
235         String(xAxis.currentPosition())+
236         String(F(„ y"))+
237         String(yAxis.currentPosition());
238
239 return PostError(0, pos);
240 }
```

02bb_Send.ino

StepperDriver\02_Communication\02b_SPI\02bb_Send.ino

```
241 void TriggerSend()
242 {
243 digitalWrite(MASTER_SELECT, HIGH);
244 delayMicroseconds(WAIT);
245 digitalWrite(MASTER_SELECT, LOW);
246 }
247
248 bool Post(String text)
249 {
250 if(!isReady())
251 {
252   Log(F("„SPI is not ready."));
253   return false;
254 }
255
256 if(text.length() < 1)
257   return false;
258
259 resetSPI();
260 dataLength = min(text.length(), SPI_BUFFER);
261
262 while(pos < dataLength)
263   buf[pos] = text[pos++];
264
265 uint16_t sum = checksum(text);
266 high_byte = highByte(sum);
267 low_byte = lowByte(sum);
268
269 Debug(String(F("„OUT„"))+String(dataLength)+String(F("„):"))+text+String(F("„ H"))+highBy-
    te(sum)+String(F("„ L"))+lowByte(sum));
270 pos = postAttempts = 0;
271
272 TriggerSend();
273 return true;
274 }
275
276 void Send(uint8_t in)
277 {
278 if(in != ENQ)
279 {
280   resetSPI();
281
282   if(in != ACK && postAttempts < 5)
283   {
284     Debug(String(F("„Sending over SPI failed on attempt #"))+String(postAttempt-
    s+1)+String(F("„ Code:"))+String(in));
285     postAttempts++;
286     delay(500);
287     TriggerSend();
288   }
289 }
290
291 if(buf[pos] >= 9 && pos < min(dataLength, SPI_BUFFER))
292   setState(buf[pos++]);
293 else if(!spiState.highByte)
294 {
295   setState(high_byte);
296   spiState.highByte = true;
297 }
298 else
299 {
300   setState(low_byte);
301   spiState.highByte = false;
302 }
303 }
304
305 void setState(uint8_t ack)
306 {
307 SPDR = ack;
308 }
```

02bc_Receive.ino

StepperDriver\02_Communication\02b_SPI\02bc_Receive.ino

```
309 // SPI interrupt routine
310 ISR (SPI_STC_vect)
311 {
312 byte c = SPDR;
313
314 // IDLE
315 if(spiState.processIt)
316   return setState(IDLE);
317
318 // RESET
319 else if(c == RESET)
320 {
321   resetSPI();
322   return setState(ACK);
323 }
324
325 // SENDING
326 if(spiState.sending)
327   return Send(c);
328
329 // READING
330 if(spiState.reading)
331 {
332   // Getting Length byte
333   if( spiState.nextByteIsLength )
334   {
335     spiState.nextByteIsLength = false;
336     dataLength = c;
337   }
338
339   // EOT (end of transmission)
340   else if (c == EOT)
341     spiState.highByte = true;
342
343   // HighByte
344   else if(spiState.highByte)
345   {
346     high_byte = c;
347     spiState.highByte = false;
348     spiState.lowByte = true;
349   }
350
351   // LowByte
352   else if(spiState.lowByte)
353   {
354     low_byte = c;
355     spiState.lowByte = spiState.reading = false;
356     spiState.processIt = true;
357   }
358
359   // buffer data
360   else if( pos < SPI_BUFFER )
361     buf[pos++] = c;
362
363   return setState(pos);
364 }
365
366 // SOH (start of transmission)
367 if ( c == SOH )
368 {
369   spiState.reading = true;
370   spiState.nextByteIsLength = true;
371 }
372
373 // STX (start of text)
374 else if( c == STX )
375 {
376   spiState.sending = true;
377   return setState(dataLength);
378 }
379
380 // Return error
381 else if( c == IDLE )
382   return setState(errno);
383
384 // Return ACK for Ready
385 return setState(ACK);
386 }
```

02bd_Init.ino

StepperDriver\02_Communication\02b_SPI\02bd_Init.ino

```
387 void initSPI()
388 {
389 // pin for response to master
390 pinMode(MASTER_SELECT, OUTPUT);
391 digitalWrite(MASTER_SELECT, LOW);
392
393 // have to send on master in, *slave out*
394 pinMode(MISO, OUTPUT);
395
396 // turn on SPI in slave mode
397 SPCR |= _BV(SPE);
398
399 // now turn on interrupts
400 SPI.attachInterrupt();
401 Log(F("„Init SPI Slave"));
402 }
403
404 void resetSPI()
405 {
406 //Debug(„resetSPI");
407 pos = 0;
408 dataLength = 0;
409 high_byte = 0;
410 low_byte = 0;
411 spiState.processIt = false;
412 spiState.reading = false;
413 spiState.sending = false;
414 spiState.lowByte = false;
415 spiState.highByte = false;
416 }
```

02be_Looper.ino

StepperDriver\02_Communication\02b_SPI\02be_Looper.ino

```
bool spiLooper(bool force)
{
if(!spiState.processIt && !force)
  return false;

buf [pos] = 0;
String command = buf;
command.toLowerCase();

if(command.length() < 1)
{
  resetSPI();
  return false;
}

word sum = checksum(command);
Log(String(F("„IN:")+command+String(F("„ H"))+highByte(sum)+String(F("„ L"))+lowByte(sum));
errno = 20;

// Error Checksum
if(highByte(sum) != high_byte || lowByte(sum) != low_byte)
  return PostError(6);

// Parse Command
else if(command == F("„reset"))
  return resetStepperPost();
else if(command == F("„restart"))
  return PostError(0, F("„RESTART"));
else if(command == F("„calibrate"))
  return Calibrate();
else if(command.startsWith(F("„set:")))
  return Setting(command);
else if(command == F("„shutdown"))
  return Shutdown();
else if(command == F("„parking_position"))
  return ParkingPosition();
else if(command == F("„get_pos"))
  return PostPosition();
else if(command == F("„shutdown_succeeded"))
  return ShutdownSucceeded();
else if(command == F("„hello world"))
  return PostError(0, F("„hello world"));
else if(config.isPlaying)
  return PostError(55);
else
  return Interprete(command);

resetSPI();
return true;
}

void spiLooper()
{
spiLooper(false);
}
```

03a_Helpers.ino

StepperDriver\03_Stepper\03a_Helpers.ino

```
void LogPosition()
{
Log(//lZeros(lastBusyLine)+
    String(F("„xPos „"))+
    String(xAxis.currentPosition())+
    String(DBLPT)+
    String(xSpeed)+
    String(SLASH)+
    String(xAxis.acceleration())+
    String(F("„ yPos „"))+
    String(yAxis.currentPosition())+
    String(DBLPT)+
    String(ySpeed)+
    String(SLASH)+
    String(yAxis.acceleration())+
    String(F("„ zPos „"))+
    String(zAxis.currentPosition())+
    String(DBLPT)+
    String(zSpeed)+
    String(SLASH)+
    String(zAxis.acceleration()));
}

void DebugPosition()
{
if(config.debug)
  LogPosition();
}
```

03b_Emergency.ino

StepperDriver\03_Stepper\03b_Emergency.ino

```
void initEmergency()
{
pinMode(EX_MIN, INPUT);
//digitalWrite(EX_MIN, HIGH);

pinMode(EX_MAX, INPUT);
//digitalWrite(EX_MAX, HIGH);

pinMode(EY_MIN, INPUT);
//digitalWrite(EY_MIN, HIGH);

pinMode(EY_MAX, INPUT);
//digitalWrite(EY_MAX, HIGH);
}

void ImmediateStop(bool xaxis)
{
if(xaxis)
{
  xAxis.move(0);
  xAxis.setSpeed(0);
  xAxis.runSpeed();
}
else
{
  yAxis.move(0);
  yAxis.setSpeed(0);
  yAxis.runSpeed();
}
}

bool EmergencyStop(uint8_t i)
{
// Stop immediately
if(i == 0 || i == 1)
  ImmediateStop(true);
else if(i == 2 || i == 2)
  ImmediateStop(false);
else
{
  ImmediateStop(true);
  ImmediateStop(false);
}

config.emergencyStop = true;

// Post Error to Controller
if(i == 0)
  PostError(53, F("„E-STOP X_MIN"));
else if(i == 1)
  PostError(53, F("„E-STOP X_MAX"));
else if(i == 2)
  PostError(53, F("„E-STOP Y_MIN"));
else if(i == 3)
  PostError(53, F("„E-STOP Y_MAX"));
else
  PostError(53, F("„E-STOP UNKNOWN"));

return true;
}

bool EmergencyLooper()
{
bool xmin = digitalRead(EX_MIN) == LOW;
bool xmax = digitalRead(EX_MAX) == LOW;
bool ymin = digitalRead(EY_MIN) == LOW;
bool ymax = digitalRead(EY_MAX) == LOW;

// Immediate Stop wenn ein Emergency Schalter gedr?ckt wurde.
if( !config.emergencyStop )
{
  if(xmin)
    EmergencyStop(0);

  if(xmax)
    EmergencyStop(1);

  if(ymin)
    EmergencyStop(2);

  if(ymax)
    EmergencyStop(3);

  return xmin || xmax || ymin       || ymax;
}

// Wenn alle Emergency Schalter auf aus sind.
if( !xmin && !xmax && !ymin && !ymax )
{
  config.emergencyStop = false;
  return false;
}

// R?ckgabe false, nur wenn neues Ziel in der Gegenrichtung liegt
bool runXmin = xAxis.targetPosition() < xAxis.currentPosition();
bool runXmax = xAxis.targetPosition() > xAxis.currentPosition();
bool runYmin = yAxis.targetPosition() < yAxis.currentPosition();
bool runYmax = yAxis.targetPosition() > yAxis.currentPosition();

// Entweder: MIN & richtungMax ODER MAX & richtungMIN oder kein MIN & kein MAX gedr?ckt
bool xOK = (xmin && runXmax) || (xmax && runXmin) || (!xmin && !xmax);
bool yOK = (ymin && runYmax) || (ymax && runYmin) || (!ymin && !ymax);

if(xOK && yOK)
  return false;

return true;
}
```

03c_WorkingLed.ino

StepperDriver\03_Stepper\03c_WorkingLed.ino

```
void initWorking()
{
// Working LED
pinMode(WORKING_LED, OUTPUT);
digitalWrite(WORKING_LED, LOW);
}

void WorkingLED(bool on)
{
if(on)
  digitalWrite(WORKING_LED, HIGH);
else
  digitalWrite(WORKING_LED, LOW);
}

void workingLedLooper()
{
if(!config.ready)
  return WorkingLED(false);

if(config.emergencyStop)
{
  WorkingLED(millis() % 500 <= 250); //Blink every 250ms
  return;
}

bool status = xAxis.isRunning() || yAxis.isRunning() || zAxis.isRunning();
if(status == digitalRead(WORKING_LED))
  return;

WorkingLED(status);
}
```

03d_Init.ino

StepperDriver\03_Stepper\03d_Init.ino

```
void initStepper()
{
xAxis.setAcceleration(MAX_SPEED);
xAxis.setSpeed(MAX_SPEED);
xSpeed = MAX_SPEED;
//xAxis.setMaxSpeed(MAX_SPEED);
//xAxis.moveTo(0);

yAxis.setAcceleration(MAX_SPEED);
yAxis.setSpeed(MAX_SPEED);
ySpeed = MAX_SPEED;
//yAxis.setMaxSpeed(MAX_SPEED);
//yAxis.moveTo(0);

zAxis.setAcceleration(MAX_ACCEL);
zAxis.setSpeed(MAX_SPEED);
zSpeed = MAX_SPEED;
//zAxis.setMaxSpeed(MAX_SPEED);
//zAxis.moveTo(py);
}

bool resetStepperPost()
{
resetStepper();
return config.isPlaying ? PostError(54, „RESET") : PostError(20);
}

void resetStepper()
{
resetStepper(true);
}

void resetStepper(bool status)
{
xAxis.stop();
yAxis.stop();
zAxis.stop();

if(status)
{
  DebugPosition();
  //Debug(F("„Reset Driver"));
  Debug(F("„------------------------"));
}
lastBusyLine = 0;
config.ackPending = false;
}
```

03e_Looper.ino

StepperDriver\03_Stepper\03e_Looper.ino

```
void stepperLooper()
{
if(config.emergencyStop)
  return;

bool xRun = xAxis.run();
bool yRun = yAxis.run();
bool zRun = zAxis.run();

config.isPlaying = xRun || yRun || zRun;

if(config.isPlaying)
  return;

if(config.shutDown && config.ackPending)
{
  Post(F("„PARKING_POSITION"));
  config.ackPending = false;
}

if(config.ackPending)
{
  Post(String(F("„ACK"))+String(lastBusyLine));
  config.ackPending = false;
}

return;
}
```

04a_Helpers.ino

StepperDriver\04_Interpreter\04a_Helpers.ino

```
bool SkipLine(uint16_t lineNumber)
{
lastBusyLine = lineNumber;
return PostError(60); //SKIP
}

bool AckLine(uint16_t lineNumber)
{
lastBusyLine = lineNumber;
config.ackPending = true;
return PostError(61); //OK
}

bool NackLine(uint16_t lineNumber)
{
lastBusyLine = lineNumber;
return PostError(66);
}
```

04b_AxisOperations.ino

StepperDriver\04_Interpreter\04b_AxisOperations.ino

```
bool parseAxisCommand(String line, String axis)
{
int16_t cmdStart = line.indexOf(axis);

// Check if axis command doesn't exist or empty
if(cmdStart < 0 || line.length() < cmdStart + 1)
  return true;

cmdStart++; //skip axis id char

uint16_t cmdEnd = line.indexOf(SPACE, cmdStart);
if(cmdEnd < 0)
  cmdEnd = line.length();

// Check if axis command is empty
String command = line.substring(cmdStart, cmdEnd);
if(command.length() < 1)
  return true;

// Axis pointer for subparser
AccelStepper* axisPtr = NULL;
if(axis == XCHAR)
  axisPtr = &xAxis;
else if(axis == YCHAR)
  axisPtr = &yAxis;
else if(axis == ZCHAR)
  axisPtr = &zAxis;

// Subparser
return  parseAccelCommand(command, axisPtr) &&
    parseSpeedCommand(command, axisPtr) &&
    parsePosCommand(command, axisPtr);
}

bool parseAccelCommand(String command, AccelStepper* axis){ return parseCommand(command, axis, SLASH); }

bool parseSpeedCommand(String command, AccelStepper* axis){ return parseCommand(command, axis, DBLPT); }

bool parsePosCommand(String command, AccelStepper* axis){ return parseCommand(command, axis, EMPTY); }

bool parseCommand(String command, AccelStepper* axis, String cmdChar)
{
int16_t separator = (cmdChar == EMPTY) ? 0 : command.indexOf(cmdChar);

// Check if no accel command or accel cmd is empty
if(separator < 0 || command.length() < separator + 1)
  return true;

// Skip separator char
if(cmdChar != EMPTY)
  separator++;

// Check for separator (end of command value)
int16_t ende = command.indexOf(SPACE, separator);
int16_t dblpt = command.indexOf(DBLPT, separator);
int16_t slash = command.indexOf(SLASH, separator);

ende = ( ende >= 0 && dblpt >= 0 ) ? min(ende, dblpt) : max(ende, dblpt);
ende = ( ende >= 0 && slash >= 0 ) ? min(ende, slash) : max(ende, slash);

if(ende < 0)
  ende = command.length();

// Check if command value is empty
String valStr = command.substring(separator, ende);
if(valStr.length() < 1)
  return true;

// Check if command value is valid
if(cmdChar == SLASH)
  return SetAcceleration(valStr, axis);
else if(cmdChar == DBLPT)
  return SetSpeed(valStr, axis);
else if(cmdChar == EMPTY)
{
  if(!isInt(valStr))
```

```
bool isValidLineNumber(uint16_t lineNumber)
{
if(lineNumber == 0)
  return true;

if(lineNumber <= lastBusyLine)
  return false;

return true;
}

bool isValidCode(String line)
{
int16_t space = line.indexOf(SPACE);

// Check if empty line
if(line.length() < space + 2)
  return true;

// Check if comment
if( line.substring(space + 1, space + 2) == String(HASHTAG) )
  return true;

// Check if line contains only valid chars
String validChars = VALID_CHARS;
for(uint8_t i = 0; i < line.length(); i++)
{
  bool valid = false;
  for(uint8_t j = 0; j < validChars.length(); j++)
    if(line[i] == validChars[j])
      valid = true;

  if(!valid)
    return false;
}

return true;
}
```

```
848       return PostError(65);
849
850     if(axis != NULL)
851       if(axis == &zAxis)
852         axis->move(valStr.toInt());
853       else
854         axis->moveTo(valStr.toInt());
855   }
856
857   return true;
858   }
```

04c_Interpreter.ino

StepperDriver\04_Interpreter\04c_Interpreter.ino

```
859   bool Interprete(String line)
860   {
861   line.trim();
862   line.toLowerCase();
863
864   if(line.length() < 1)
865     return true;
866
867   if(line[0] == -1)
868   {
869     return PostError(0, „Connected to WireBender Driver.“);
870   }
871
872   if(!isValidCode(line))
873   {
874     /* //Debug Purpose:
875     Serial.print(„Interpreter:“);
876     int i = 0;
877     while(i < line.length())
878     {
879       Serial.print(line[i++],DEC);
880       Serial.print(„ | „);
881     }
882     */
883     return PostError(62);
884   }
885
886   // Check for lineNumber
887   int16_t space = line.indexOf(SPACE);
888   line.toLowerCase();
889
890   String numStr = (space < 1) ? line : line.substring(0, space);
891
892   // Check if LineNumber is correct
893   if(numStr.length() < 1 || !isInt(numStr))
894     return PostError(63);
895
896   uint16_t lineNumber = numStr.toInt();
897   if(!isValidLineNumber(lineNumber))
898     return PostError(64);
899
900   // Check if line is empty
901   if( line.length() < space + 2 )
902     return SkipLine(lineNumber);
903
904   // Check if line is comment
905   if( line.substring(space + 1).startsWith(HASHTAG) )
906     return SkipLine(lineNumber);
907
908   // Check if line is Skip
909   if(line.substring(space + 1).startsWith(F(„skip“)))
910     return SkipLine(lineNumber);
911
912   int16_t xChar = line.indexOf(XCHAR);
913   int16_t yChar = line.indexOf(YCHAR);
914   int16_t zChar = line.indexOf(ZCHAR);
915
916   // Check if X, Y or Z axis command exists
917   if( xChar < 0 && yChar < 0 && zChar < 0 )
918     return SkipLine(lineNumber);
919
920   if(
921     parseAxisCommand(line, XCHAR) &&
922     parseAxisCommand(line, YCHAR) &&
923     parseAxisCommand(line, ZCHAR)
924   )
925     return AckLine(lineNumber);
926
927   return false;
928   }
```

05a_Calibration.ino

StepperDriver\05_Calibration\05a_Calibration.ino

```
929   bool Calibrate()
930   {
931     return Calibrate(false);
932   }
933
934   bool Calibrate(bool serial)
935   {
936   if(config.isPlaying)
937     return PostError(55);
938
939   ImmediateStop(true);
940   ImmediateStop(false);
941   resetStepper(true);
942
943   if(serial)
944   {
945     Log(F(„----------“));
946     Log(F(„Calibration started.“));
947     LogPosition();
948   }
949   else
950     PostError(71);
951
952   float xAcc = xAxis.acceleration();
953   xAxis.setSpeed(400.00f);
954   xAxis.setAcceleration(200.00f);
955
956   float yAcc = yAxis.acceleration();
957   yAxis.setSpeed(200.00f);
958   yAxis.setAcceleration(100.00f);
959
960   xAxis.moveTo(-42000);
961   yAxis.moveTo(-24000);
962
963   Post(String(F(„Calibration started with:#X-Pos=“))+String(xOrigin)+String(F(„#X-Pos=“))+Str
      ing(yOrigin));
964
965   bool xmin = digitalRead(EX_MIN) == LOW;
966   bool ymin = digitalRead(EY_MIN) == LOW;
967
968   if(!xmin || !ymin)
969     digitalWrite(WORKING_LED, HIGH);
970
971   while((!xmin || !ymin))
972   {
973     xmin = digitalRead(EX_MIN) == LOW;
974     if(!xmin)
975       xAxis.run();
976
977     ymin = digitalRead(EY_MIN) == LOW;
978     if(!ymin)
979       yAxis.run();
980   }
981
982   xmin = digitalRead(EX_MIN) == LOW;
983   ymin = digitalRead(EY_MIN) == LOW;
984   digitalWrite(WORKING_LED, LOW);
985
986   if(!xmin || !ymin)
987     return Post(String(F(„Calibration failed.#The end switch „)) + String(xmin ? F(„Y-Min“) :
      F(„X-Min“)) + String(F(„ couldn't be reached.“)));
988
989   Post(F(„Both end switches reached.#Running to home position...“));
990
991   xAxis.setCurrentPosition(0);
992   yAxis.setCurrentPosition(0);
993
994   xAxis.setSpeed(xSpeed);
995   xAxis.setAcceleration(xAcc);
996
997   yAxis.setSpeed(ySpeed);
998   yAxis.setAcceleration(yAcc);
999
1000    xAxis.moveTo(xOrigin);
1001    yAxis.moveTo(yOrigin);
1002    bool xrun = true;
1003    bool yrun = true;
1004
1005    digitalWrite(WORKING_LED, HIGH);
1006
1007    while(xrun || yrun)
1008    {
1009      if(digitalRead(EX_MAX) == LOW)
1010        xrun = false;
1011
1012      if(xrun)
1013        xrun = xAxis.run();
1014
1015      if(digitalRead(EY_MAX) == LOW)
1016        yrun = false;
1017
1018      if(ymin)
1019        yrun = yAxis.run();
1020    }
1021
1022    digitalWrite(WORKING_LED, LOW);
1023
1024    xAxis.setCurrentPosition(0);
1025    yAxis.setCurrentPosition(0);
1026
1027    if
1028    (
1029      digitalRead(EX_MIN) == LOW ||
1030      digitalRead(EX_MAX) == LOW ||
1031      digitalRead(EY_MIN) == LOW ||
1032      digitalRead(EY_MAX) == LOW
1033    )
1034      Post(F(„The calibration ended unexpected.#Check the emergency stop switches!“));
1035    else
1036    {
1037      Post(F(„The calibration was successful.“));
1038    }
1039
1040    delay(1000);
1041
1042    return true;
1043  }
```

06a_Power.ino

StepperDriver\06_Power\06a_Power.ino

```
1044  void initPower()
1045  {
1046    // Power Button
1047    pinMode(POWER_BTN, INPUT);
1048    digitalWrite(POWER_BTN, HIGH);
1049
1050    // Power LED
1051    pinMode(POWER_LED, OUTPUT);
1052    analogWrite(POWER_LED, 0);
1053
1054    // Power OFF
1055    config.ready = false;
1056    config.shutDown = false;
1057  }
1058
1059  bool PowerLooper()
1060  {
1061    if(digitalRead(POWER_BTN) == LOW && millis() - lastPowerButton > MINELAPSEDTIME)
1062    {
1063      lastPowerButton = millis();
1064
1065      if(!config.ready && !config.shutDown)
1066        return PowerOn();
1067      else
1068        return TriggerShutdown();
1069    }
1070
1071    PowerLedLooper();
1072
1073    return !config.ready;
1074  }
1075
1076  bool PowerOn()
1077  {
1078    Log(F(„POWER ON“));
1079    resetStepper();
1080
1081    xAxis.enableOutputs();
1082    yAxis.enableOutputs();
1083    zAxis.enableOutputs();
1084
1085
1086    if(Post(F(„RESTART“)))
1087      config.ready = true;
1088    else
1089      PostError(10, F(„DRIVER_START_FAILED“));
1090
1091    return false;
1092  }
1093
1094  void PowerLedLooper()
1095  {
1096    if(config.ready)
1097    {
1098      if(config.shutDown)
1099        analogWrite(POWER_LED, POWER_PWM * (millis() % 500 <= 800)); //Blink every 800ms
1100      else
1101      analogWrite(POWER_LED, POWER_PWM);
1102    }
1103    else
1104    {
1105      if(config.shutDown)
1106        return analogWrite(POWER_LED, 0);
1107
1108      // BREATH LED
1109      float val = (exp(sin(millis()/1000.0*PI)) - 0.36787944)*108.0;
1110      analogWrite(POWER_LED, constrain(map(val, 0, 255, 0 , POWER_PWM), 0 , POWER_
      PWM));
1111    }
1112  }
```

06b_Shutdown.ino

StepperDriver\06_Power\06b_Shutdown.ino

```
1113  bool TriggerShutdown()
1114  {
1115    return Post(F(„SHUTDOWN“));
1116  }
1117
1118  bool Shutdown()
1119  {
1120    resetStepper(0);
1121    WorkingLED(false);
1122    analogWrite(POWER_LED, 0);
1123    resetStepper(false);
1124
1125    xAxis.disableOutputs();
1126    yAxis.disableOutputs();
1127    zAxis.disableOutputs();
1128
1129    config.shutDown = true;
1130
1131    return PostError(10);
1132  }
1133
1134  bool ShutdownSucceeded()
1135  {
1136    config.ready = false;
1137    config.ackPending = false;
1138    return PostError(10);
1139  }
1140
1141  bool ParkingPosition()
1142  {
1143    config.shutDown = true;
1144    config.ackPending = true;
1145
1146    if(abs(xAxis.currentPosition() - 12000) < 5 && abs(yAxis.currentPosition() - 10000) < 5)
1147      return PostError(10);
1148
1149    xAxis.moveTo(12000);
1150    yAxis.moveTo(10000);
1151
1152    return PostError(11);
1153  }
```

10a_Settings.ino

StepperDriver\10_Settings\10a_Settings.ino

```
1154  bool ParametersAreValid(String val, AccelStepper* axis, bool floating = true)
1155  {
1156    if((floating && !isFloat(val)) || (!floating && !isInt(val)))
1157      return PostError(73, val);
1158
1159    if(axis == NULL)
1160    return PostError(75);
1161
1162    return true;
1163  }
1164
1165  bool Setting(String data)
1166  {
1167  if(!data.startsWith(F(„set:“)))
1168      return PostError(72);
1169
1170    if(data.length() == 4)
1171    {
1172      PostError(70);
1173      return false;
1174  }
1175
1176    int16_t separator = data.indexOf(SEPARATOR);
1177    if(separator < 0 || separator >= 30)
1178    return PostError(72, data);
1179
1180    String name = data.substring(4, separator);
1181    String val = data.substring(separator+1);
1182
1183    if(!isInt(val) && !isFloat(val))
1184      return PostError(73, val);
1185
1186    if(name == F(„debug“))
1187    {
1188      if(isInt(val))
1189      {
1190        config.debug = val.toInt();
1191        Log(String(F(„Debug: „))+String(config.debug));
1192        return PostError(71);
1193      }
1194      else
1195        return PostError(73, val);
1196    }
1197    else if(name == F(„selected line“))
1198      return PostError(70);
1199
1200    AccelStepper* axis = NULL;
1201    if(name.startsWith(XCHAR))
1202      axis = &xAxis;
1203    else if(name.startsWith(YCHAR))
1204      axis = &yAxis;
1205    else if(name.startsWith(ZCHAR))
1206      axis = &zAxis;
1207    else
1208      return PostError(74, name);
1209
1210  if(name.endsWith(F(„speed“)))
1211      return SetSpeed(val, axis);
1212    else if(name.endsWith(F(„max speed“)))
1213      return SetMaxSpeed(val, axis);
1214    else if(name.endsWith(F(„acceleration“)))
1215      return SetAcceleration(val, axis);
1216    else if(name.endsWith(F(„origin“)))
1217    {
1218      if(name.startsWith(XCHAR))
1219        xOrigin = val.toInt();
1220      else
1221        yOrigin = val.toInt();
1222    }
1223    else if(name.endsWith(F(„pos“)) && !name.startsWith(ZCHAR))
1224    {
1225      if(!ParametersAreValid(val, axis))
1226        return false;
1227
1228      axis->setCurrentPosition(val.toInt());
1229      bool spdOK = SetSpeed(String(axis == &xAxis ? xSpeed : ySpeed), axis);
1230      if(abs(axis->currentPosition()) < 5)
1231        axis->moveTo(0);
1232
1233      return spdOK;
1234    }
1235    else
1236      return PostError(74, name);
1237
1238    return PostError(71);
1239  }
1240
1241  bool SetSpeed(String val, AccelStepper* axis)
1242  {
1243    if(!ParametersAreValid(val, axis))
1244      return false;
1245
1246    float v = val.toFloat();
1247    if(v <= 0)
1248      return PostError(70);
1249
1250    if(axis == &xAxis)
1251      xSpeed = v;
1252    else if(axis == &yAxis)
1253      ySpeed = v;
1254    else if(axis == &zAxis)
1255      zSpeed = v;
1256
1257    axis->setSpeed(v);
1258    //Debug(F(„setSpeed: „), String(v).c_str());
1259    return PostError(71);
1260  }
1261
1262  bool SetMaxSpeed(String val, AccelStepper* axis)
1263  {
1264    if(!ParametersAreValid(val, axis))
1265      return false;
1266
1267    if(val.toFloat() <= 0.009)
1268      return PostError(70);
1269
1270    axis->setMaxSpeed(val.toFloat());
1271    //Debug(F(„setMaxSpeed: „), val.c_str());
1272    return PostError(71);
1273  }
1274
1275  bool SetAcceleration(String val, AccelStepper* axis)
1276  {
1277    if(!ParametersAreValid(val, axis))
1278      return false;
1279
1280    if(val.toFloat() <= 0)
1281      return PostError(70);
1282
1283    axis->setAcceleration(val.toFloat());
1284    //Debug(F(„setAcceleration: „), val.c_str());
1285    return PostError(71);
1286  }
```

99a_Init.ino

StepperDriver\99a_Init.ino

```
1287  void setup()
1288  {
1289    //COMMUNICATION
1290    Serial.begin(19200);
1291    Log(String(F(„Welcome to WireDriver „))+String(VERSION));
1292    initSPI();
1293
```

```
  // Power BTN + LED
  initPower();

  // Working LED
  initWorking();

  // Emergency SWITCHES
  initEmergency();

  // Stepper
  initStepper();

  //PowerOn();
  //PostError(0,„RESTART“);
  WorkingLED(false);
}
```

99b_Loop.ino

StepperDriver\99b_Loop.ino

```
void loop() {
  spiLooper();
  serialLooper();
  workingLedLooper();

if(PowerLooper())
    return;

  if(EmergencyLooper())
  return;

  stepperLooper();
}
```

Application.ino

StepperDriver\Application.ino

```
/*********************************************/
/*        INCLUDES          */
/*********************************************/
#include <Defines.h>
#include <SPI/SPI.h>
#include <AccelStepper.h>



/*********************************************/
/*        GLOBALS           */
/*********************************************/
volatile SpiStates spiState = {
  false,   //   reading
  false,   //   sending
  false,   //   nextByteIsLength
  false,   //   processIt
  false,   //   highByte
  false,   //   lowByte
  false,   //   ???
  false  //     ???
};

volatile Configuration config = {
  false,   //   powerOn
  false,   //   ready
  false,   //   isPlaying
  false,   //   emergencyStop
  true,  //     debug
  false,   //   debugCache
  false,   //   ???
  false  //     ???
};

// SPI
char buf[SPI_BUFFER];
volatile uint8_t dataLength;
volatile uint8_t high_byte;
volatile uint8_t low_byte;
volatile uint8_t pos;
volatile uint8_t errno;
volatile uint8_t postAttempts;


// Stepper Driver
AccelStepper xAxis(AccelStepper::DRIVER, X_PUL, X_DIR);  //X-Achse
68bytes
AccelStepper yAxis(AccelStepper::DRIVER, Y_PUL, Y_DIR);  //Y-Achse
68bytes
AccelStepper zAxis(AccelStepper::DRIVER, Z_PUL, Z_DIR);  //Z-Achse
68bytes
static float xSpeed, ySpeed, zSpeed;
static int16_t xOrigin, yOrigin;


// Power
static uint32_t lastPowerButton;

// Interpreter
static uint16_t lastBusyLine;

// Settings
//static uint32_t lastTime;
```

Defines.h

StepperDriver\Defines.h

```
/*
 * Defines.h
 *
 * Created: 19.01.2016 20:49:53
 * Author: Florian Frank
 */
#define VERSION 2.1



/*********************************************/
/*        PINS
          */
/*********************************************/

// Emergency
#define EX_MAX A0
#define EX_MIN A1
#define EY_MIN A2
#define EY_MAX A3

// SPI
//#define CS 10
#define MASTER_SELECT A4

// Working
#define WORKING_LED A5

// Stepper
#define X_DIR 4
#define X_PUL 5
#define Y_DIR 2
#define Y_PUL 3
#define Z_DIR 8
#define Z_PUL 9

// Power
#define POWER_LED 6
#define POWER_BTN 7




/*********************************************/
/*       POWER                */
/*********************************************/
#define POWER_PWM 180
#define MINELAPSEDTIME 800



/*********************************************/
/*       STEPPER             */
/*********************************************/
#define MAX_POS 10000
#define MAX_SPEED 2000
#define MAX_ACCEL 800



/*********************************************/
/*       INTERPRETER          */
/*********************************************/
#define ZEROING 4

#define EMPTY F(„“)
#define SPACE F(„ „)
#define DOT F(„.“)
#define ZERO F(„0“)
#define HASHTAG F(„#“)
#define SEPARATOR F(„=“)
#define SLASH F(„/“)
#define DBLPT F(„:“)

// Interpreter
#define VALID_CHARS F(„-.0123456789#:/ xyz“)
#define XCHAR F(„x“)
#define YCHAR F(„y“)
#define ZCHAR F(„z“)



/*********************************************/
/*       SPI                 */
/*********************************************/
#define SPI_BUFFER 256
#define TIMEOUT 600
#define WAIT 20

#define RESET 0
#define SOH 1
#define STX 2
#define EOT 4
#define ENQ 5
#define ACK 6
#define NAK 21
#define IDLE 22

/*********************************************/
/*       STRUCTS             */
/*********************************************/
typedef struct Configuration {
  bool
  ready : 1,
  ackPending : 1,
  isPlaying : 1,
  emergencyStop : 1,
  debug : 1,
  debugCache : 1,
  shutDown : 1,
  b8 : 1;
} Configuration;

typedef struct SpiStates {
  bool
  reading : 1,
  sending : 1,
  nextByteIsLength : 1,
  processIt : 1,
  highByte : 1,
  lowByte : 1,
  b7 : 1,
  b8 : 1;
} SpiStates;
```


7.2. Projekt WireBender2

01a_Progmem.ino

WireBender2\01_Helpers\01a_Progmem.ino

```
#define P(string_literal) (getPstring(PSTR(string_literal)))
char* getPstring(const char* addr)
{
char* buffer = new char[30];
strcpy_P(buffer, addr);
return buffer;
}
```

01b_RamInfo.ino

WireBender2\01_Helpers\01b_RamInfo.ino

```
void ramInfo()
{
ramInfo(false);
}

void ramInfo(bool extended)
{
if(config.debug != 1)
  return;

extern uint8_t __data_end;
extern uint8_t __bss_start;
extern uint8_t __bss_end;
extern uint8_t __heap_start;
extern void *__brkval;

Serial.println(F(„Memory Monitor:“));
if(extended)
{
  Serial.print(F(„I/O                = „));
  Serial.print((int)RAMSTART);

  Serial.print(F(„ bytes\n.data      = „));
  printMem((int)&__data_end - (int)RAMSTART);

  Serial.print(F(„.bss               = „));
  printMem((int)&__bss_end - (int)&__bss_start);
}
int heapPos = ((int)__brkval == 0) ? (int)&__heap_start : (int)__brkval;
Serial.print(F(„heap               = „));
printMem(heapPos - (int)&__heap_start, extended);

Serial.print(F(„stack              = „));
printMem((int)RAMEND + 1 - (int)SP, extended);

Serial.print(F(„Total              = „));
if(extended)
printMem(heapPos + ((int)RAMEND + 1 - (int)SP) - (int)RAMSTART);
else
printMem(heapPos - (int)&__heap_start + (int)RAMEND + 1 - (int)SP, false);

Serial.print(F(„Free               = „));
printMem((int)SP - heapPos, extended);

Serial.print(F(„RAM                = „));
if(extended)
printMem((int)RAMEND + 1 - (int)RAMSTART);
else
printMem((int)RAMEND + 1 - (int)&__heap_start, false);

Serial.println(F(„----------“));
}

void printMem(int val)
{
printMem(val, true);
}

void printMem(int val, bool extended)
{
extern uint8_t __heap_start;
int total = (extended) ?
((int)RAMEND + 1 - (int)RAMSTART :
((int)RAMEND + 1 - (int)&__heap_start;

Serial.print(val);
Serial.print(F(„ bytes „));
Serial.print(F(„(„));
Serial.print((float) val / ((float) total) * 100.0, 1);
Serial.println(F(„%)“));
}
```

01c_Console.ino

WireBender2\01_Helpers\01c_Console.ino

```
void SuspendDebug()
{
config.debugCache = config.debug;
config.debug = false;
}

void ResumeDebug()
{
config.debug = config.debugCache;
}

// To Replace with a DIALOG
size_t Warning(String message)
{
if(config.debug != 1)
  return 0;

Serial.print(F(„WARNING: „));
Serial.println(message);
return 0;
}

size_t Error(uint8_t errno, String text)
{
if(config.debug != 1)
  return 0;

Serial.print(F(„ERROR: „));
Serial.print(text);
Serial.print(F(„ („));
Serial.print(errno);
Serial.println(F(„)“));
return 0;
}

size_t Log(String message, const char* v1 = RESET, const char* v2 = RESET)
{
if(v1 != RESET)
  Serial.print(message);
else
  return Serial.println(message);

if(v2 != RESET)
  Serial.print(v1);
else
  return Serial.println(v1);

if(v2 != RESET)
  Serial.println(v2);

return 0;
}

size_t Debug(String message, const char* v1 = RESET, const char* v2 = RESET)
{
if(config.debug != 1)
  return 0;

return Log(message, v1, v2);
}
```

01d_String.ino

WireBender2\01_Helpers\01d_String.ino

```
String shortName(String fullname)
{
String name = fullname;
name = (name.length() > 33) ? String(F(„...“)) + name.substring(name.length() - 30) : name;
return name;
}

String lZeros(uint16_t value)  // shorter than unsigned long :)
{
String result = String(value);
if(result.length() < ZEROING)
  for(unsigned int i = result.length(); i < ZEROING; i++)
    result = String(ZERO) + result;

return result;
}

int32_t intVal(String text)
{
if(!isInt(text))
  return 0;

text.trim();
return text.toInt();
}

bool isInt(String text){ return isNumber(text, false); }

bool isFloat(String text){ return isNumber(text, true); }

bool isNumber(String text, bool floating)
{
if(text.length() < 1)
return false;

// comparer string
char comp[text.length()];
bool skip = true;
int j = 0;
for(int i = 0; i < text.length(); i++)
{
  if(text[i] != 48)
  skip = false;

  if(!skip)
  comp[j++] = text[i];
}
comp[j] = 0;
String numStr = String(comp);
numStr.trim();

//Debug(String(F(„Clean:“))+numStr);

if(floating)
{
  String comparer = numStr;

  if( numStr.indexOf(DOT) < 0 )
    comparer = comparer+String(F(„.00“));

  if( comparer.indexOf(DOT) == 0 )
    comparer = String(ZERO)+comparer;

  if( comparer.indexOf(DOT) == comparer.length() - 1 )
    comparer = comparer + String(F(„00“));

  if( comparer.indexOf(DOT) == comparer.length() - 2 )
    comparer = comparer + String(ZERO);

  float num = text.toFloat();
  //Debug(„isFloat:“+String(num)+String(SPACE)+comparer);
  if( comparer.length() > 0 && String(num) != comparer )
    return false;
}
else
```

```
{
  String comparer = (numStr.length() < 1) ? String(ZERO) : numStr;
  int32_t num = text.toInt();
  //Debug(„isInt:“+String(num)+String(SPACE)+comparer);
  if( comparer.length() > 0 && String(num) != comparer )
  return false;
}

return true;
}
```

01e_Encoder.ino

WireBender2\01_Helpers\01e_Encoder.ino

```
void initEncoder()
{
//ENCODER START
pinMode(E1, INPUT);
pinMode(E2, INPUT);
pinMode(BTN, INPUT);

attachInterrupt(0, encoderEvent, CHANGE);
attachInterrupt(1, encoderEvent, CHANGE);

digitalWrite(E1, HIGH); //turn pullup resistor on
digitalWrite(E2, HIGH); //turn pullup resistor on
digitalWrite(BTN, HIGH); //turn pullup resistor on
}

void encoderEvent() {
int MSB = digitalRead(E1); //MSB = most significant bit
int LSB = digitalRead(E2); //LSB = least significant bit

int encoded = (MSB << 1) | LSB; //converting the 2 pin value to single number
int sum  = (lastEncoded << 2) | encoded; //adding it to the previous encoded value

if ( sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011 )
  Encoder += getEncoderIncrement();
if ( sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000 )
  Encoder -= getEncoderIncrement();

lastEncoded = encoded; //store this value for next time
}

void encoderLooper()
{
//ENCODER BTN PRESSED
/*
bool btn = digitalRead(BTN);
if (!btn && encoderSettings.b2)
{
  screensaverOff();
  lastAction = millis();
  encoderSettings.b2 = false;

  if(encoderCallback == selectToolbar)
  {
    triggerToolbar();
  }
  if(encoderBtnCallback != NULL)
  {
    Debug(F(„encoderBtnCallback“));
    encoderBtnCallback();
  }
}
else if ( btn && !encoderSettings.b2)
  encoderSettings.b2 = true;
*/
//ENCODER BTN END


if (Encoder == lastEncoder && lastEncoder != 0)
  return;

//ENCODER VALUE START
if (lastEncoder / 4 != lastEncoderValue && abs((lastEncoder) - lastEncoderValue * 4) > 2)
{
  lastEncoderValue = lastEncoder / 4;

  if (encoderCallback != NULL)
  {
    Debug(F(„encoderCallback: „), String(lastEncoderValue).c_str());
    encoderCallback(lastEncoderValue);
    screensaverOff();
  }
  //else
  //Debug(„No EnoderCallBack: „+String(lastEnoderValue) + „ („ + String(lastEncoder) + „)“);
}
//ENCODER VALUE END

//ONLY at 1st RUN
if (encoderSettings.b1)
{
  if (Encoder == -1)
  {
    lastEncoder = 0;
    Encoder = 1;
  }
  else if (Encoder == 1)
  {
    lastEncoder = 0;
    Encoder = 1;
  }
  encoderSettings.b1 = false;
}

lastEncoder = Encoder;
}

void setEncoderIncrement(uint16_t increment = 0)
{
/*
  false,    //b310 / 1f
  false,    //b4100 / 10f
  false,    //b51000 / 100f
*/

encoderSettings.b3 =
encoderSettings.b4 =
encoderSettings.b5 =
  false;

if(increment > 999)
  encoderSettings.b5 = true;
else if(increment > 99)
  encoderSettings.b4 = true;
else if(increment > 9)
  encoderSettings.b3 = true;

//Debug(String(F(„Increment set to „))+String(getEncoderIncrement()));
}

uint16_t getEncoderIncrement()
{
uint16_t increment = 1;
if(encoderSettings.b3)
  increment = 10;
else if(encoderSettings.b4)
  increment = 100;
else if(encoderSettings.b5)
  increment = 1000;

if(!encoderSettings.b6)
  return increment;

return increment != 1 ? increment * 10 : increment;
}

void resetEncoder(float startValue = 0, EncoderCallback callback = NULL, Callback
btnCallback = NULL)
{
setEncoderIncrement();
Encoder = lastEncoder = startValue * 4;
encoderCallback = callback;
encoderBtnCallback = btnCallback;
}
```

01f_Checksum.ino

WireBender2\01_Helpers\01f_Checksum.ino

```
// Fletcher's 16 Checksum

uint16_t checksum( uint8_t const *data, size_t bytes )
{
uint16_t sum1 = 0xff, sum2 = 0xff;
size_t tlen;

while (bytes) {
  tlen = bytes >= 20 ? 20 : bytes;
  bytes -= tlen;
  do {
    sum2 += sum1 += *data++;
  } while (--tlen);
  sum1 = (sum1 & 0xff) + (sum1 >> 8);
  sum2 = (sum2 & 0xff) + (sum2 >> 8);
}
/* Second reduction step to reduce sums to 8 bits */
sum1 = (sum1 & 0xff) + (sum1 >> 8);
sum2 = (sum2 & 0xff) + (sum2 >> 8);
return sum2 << 8 | sum1;
}

uint16_t checksum(const char* data, size_t bytes)
{
return checksum(reinterpret_cast<const uint8_t*>(data), bytes);
}

uint16_t checksum( String text )
{
const uint8_t* data = reinterpret_cast<const uint8_t*>(text.c_str());
size_t bytes = text.length();
return checksum(data, bytes);
}
```

01fa_SD_Read.ino

WireBender2\01_Helpers\01f_SD\01fa_SD_Read.ino

```
bool openFolder(SdFile* dir, const char* path)
{
String npath = String(path);
if(npath.indexOf(„.“) >= 0)
{
  npath = npath.substring(0, npath.lastIndexOf(„/“));
  npath.toCharArray(currentPath, 255);
}

bool success = sd.exists(npath.c_str());

if(!success && npath.length() < 1)
  success = dir->openRoot(sd.vol());
else if(success)
  success = dir->open(npath.c_str(), O_READ) && dir->isDir();

return success;
}

bool openFile(SdFile* file, const char* path, bool write = false)
{
if(!sd.exists(path) && !write)
{
  ErrorDialog(String(F(„The file:#“))+shortName(path)+String(F(„#doesn't exist.“)));
  return false;
}

bool result = false;
if(String(path).length() < 1)
  result = file->openRoot(sd.vol());
else
  result = file->open(path, write ? FILE_WRITE : O_READ);

if(!result)
{
  String what = write ? F(„#has no write access.“) : F(„#couldn't be read.“);
  ErrorDialog(String(F(„The file#“))+shortName(path)+what);
  return false;
}

return true;
}
```



```
}

uint16_t countLines(SdFile* file)
{
uint16_t lCount = 0;
uint32_t lastSeek = getNextSeek(file, 0);
if(lastSeek == 0)
  return 0;

lCount++;

uint32_t nextSeek = getNextSeek(file, lastSeek);
if(nextSeek == lastSeek)
  return lCount;

while(lastSeek < nextSeek)
{
  lastSeek = nextSeek;
  nextSeek = getNextSeek(file, lastSeek);
  lCount++;
}

return lCount;
}

uint32_t getLineSeek(SdFile* file, uint16_t line)
{
if(line == 0)
  return 0;


uint16_t lCount = 0;
uint32_t lastSeek = getNextSeek(file, 0);
if(lastSeek == 0)
  return 0;
else if(line == 1)
  return lastSeek;

uint32_t nextSeek = getNextSeek(file, lastSeek);
while(lastSeek < nextSeek)
{
  lCount++;
  if(lCount == line)
    return lastSeek;

  lastSeek = nextSeek;
  nextSeek = getNextSeek(file, lastSeek);
}
return nextSeek;
}

uint32_t getNextSeek(SdFile* file, uint32_t seek)
{
uint32_t eof = file->fileSize();
if(eof == 0)
{
  ErrorDialog(getFileName(file)+F(„#is an empty file.“));
  return 0;
}

char c;
if(!file->seekSet(seek))
{
  Debug(String(F(„End of file reached.#“))+getFileName(file));
  return eof;
}

while(file->peek() >= 9)
{
  c = file->read();

  if(c == '\n')
  {
    //Serial.println(EMPTY);
    //Debug(String(F(„getNextSeek: „))+String(seek)+String(SPACE)+String(file->curPositi-
on()));
    return file->curPosition();
  }
  //else
    //Serial.print(String(c)+String(file->curPosition())+String(„ | „));
}

return eof;
}

String readLine(SdFile* file, uint32_t start, uint32_t end)
{
if(end - start > 256)
  Log(String(F(„Line seek“))+String(start)+String(F(„ in file „))+getFileName(file)+String(F(„
has more than 256 bytes.“)));

if(start >= file->fileSize())
{
  ErrorDialog(String(F(„The line couldn't be read,#because pointer is set after end of file.#“))+-
getFileName(file));
  return EMPTY;
}

if(end > file->fileSize())
  end = file->fileSize();

if(end - start < 1)
  return EMPTY;

uint16_t len = min(end - start, 256);
char content[len + 1];
file->seekSet(start);
file->read(content, len);
content[len] = 0;
String result = String(content);
result.trim();
//Debug(content+String(SPACE)+String(seek)+String(SPACE)+String(end)+String(SPA-
CE)+String(len)+String(SPACE)+String(result.length()));
return result;
}

String readLine(SdFile* file, uint32_t seek)
{
uint32_t end = getNextSeek(file, seek);
if(end <= seek)
  return EMPTY;

return readLine(file, seek, end);
}

String getFileName(SdFile* file)
{
char name[256];
if(!file->getName(name, 256))
  return EMPTY;

return name;
}

/*
String readLine(const char* path, uint32_t seek)
{
SdFile myFile;
if(!openFile(&myFile, path))
{
  ErrorDialog(String(path)+F(„#existiert nicht.“));
  return EMPTY;
}

uint8_t pos = 0;
char content[256];
memset(content, 0, sizeof(content));

char c;
myFile.seekSet(seek);
while(myFile.peek() >= 9)
{
  c = myFile.read();

  if(c == '\n' && pos > 0)
    return String(content);
  else if(c != '\n')
  {
    content[pos] = c;

    if(pos < 256)
      pos++;
    else
      pos = 0;
  }
}
myFile.close();
return EMPTY;
}
*/
```

01fb_SD_Write.ino

WireBender2\01_Helpers\01f_SD\01fb_SD_Write.ino

```
bool overrideLine(SdFile* file, String text, uint32_t seek)
{
return true;
}
```

02a_Serial.ino

WireBender2\02_Communication\02a_Serial.ino

```
bool serialLooper()
{
if(Serial.available() < 1)
  return false;

String command = Serial.readStringUntil('\n');
Serial.flush();

if( command.startsWith(F(„TEST“ )) )
  return test(command.substring(4));

if(command != F(„great“) && command != F(„wink“))
  screensaverOff();

if(command[0] == -1)
  Log(String(F(„Welcome to WireBender „))+String(VERSION));
else if(command.startsWith(F(„upload „)))
  Upload(command.substring(7));
else if(command.startsWith(F(„delete „)))
  Delete(command.substring(7));
else if(command.startsWith(F(„out „)))
  Post(command.substring(4));
else if(command == F(„browser“))
  initBrowser();
else if(command == F(„viewer“))
  initViewer();
else if(command == F(„jogging“))
  initJogging();
else if(command == F(„great“))
  Easteregg(F(„great“));
else if(command == F(„wink“))
  Easteregg(F(„wink“));
else if(command == F(„calibrate“))
  Calibrate();
else if(command == F(„shutdown“))
  Shutdown();
else if(command.startsWith(F(„man „)))
  ManualCommand(command.substring(4));
else
  Log(F(„Unknown command“));

return true;
}


// Test Code
bool test(String params)
{
params.trim();

if( params.length() < 1 )
  SaveSetting(F(„X Pos“), 999);
else
  SaveSetting(params, 999);
}
```

02b_FileSystem.ino

WireBender2\02_Communication\02b_FileSystem.ino

```
664  void cancelClick(ButtonInfo* i)
665  {
666  encoderSettings.b7 = true;
667  }
668
669  String getResponse()
670  {
671  uint32_t start = millis();
672  while(millis() - start < 5000 && !UsbCancelLooper())
673  {
674    delay(3);
675    if(Serial.available() < 1)
676      continue;
677
678    String s = Serial.readStringUntil('\n');
679    s.trim();
680    return s;
681  }
682  return EMPTY;
683  }
684
685  bool UsbCancelLooper()
686  {
687  uint32_t start = millis();
688  while(Serial.available() < 1 && millis() - start < 5)
689  {
690    touchLooper();
691    if(encoderSettings.b7)
692      return true;
693  }
694  return false;
695  }
696
697  bool UsbError(String msg)
698  {
699  ErrorDialog(String(F("SERIAL_OPERATION_FAILED.#-#"))+msg);
700  Serial.println(msg);
701    ResumeDebug();
702  return false;
703  }
704
705  /*****************************************************************************/
706  /*                                                                  UPLOAD   */
707  /*****************************************************************************/
708  bool Upload(String path)
709  {
710  encoderSettings.b7 = false;
711
712    SuspendDebug();
713  UsbDialog(F("Upload"), F("Data transfer via USB started."));
714
715    Serial.println("Upload started");  // ---->
716
717    // Get Path
718    if(path.length() < 1)
719      return UsbError(F("Timeout on upload (retrieve path)."));
720    else if(path == F("CANCEL"))
721      return UsbError(F("The upload has been cancelded by the user."));
722
723    if(!path.startsWith(SLASH) || path.endsWith(SLASH))
724      return UsbError(String(F("The file path is invalid:#"))+path);
725
726    if(!path.endsWith(F(".w")))
727      path += F(".w");
728
729    if(sd.exists(path.c_str()))
730      sd.remove(path.c_str());
731
732    String folder = path.substring(0, path.lastIndexOf(SLASH));
733    if(!sd.exists(folder.c_str()))
734      sd.mkdir(folder.c_str());
735
736    SdFile file;
737    if(!openFile(&file, path.c_str(), true))
738      return false;//UsbError(String(path)+F("#kann nicht geschrieben werden."));
739
740    Serial.println(String(checksum(String(F("upload ")+path))); // ---->
741
742    //<---- Get LineCount
743    String lineCount = getResponse();
744    if(lineCount.length() < 1)
745      return UsbError(F("Timeout on Upload (retrieve linecount)."));
746    else if(lineCount == F("CANCEL"))
747      return UsbError(F("The upload has been canceled by the user."));
748
749    uint16_t ITotal = lineCount.toInt();
750    if(ITotal < 1)
751    {
752      file.remove();
753      return UsbError(F("Error on data transfer.#The line count is invalid."));
754    }
755
756    Serial.println("Linecount confirmed");  // ---->
757    Serial.println(String(checksum(lineCount))); // ---->
758
759    //Get DATA
760    //dialogTextS(um(path+"\nmit "+String(ITotal)+" Zeilen wird übertragen."));
761
762    uint16_t ICount = 0;
763    uint8_t lastPercentage = 0;
764    UploadProgress(0,0);
765    uint32_t start = millis();
766    bool bar = false;
767
768    while(true)
769    {
770      if(millis() - start >= TIMEOUT || ICount >= ITotal || UsbCancelLooper())
771        break;
772
773        //<---- Get Line Data
774        String line = getResponse();
775        if(line.length() < 1)
776          return UsbError(F("Timeout on Upload (line data)."));
777        else if(line == F("CANCEL"))
778          return UsbError(F("The upload has been canceled by the user."));
779
780        line.trim();
781        if(ICount < ITotal - 1)
782          file.println(line);
783        else
784          file.print(line);
785
786        ICount++;
787
788        Serial.println(EMPTY);
789        Serial.println(String(checksum(line))); // ---->
790        uint8_t percentage = (uint8_t) floor((float)ICount / (float)ITotal * 100.0f);
791
792        if(millis() - start > 500)
793        {
794          if(percentage != lastPercentage)
795          {
796            drawUsbText(String(ICount)+String(F(" of "))+String(ITotal)+String(F(" lines uploaded.
(,,)")+String(percentage)+String(F(",,%")));
797              UploadProgress(percentage, lastPercentage);
798              lastPercentage = percentage;
799            }
800          start = millis();
801        }
802    }
803
804    if(!encoderSettings.b7 && ICount == ITotal)
805      UploadProgress(100, lastPercentage);
806
807    file.close();
808    delay(3);
809
810    if(encoderSettings.b7)
811      return UsbError(F("The upload has been canceled by the user."));
812    else if(ICount < ITotal)
813      return UsbError(F("Upload failed because linecount does not match#the received data."));
814
815    Serial.println(String(F("Upload successful "))+String(ICount)+String(F(" of "))+String(ITo-
tal)); // ---->
816    Serial.println(F("success")); // ---->
817    ResumeDebug();
818
819
820    setViewerPath(path);
821    drawUsbText(String(ITotal)+String(F(" lines successfully uploaded.")));
822    dialogBtn3ViewFile();
823    return true;
824  }
825
826  void UploadProgress(uint8_t percent, uint8_t lastpercent)
827  {
828  if(percent == 100 && lastpercent == 0)
829    t.fillRect(85, 165, 225, 20, BLUE);
830  else if(percent == 0)
831    t.fillRect(85, 165, 225, 20, GRAY);
832
833  if(percent <= lastpercent)
834    return;
835
836  percent = constrain(percent, 0, 100);
837  lastpercent = constrain(lastpercent, 0, 100);
838
839  uint16_t pos = map(lastpercent, 0, 100, 0, 225) + 85;
840  uint8_t length = map(percent - lastpercent, 0, 100, 0, 225);
841
842  t.fillRect(pos, 165, length, 20, BLUE);
843  }
844
845
846  /*****************************************************************************/
847  /*                                                                  DELETE   */
848                  */
849  /*****************************************************************************/
850  bool Delete(String path)
851  {
852  SuspendDebug();
853  UsbDialog(F("Delete"), F("File will be deleted."));
854
855  //Get Path
856  if(path.length() < 1)
857    return UsbError(F("No path for deleting received."));
858
859    if(!path.startsWith(SLASH) || (!path.endsWith(F(".w")) && !path.endsWith(F(".s")) && !path.
endsWith(SLASH)))
860      return UsbError(String(F("The path is invalid:#"))+path);
861
862    String typ = path.endsWith(SLASH) ? F("The folder:#") : F("The file:#");
863
864    if(!sd.exists(path.c_str()))
865      return UsbError(typ+path+F("#doesn't exist."));
866
867    if(path.endsWith(SLASH))
868    {
869      if(!DeleteDir(path))
870        return false;
871    }
872    else if(!sd.remove(path.c_str()))
873      return UsbError(typ+path+F("#couldn't be deleted."));
874
875    Serial.println("success");
876    ResumeDebug();
877
878
879    drawUsbText(typ+path+String(F("#has been deleted successfully.")));
880    dialogOK();
881    return true;
882  }
883
884  bool DeleteDirectory(SdFile* dir)
885  {
886  if(dir->isRoot())
887    return UsbError(F("The root folder can't be deleted."));
888
889    SdFile child;
890    //Delete all Files
891    while (child.openNext(dir, O_WRITE))
892    {
893      char name[40];
894      child.getName(name, 40);
895      if(child.isFile())
896      {
897        if(!child.remove())
898          return UsbError(String(F("The File:#"))+String(name)+String(F("#couldn't be dele-
ted.")));
899      }
900      child.close();
901    }
902
903    sd.vwd()->seekSet(0);
904    while (child.openNext(dir))
905    {
906      char name[40];
907      child.getName(name, 40);
908      if(child.isDir())
909      {
910        if(!DeleteDirectory(&child))
911          return false;
912
913        if(!child.rmdir())
914          return UsbError(String(F("The folder:#"))+String(name)+String(F("#couldn't be dele-
ted.")));
915      }
916      child.close();
917    }
918
919    return true;
920  }
921
922  bool DeleteDir(String path)
923  {
924  String typ = F("The folder:#");
925
926    SdFile dir;
927    if(!dir.open(path.c_str()) || !dir.isDir())
928      return UsbError(typ+path+F("#couldn't be accessed."));
929
930    bool result = DeleteDirectory(&dir);
931    if(!dir.rmdir())
932      return UsbError(typ+path+F("#couldn't be deleted."));
933
934    dir.close();
935    return result;
936  }
```

02ba_Helpers.ino

WireBender2\02_Communication\02b_SPI\02ba_Helpers.ino

```
937  void begin()
938  {
939  SPI.beginTransaction(_spiSettings);
940  digitalWrite(CS, LOW);
941    // SS is pin 53
942  }
943
944  void end()
945  {
946  digitalWrite(CS, HIGH);
947  SPI.endTransaction();
948  }
949
950  bool DriverReady()
951  {
952  SPI.transfer(ENQ);
953  delayMicroseconds(WAIT);
954  uint8_t ack = SPI.transfer(ENQ);
955  uint16_t attempts = 0;
956  while(ack != ACK && attempts < TIMEOUT)
957  {
958    ack = SPI.transfer(ENQ);
959    if(ack == ACK)
960      break;
961    else
962    {
963      //Debug("ENQ="+String(ack)+" "+String(attempts));
964      if(attempts > 10)
965        delayMicroseconds(WAIT * 100);
966      else
967        delayMicroseconds(WAIT);
968      attempts++;
969    }
970  }
971  //if(attempts > 0)
972    //Debug("Attempts="+String(attempts)+", Ack="+String(ack));
973  return ack == ACK;
974  }
975
976  bool DriverProcessFinished()
977  {
978  SPI.transfer(IDLE);
979  delayMicroseconds(WAIT);
980  uint8_t ack = SPI.transfer(IDLE);
981  uint16_t attempts = 0;
982  while(ack == IDLE && attempts < TIMEOUT)
983  {
984    ack = SPI.transfer(IDLE);
985
986    if(ack != IDLE)
987      break;
988    else
989    {
990      delayMicroseconds(WAIT);
991      attempts++;
992    }
993  }
994  delayMicroseconds(WAIT);
995  //Debug("Attempts="+String(attempts)+", Ack="+String(ack));
996  return ack != IDLE;
997  }
998
999  bool RaiseError(uint8_t err)
1000 {
1001 end();
1002   resetSpi();
1003   return ErrorLog(err);
1004 }
```

02bb_Send.ino

WireBender2\02_Communication\02b_SPI\02bb_Send.ino

```
1005 bool Post(String text)
1006 {
1007   uint8_t postAttempts = 0;
1008   uint8_t success = 2;
1009
1010   lastError = 0;
1011   while(postAttempts < MAX_SENDS && success == 2)
1012   {
1013     success = Send(text);
1014     postAttempts++;
1015   }
1016   if(success != 1 && lastError > 0)
1017     Debug(String(F("_Post Failure(,,)")+String(lastError)+String(F(",,:"))+String(success ==
1)+String(F(" E:"))+String(eText(lastError)));
1018
1019   return success == 1;
1020 }
1021 uint8_t Send(String text)
1022 {
1023   text.toLowerCase();
1024   //Debug("--------------------");
1025   word sum = checksum(text);
1026   Debug(String(F("_OUT(,,)")+String(text.length())+F(",,:")+text+String(F(" H"))+highBy-
te(sum)+String(F(" L"))+lowByte(sum));
1027
1028   begin();
1029
1030   // Waiting for Driver to be ready
1031   if(!DriverReady())
1032     return RaiseError(1);
1033
1034   // Init Sending
1035   SPI.transfer(SOH);
1036       // SOH (start of header)
1037   SPI.transfer((uint8_t)text.length());
1038   // Length of data
1039
1040   // Sending Data
1041   int i = 0;
1042
1043   while(i < (uint8_t) text.length())
1044   // Send the actual data
1045   {
1046     SPI.transfer(text[i++]);
1047     delayMicroseconds(1);
1048   }
1049
1050   // End Sending
1051   delayMicroseconds(3);
1052   SPI.transfer(EOT);
1053       // EOT (end of transmission)
1054   delayMicroseconds(3);
1055   SPI.transfer(highByte(sum));
1056   // Send HighByte of checksum
1057   delayMicroseconds(3);
1058   SPI.transfer(lowByte(sum));
1059     // Send LowByte of checksum
1060   delayMicroseconds(WAIT);
1061
1062   /* REPLACED WITH CHECKSUMs
1063   Length Integrity Check
1064   if(lenAck != text.length())
1065   {
1066     Debug("_Ack=-"+String(lenAck));
1067     RaiseError(2);
1068     return 2; //repeat send
1069   }
1070   */
1071
1072   // Waiting for driver processing finihed
1073   if(DriverProcessFinished())
1074   {
1075     uint8_t errno = SPI.transfer(IDLE);
1076     bool err = RaiseError(errno);
1077
1078     // Checksum Error
1079     if(errno == 6)
1080       return 2; //repeat send
1081   else
1082     return err;
1083   }
1084
1085   return RaiseError(3);
1086 }
```

02bc_Receive.ino

WireBender2\02_Communication\02b_SPI\02bc_Receive.ino

```
1087 void Receive()
1088 {
1089   config.requestIt = true;
1090 }
1091
1092 void ReceiveProcessError(uint8_t errno = 0)
1093 {
1094   resetSpi();
1095   SPI.transfer(NAK);
1096   delayMicroseconds(WAITLONG);
1097   SPI.transfer(RESET);
1098   end();
1099
1100   if(errno > 0)
1101     RaiseError(errno);
1102 }
1103
1104 void Request()
1105 {
1106   if(!config.requestIt)
1107     return;
1108
1109   begin();
1110
1111   // Waiting for Driver to be ready
1112   if(!DriverReady())
1113   {
1114     RaiseError(1);
1115     return;
1116   }
1117
1118   // Init Receiving
1119   SPI.transfer(STX);
1120       // STX (start of text)
1121   delayMicroseconds(WAITLONG);
1122   dataLength = SPI.transfer(ENQ);                                    // Length of data
1123   delayMicroseconds(WAITLONG);
1124
1125 // dataLength must fit Buffer
1126 if(dataLength < 1)
1127     return ReceiveProcessError();
1128
```

```
    if(dataLength >= SPI_BUFFER)
      dataLength = SPI_BUFFER - 1;

    // Receiving byte after byte
    pos = 0;
    delayMicroseconds(WAITLONG);
    while(pos <= dataLength)
    {
      buf[pos++] = SPI.transfer(ENQ);                    // Receive the
    actual data
      delayMicroseconds(WAITLONG);
    }

    uint8_t low_byte = SPI.transfer(ENQ);                // Receive HighBy-
    te of checksum
    delayMicroseconds(WAITLONG);
    uint8_t high_byte = SPI.transfer(ENQ);               // Receive LowByte
    of checksum
    delayMicroseconds(WAITLONG);

    // Compare Checksums
    buf[dataLength] =  0;
  String command = String(buf);
  uint16_t sum = checksum(command);

    if(high_byte != highByte(sum) || low_byte != lowByte(sum))
    {
    ReceiveProcessError(6);

      // TESTING
      Debug(String(F("H"))+high_byte+String(F("L"))+low_byte);
      Debug(String(F("IN("))+String(command.length())+String(F("):"))+command+String(F("
    H"))+highByte(sum)+String(F("L"))+lowByte(sum));
      return;
    }

    SPI.transfer(ACK);
    delayMicroseconds(WAITLONG);
    SPI.transfer(RESET);

    end();
    config.processIt = true;
    config.requestIt = false;
  }
```

02bd_Init.ino

WireBender2\02_Communication\02b_SPI\02bd_Init.ino

```
  void initSPI()
  {
    pinMode(CS, OUTPUT);
    digitalWrite(CS, HIGH); // ensure SS stays high for now
    attachInterrupt(digitalPinToInterrupt(MS), Receive, HIGH);

  // Put SCK, MOSI, SS pins into output mode
    // also put SCK, MOSI into LOW state, and SS into HIGH state.
  // Then put SPI hardware into Master mode and turn SPI on
    SPI.begin();

    // Slow down the master a bit
    SPI.setClockDivider(SPI_CLOCK_DIV2);
  Log(String(F("SPI Master SS=")+String(SS));
  }

  void resetSpi()
  {
    pos = dataLength = 0;
    config.processIt = config.requestIt = false;
  }
```

02be_Errors.ino

WireBender2\02_Communication\02b_SPI\02be_Errors.ino

```
  bool ErrorLog(uint8_t errno)
  {
    lastError = errno;
    if(lastError < 1)
      return true;

    switch(errno)
    {
      // GENERAL
      case 10: //Shutdown
      case 11: //Parking Position
        return true;

      // SPI Errors
      case 20:
      case 21:
        return false;

      // Driver Interpreter
      case 60:    //SKIP
        AckLine(String(F("ACK "))+String(selectedLine + 1));
        return true;
      case 61:    //OK

      // Driver Setting Interpreter
      case 70:    //SKIP
      case 71:    //OK

    // Driver Calibration
      case 80:    //X-Axis
      case 81:    //Y-Axis
        return true;

      default:
        return Error(errno, eText(errno));
    }

    return false;
  }

  const String eText(uint8_t errno)
  {
    switch(errno)
    {
      // NO ERROR
      case 20:    //SKIP
      case 21:    //OK
      case 60:    //SKIP
      case 61:    //OK
      case 70:    //SKIP
      case 71:    //OK
        return EMPTY;

      // Controller Internal Errors
      case 1:
        return F("DRIVER_NOT_RESPONDING");
      case 2:
        return F("INTEGRITY_LOST_SENDING");
      case 3:
        return F("PROCESSING_TIMEOUT");
      case 4:
        return F("SPI_TIMEOUT");
      case 5:
        return F("DATA_TOO_BIG");
      case 6:
        return F("CHECKSUM_ERROR");
      case 7:
        return F("DRIVER_RESTART");

      case 10:
        return F("DRIVER_START_FAILED");

      // Playing Errors
      case 53:
        return F("EMERGENCY_STOP");
      case 54:
        return F("USER_DEFINED_STOP");
      case 55:
        return F("DRIVER_IS_BUSY");

      // Driver Interpreter Errors

      case 62:
        return F("INTERPRETER_INVALID_CHARS");
      case 63:
        return F("INTERPRETER_LINE_NUMBER_NOT_FOUND");
      case 64:
        return F("INTERPRETER_LINE_NUMBER_INVALID");
      case 65:
        return F("INTERPRETER_VALUE_PARSE_ERROR");
      case 66:
        return F("INTERPRETER_AXIS_COMMAND_ERROR");
      case 67:
        return F("ACK_LINE_NUMBER_MISSING");
      case 68:
        return F("ACK_LINE_NUMBER_INVALID");

      // Driver Setting Interpreter Errors

      case 72:
        return F("SET_INVALID_SYNTAX");
      case 73:
        return F("SET_INVALID_VALUE");
      case 74:
        return F("SET_INVALID_NAME");
      case 75:
        return F("SET_FAILED_ON_EMPTY_AXIS");

      // Driver Calibration

      case 82:
        return F("CALIBRATION_NOT_POSSIBLE");
      case 83:
        return F("CALIBRATION_FAILED");

      // GENERAL


      // UNKNOWN Errors
      default:
        return F("UNKNOWN");
    }
  }
```

02bf_Looper.ino

WireBender2\02_Communication\02b_SPI\02bf_Looper.ino

```
  void spiLooper()
  {
    if(config.requestIt)
      Request();

  if(!config.processIt)
      return;

    if(dataLength < 1)
      return resetSpi();

    screensaverOff();

    buf[min(dataLength, SPI_BUFFER - 1)] =  0;
  String command = buf;

    // Parsers
    if(receiveCallback != NULL)
    {
      receiveCallback(command);
      return resetSpi();
    }

    if(command.startsWith(F("ACK")))
      AckLine(command);
    else if(command.startsWith(F("RESTART")))
      bootUp();
    else if(command == F("RESET"))
    {
      if(config.programRunning)
        Stop();
    }
    else if(command == F("SHUTDOWN"))
      Shutdown();
    else
      Debug(String(F("IN("))+String(command.length())+String(F("):"))+command);

    resetSpi();
  }
```


02a_Helpers.ino

WireBender2\03_TFT\02a_Helpers.ino

```
void clearRect(uint16_t x, uint8_t y, uint16_t width = 0, uint8_t height = 0)
{
  uint16_t w = (width == 0) ? t.width() - x : width;
  uint8_t h = (height == 0) ? t.height() - y : height;
  t.setTextArea(x, y, w, h);
  t.setTextColor(BLACK, WHITE);
  t.clearTextArea();
}

void clearLineButtonRect()
{
  clearRect(46, 24);
}

void displayText(uint16_t x, uint8_t y, String text, uint16_t width = 0, uint8_t height = 0)
{
  uint16_t w = (width == 0) ? t.width() - x : width;
  uint8_t h = (height == 0) ? t.height() - y : height;
  t.setFont(Calibri13);
  t.setTextArea(x, y, w, h);
  t.setTextColor(GRAY, WHITE);
  t.printAligned(text, gTextAlignMiddleCenter);
}

void terminatePreviousView(bool clear = false)
{
  terminateToolbar();
  terminateLineButtons(clear);
  resetEncoder();

  if(terminateCallback != NULL)
    terminateCallback();

  terminateCallback = NULL;
  errorCallback = NULL;
}

void terminateButtonCaller(ButtonInfo* i)
{
  terminatePreviousView();
}
```

02a_TFT.ino

WireBender2\03_TFT\02a_TFT.ino

```
void initTFT()
{
  if(encoderSettings.b8)
    return;

  t.begin();
  t.setPowerLevel(pwrLevelNormal);
  t.display(true);
  t.invertDisplay(false);
  t.setRotation(iliRotation270);
  t.setFont(Calibri13);
  t.setTextColor(BLACK, WHITE);
  digitalWrite(LITE, HIGH);
  encoderSettings.b8 = true;
}

void bmpDraw(String file, uint16_t x, uint16_t y)
{
  if(( x >= (uint16_t) t.width()) || (y >= (uint16_t) t.height()))
    return;

  const char* filename = file.c_str();

  if(!sd.exists(filename))
  {
    Debug(F("The file "), filename, String(F(" doesn't exist")).c_str());
  return;
  }

  SdFile  bmpFile;
  int     bmpWidth, bmpHeight;  // W+H in pixels
  uint8_t  bmpDepth;            // Bit depth (currently must be 24)
  uint8_t  headerSize;
  uint32_t bmpImageoffset;      // Start of image data in file
  uint32_t fileSize;
  boolean goodBmp = false;      // Set to true on valid header parse
  boolean  flip = true;      // BMP is stored bottom-to-top
  uint16_t w, h;

  if (!bmpFile.open(filename, O_READ))
  {
    Debug(F("Failed to open file: "), filename);
    return;
  }

  // Parse BMP header
  if (read16(bmpFile) == 0x4D42) // BMP signature
  {
    fileSize = read32(bmpFile);
    (void)read32(bmpFile); // Read & ignore creator bytes
    bmpImageoffset = read32(bmpFile); // Start of image data
    headerSize = read32(bmpFile);
    bmpWidth = read32(bmpFile);
    bmpHeight = read32(bmpFile);
    if (read16(bmpFile) == 1) // # planes -- must be ,1'
    {
      bmpDepth = read16(bmpFile); // bits per pixel
      if (read32(bmpFile) == 0) // 0 = uncompressed
      {
        if (bmpHeight < 0) {
          bmpHeight = -bmpHeight;
          flip = false;
        }

        // Crop area to be loaded
        w = bmpWidth;
        h = bmpHeight;
        if (((int16_t) x + (int16_t)w - 1) >= t.width())
          w = t.width() - x;
        if (((int16_t) y + (int16_t)h - 1) >= t.height())
          h = t.height() - y;

        // Set TFT address window to clipped image bounds
        t.setAddrWindowRect(x, y, w, h);

        if (bmpDepth == 16) //565 format
        {
          goodBmp = true; // Supported BMP format -- proceed!

          uint16_t buffer[BUFFPIXELCOUNT]; // pixel buffer

          bmpFile.seekSet(54);  //skip header
          uint32_t totalPixels = (uint32_t)bmpWidth * (uint32_t)bmpHeight;
          uint16_t numFullBufferRuns = totalPixels / BUFFPIXELCOUNT;
          for (uint32_t p = 0; p < numFullBufferRuns; p++) {
            // read pixels into the buffer
            bmpFile.read(buffer, 2 * BUFFPIXELCOUNT);
            // push them to the diplay
            t.pushColors(buffer, 0, BUFFPIXELCOUNT);

          }

        // render any remaining pixels that did not fully fit the buffer
          uint32_t remainingPixels = totalPixels % BUFFPIXELCOUNT;
          if (remainingPixels > 0)
          {
            bmpFile.read(buffer, 2 * remainingPixels);
            t.pushColors(buffer, 0, remainingPixels);
          }
        }
      }
    }
  }

  bmpFile.close();
}

// These read 16- and 32-bit types from the SD card file.
// BMP data is stored little-endian, Arduino is little-endian too.
// May need to reverse subscript order if porting elsewhere.

uint16_t read16(SdFile& f)
{
  uint16_t result;
  ((uint8_t *)&result)[0] = f.read(); // LSB
  ((uint8_t *)&result)[1] = f.read(); // MSB
  return result;
}

uint32_t read32(SdFile& f)
{
  uint32_t result;
  ((uint8_t *)&result)[0] = f.read(); // LSB
  ((uint8_t *)&result)[1] = f.read();
  ((uint8_t *)&result)[2] = f.read();
  ((uint8_t *)&result)[3] = f.read(); // MSB
  return result;
}
```

02b_Screensaver.ino

WireBender2\03_TFT\02b_Screensaver.ino

```
void screensaver() {
  if(!config.screenActive)
    return;

  unsigned long screensaverTime = (long)SCREENSAVER * long(1000);
  unsigned long elapsed = millis() - lastAction;

  if(elapsed < screensaverTime)
    return;

  for (int i = 255; i >= 0; i--)
  {
    analogWrite(LITE, i);
    delay(6);
  }
  digitalWrite(LITE, LOW);
  t.display(false);
  config.screenActive = false;
}

bool screensaverOff() {
  lastAction = millis();
  if (config.screenActive)
    return false;

  t.display(true);
  for (int i = 0; i <= 255; i++)
  {
    analogWrite(LITE, i);
    delay(6);
  }
  digitalWrite(LITE, HIGH);
  config.screenActive= true;

  return true;
}
```

02c_Button_Class.ino

WireBender2\03_TFT\02c_Button_Class.ino

```
class Button
{
  private:
    ButtonCallback
      _click = NULL,
      _draw = NULL;

  public:
    ButtonInfo
      _info;

    Button() { initParams(); }
```

```
1561
1562      //GENERIC
1563      Button( uint16_t x, uint8_t y, uint16_t w, uint8_t h, const __FlashStringHelper* text, bool
          enabled = true, bool visible = true, ButtonCallback clickButtonCallback = NULL, ButtonCall-
          back drawButtonCallback = NULL )
1564      {
1565        initParams();
1566        _info.enabled = enabled;
1567        _info.visible = visible;
1568
1569        _info.x = x;
1570        _info.y = y;
1571        _info.width = w;
1572        _info.height = h;
1573        _info.name = text;
1574
1575        _click = clickButtonCallback;
1576        _draw = drawButtonCallback;
1577
1578        this->draw(false);
1579      }
1580
1581      //TOOLBAR
1582      Button(uint8_t pos, const __FlashStringHelper* text, bool enabled = true, ButtonCallback
          clickButtonCallback = 0, ButtonCallback drawButtonCallback = 0)
1583      {
1584        initParams();
1585        _info.enabled = enabled;
1586        _info.visible = true;
1587
1588        _info.y = pos * 48;
1589        _info.width = 45;
1590        _info.height = 47;
1591        _info.name = text;
1592
1593        _click = clickButtonCallback;
1594        _draw = drawButtonCallback;
1595
1596        if (_info.y > 190)
1597          _info.height = 48;
1598
1599        this->draw(false);
1600      }
1601
1602      void initParams()
1603      {
1604        _info.active = false;
1605        _info.enabled = false;
1606        _info.visible = false;
1607        _info.selected = false;
1608      }
1609
1610      void enable(bool value)
1611    {
1612      if(this->_info.enabled == value)
1613          return;
1614
1615        this->_info.enabled = value;
1616        this->draw(false);
1617      }
1618
1619      bool clicked()
1620      {
1621        if(
1622          _info.enabled &&
1623          //X Koorinate liegt innerhalb des Button Bereichs
1624          _tPoint.x >= (uint16_t)_info.x &&
1625          _tPoint.x <= (uint16_t)_info.x + _info.width &&
1626          //Y Koorinate liegt innerhalb des Button Bereichs
1627          _tPoint.y >= _info.y &&
1628          _tPoint.y <= _info.y + _info.height
1629        )
1630        {
1631          uint32_t start = millis();
1632
1633          this->trigger();
1634
1635          if(millis() - start < 300)
1636            delay(300 - (millis() - start));
1637
1638          return true;
1639        }
1640
1641        return false;
1642      }
1643
1644      void trigger()
1645      {
1646        this->draw(true);
1647
1648        Debug(F("Button triggered: "), String(_info.name).c_str());
1649
1650        if(_click != NULL)
1651          _click(&this->_info);
1652
1653        this->_info.selected = false;
1654        if(encoderCallback == selectToolbar)
1655          resetEncoder();
1656      }
1657
1658      void select(bool selected)
1659      {
1660        if(_info.selected == selected)
1661          return;
1662
1663        _info.selected = selected;
1664        this->draw(false);
1665      }
1666
1667      void draw(bool active)
1668      {
1669        _info.active = active;
1670        if(!_info.visible)
1671          return;
1672
1673        uint16_t c = (_info.active) ? MAGENTA : BLUE;
1674      if(!_info.enabled)
1675          c = GRAY;
1676
1677        if(_draw != NULL)
1678        {
1679          t.fillRect(_info.x, _info.y, _info.width, _info.height, c);
1680          _draw(&this->_info);
1681        }
1682        else
1683        {
1684          t.setTextArea(_info.x, _info.y, _info.width, _info.height);
1685          t.setTextColor(WHITE, c);
1686          t.clearTextArea();
1687          t.printAligned(_info.name, gTextAlignMiddleCenter);
1688        }
1689
1690        if(_info.selected)
1691        {
1692          t.drawRect(_info.x, _info.y, _info.width, _info.height, MAGENTA);
1693          t.drawRect(_info.x + 1, _info.y + 1, _info.width - 2, _info.height - 2, MAGENTA);
1694        }
1695      }
1696  };
```

02d_Line_Class.ino

WireBender2\03_TFT\02d_Line_Class.ino

```
1697  class Line
1698  {
1699    private:
1700    LineCallback
1701      _click = NULL,
1702      _draw = NULL;
1703
1704    public:
1705    LineInfo
1706      _info;
1707
1708    Line() { initParams(); }
1709
1710    //BROWSER Line
1711    Line(uint8_t lineNumber, LineCallback clickButtonCallback, LineCallback drawButton-
          Callback, uint8_t type = 0, uint32_t seek = 0)
1712    {
1713      initParams();
1714        _info.isBrowserLine = type == 0;
1715        _info.isViewerLine = type == 1;
1716        _info.lineNumber = lineNumber;
1717        _info.seek = seek;
1718
1719        _click = clickButtonCallback;
1720        _draw = drawButtonCallback;
1721
1722        draw(false);
1723    }
1724
1725    void initParams()
1726    {
1727      _info.active = false;
1728      _info.sending = false;
1729      _info.isBrowserLine = false;
1730      _info.isViewerLine = false;
1731      _info.interpreted = false;
1732      _info.isComment = false;
1733      _info.isValid = false;
1734      _info.isCommand = false;
1735    }
1736
1737    uint8_t getPos()
1738    {
1739      if(_info.isBrowserLine)
1740        return _info.lineNumber % BROWSELINES;
1741      else if(_info.isViewerLine)
1742        return _info.lineNumber % VIEWLINES;
1743    else
1744        return _info.lineNumber % SETTINGSLINES;
1745    }
1746
1747    int16_t getHeight()
1748    {
1749      return this->_info.isBrowserLine ? 24 : 13;
1750    }
1751
1752    int8_t getTopOffset()
1753    {
1754      return (!this->_info.isBrowserLine && !this->_info.isViewerLine) ? 50 : 28;
1755    }
1756
1757    bool clicked()
1758    {
1759    if(
1760        //X Koorinate liegt innerhalb des Button Bereichs
1761        _tPoint.x >= (uint16_t)46 &&
1762        _tPoint.x <= (uint16_t)46 + 274 &&
1763        //Y Koorinate liegt innerhalb des Button Bereichs
1764        _tPoint.y >= getPos() * this->getHeight() + getTopOffset() &&
1765        _tPoint.y <= getPos() * this->getHeight() + this->getHeight() + getTopOffset()
1766      )
1767      {
1768        uint32_t start = millis();
1769
1770        this->trigger();
1771
1772        if(millis() - start < 300)
1773          delay(300 - (millis() - start));
1774
1775        return true;
1776      }
1777
1778      return false;
1779    }
1780
1781    void trigger()
1782    {
1783      this->draw(true);
1784
1785      Debug(F("Line triggered #"), String(this->_info.lineNumber).c_str());
1786
1787      if(_click != NULL)
1788        _click(&this->_info);
1789    }
1790
1791    void draw(bool active)
1792    {
1793      _info.active = active;
1794
1795      if(_draw != NULL)
1796        _draw(&this->_info);
1797    }
1798  };
```


02e_Axis_Class.ino

WireBender2\03_TFT\02e_Axis_Class.ino

```
1799  class Axis
1800  {
1801    private:
1802      uint8_t yZero = 121;
1803
1804      bool
1805        enabled : 1,
1806        selected : 1,
1807        initialized : 1,
1808        isYaxis : 1,
1809        isFeed : 1,
1810        b6 : 1,
1811        b7 : 1,
1812      b8 : 1;
1813      int32_t
1814        pos = 0,
1815        seek = 0;
1816
1817      int16_t x()
1818      {
1819        if(isYaxis)
1820          return 108;
1821        else
1822          return 54;
1823      }
1824
1825      int16_t y(int32_t pos)
1826      {
1827        if(isFeed)
1828          return map(percentage(pos), 0, 100, -90, 0);
1829
1830        return 25 + map(percentage(pos), -100, 100, 0, 192);
1831      }
1832
1833      String name()
1834      {
1835        return isFeed ? F("Feed") : isYaxis ? F("X-Axis") : F("Y-Axis");
1836      }
1837
1838      int32_t xAxisPercentage(int32_t pos)
1839      {
1840        return map(pos, -XMAX, XMAX, -100, 100);
1841      }
1842
1843      int32_t yAxisPercentage(int32_t pos)
1844      {
1845        return map(pos, -YMAX, YMAX, -100, 100);
1846      }
1847
1848      int32_t zAxisPercentage(int32_t pos)
1849      {
1850        return map(pos, 0, ZMAX, 0, 100);
1851      }
1852
1853      int8_t percentage(int32_t value)
1854      {
1855        if(isFeed)
1856          return constrain(zAxisPercentage(value), 0, 100);
1857
1858        return constrain(isYaxis ? yAxisPercentage(value): xAxisPercentage(value), -100, 100);
1859      }
1860
1861      void drawAxis()
1862      {
1863        if(isFeed)
1864        {
1865          t.fillArc(320, 218, 154, 45, -90, 0, GRAY);
1866          t.setFont(Calibri13);
1867          t.setTextArea(250, 202, 70, 16);
1868          t.setTextColor(WHITE, BLUE);
1869          t.clearTextArea();
1870          t.printAlignedOffseted(F("Feed"), gTextAlignMiddleRight, -10, 0);
1871        }
1872        else
1873        {
1874          t.setFont(Calibri13);
1875          t.setTextArea(x() + 13, 0, 19, 18);
1876          t.setTextColor(WHITE, BLUE);
1877          t.clearTextArea();
1878          t.printAligned(isYaxis ? F("Y") : F("X"), gTextAlignMiddleCenter);
1879          drawAxisBorder();
1880
1881        }
1882      }
1883
1884      void drawAxisBorder()
1885      {
1886        uint16_t c = (selected) ? MAGENTA : GRAY;
1887
1888        // Feed
1889        if(isFeed)
1890        {
1891          if(!selected)
1892            c = WHITE;
1893
1894          t.fillArc(320, 218, 156, 2, -90, 0, c);
1895          t.fillArc(320, 218, 109, 2, -90, 0, c);
1896
1897          t.drawFastHLine(165, 219, 49, c);
1898          t.drawFastHLine(165, 220, 49, c);
1899        }
1900        // Axis
1901        else
1902        {
1903          t.drawRect(x(), 23, 45, 196, c);
1904          t.drawRect(x() + 1, 24, 43, 194, c);
1905        }
1906      }
1907
1908      void drawSeek(int32_t newSeek)
1909      {
1910        int8_t ns = percentage(newSeek);
1911      int8_t s = percentage(seek);
1912
1913        int16_t nsy = y(newSeek);
1914        int16_t sy = y(seek);
1915
1916      uint8_t left = x() + 2;
1917
1918        // WIPE TEXT
1919        if(ns < 0 && s >= 0)
1920        t.fillRect(left, 105, 41, 14, WHITE);
1921        else if(ns >= 0 && s <= 0)
1922          t.fillRect(left, 124, 41, 14, WHITE);
1923
1924        // RECT
1925        if(ns <= 0 && s > 0)
1926          t.fillRect(left, yZero, 41, sy - yZero, WHITE);
1927
1928        if(ns >= 0 && s < 0)
1929          t.fillRect(left, sy, 41, yZero - sy, WHITE);
1930
1931        if(ns < 0)
1932        {
1933          if(ns > s)
1934            t.fillRect(left, sy, 41, nsy - sy, WHITE);
1935          else
1936            t.fillRect(left, nsy, 41, sy - nsy, BLUE);
1937        }
1938        else
1939        {
1940          if(ns > s)
1941            t.fillRect(left, sy, 41, nsy - sy, BLUE);
1942          else
1943            t.fillRect(left, nsy, 41, sy - nsy, WHITE);
1944        }
1945
1946        // ZERO
1947        if(ns == 0)
1948          t.drawFastHLine(left, yZero, 41, BLUE);
1949
1950        // TEXT
1951        t.setFont(Calibri13);
1952        t.setTextArea(left, (ns < 0) ? 124 : 105, 41, 14);
1953        t.setTextColor(BLACK, WHITE);
1954        t.clearTextArea();
1955        t.printAligned(String(ns) + String(F("%")), gTextAlignTopCenter);
1956
1957        // POS
1958        if(newSeek != pos)
1959        {
1960          uint8_t yPos = y(pos);
1961          t.drawFastHLine(x() + 2, yPos + (newSeek < 0 ? 1 : -1), 41, MAGENTA);
1962          t.drawFastHLine(x() + 2, yPos, 41, MAGENTA);
1963        }
1964
1965        // INFO TEXT
1966        drawInfo();
1967      }
1968
1969      void drawFeed(int32_t newSeek)
1970      {
1971        int8_t ns = percentage(newSeek);
1972        int8_t s = percentage(seek);
1973
1974        int16_t nsAngle = y(newSeek);
1975        int16_t sAngle = y(seek);
1976
1977
1978        // REDRAW
1979        if(ns > s)
1980          t.fillArc(320, 218, 154, 45, sAngle, nsAngle, BLUE);
1981        else if(ns < s)
1982          t.fillArc(320, 218, 154, 45, nsAngle, sAngle, GRAY);
1983
1984        // POS
1985        //if(newSeek != pos)
1986        //  t.fillArc(320, 218, 154, 45, min(-1, max(-90, y(pos))), min(0, max(-89, y(pos))), MAGEN-
          TA);
1987
1988        // TITLE
1989        t.setFont(Calibri13);
1990        t.setTextColor(BLACK, WHITE);
1991
1992        t.setTextArea(244, 150, 76, 16);
1993        t.clearTextArea();
1994        t.printAlignedOffseted(String(zAxisPercentage(newSeek)) + String(F("%")), gTextAlign-
          MiddleRight, -10, 0);
1995
1996        t.setTextArea(230, 166, 90, 16);
1997        t.clearTextArea();
1998        float f = float(newSeek) / ZSTEPS;
1999        t.printAlignedOffseted(String(f)+String(MM), gTextAlignMiddleRight, -10, 0);
2000
2001        t.setTextArea(224, 182, 96, 16);
2002        t.clearTextArea();
2003        t.printAlignedOffseted(String(newSeek) + String(F("p")), gTextAlignMiddleRight, -10, 0);
2004      }
2005
2006    public:
2007      Axis()
2008      {
2009        enabled = selected = initialized = isYaxis = isFeed = false;
2010      }
2011
2012      //Axis
2013      Axis(bool yAxis, int32_t startPos)
2014      {
2015        enabled = true;
2016        selected = initialized = isFeed = false;
2017        isYaxis = yAxis;
2018        pos = startPos;
2019
2020        drawAxis();
2021        setPos(startPos);
2022        initialized = true;
2023      }
2024
2025      //Feed
2026      Axis(bool FeedAxis)
2027      {
2028        enabled = isFeed = true;
2029        selected = initialized = isYaxis = false;
2030        pos = 0;
2031
2032        drawAxis();
2033        setPos(0);
2034        initialized = true;
2035      }
2036
2037      void init()
2038      {
2039        initialized = false;
2040        drawAxis();
2041        setPos(pos);
```

```
    initialized = true;
  }

  void enable(bool value)
  {
    if(enabled == value)
      return;

    enabled = value;
    this->select(selected);
  }

  int32_t getPos()
  {
    return pos;
  }

  int32_t getSeek()
  {
    return seek;
  }

  bool isInitialized()
  {
    return initialized;
  }

  bool isSelected()
  {
    return selected;
  }

  void select(bool doSelect)
  {
    if(selected == doSelect)
      return;

    selected = doSelect;

    if(isFeed && selected)
    {
      setActive(3);
      drawArrow(187);
    }
    else if(selected)
    {
      setActive(isYaxis ? 2 : 1);
      drawArrow(isYaxis ? 130 : 76);
    }

    drawAxisBorder();
    drawInfoTitle();
  }

  bool clicked()
  {
    if(!enabled)
      return false;

    bool click = false;
    if(isFeed)
      click = (_tPoint.y > 60 && _tPoint.y < 218 && _tPoint.x > 200) || (_tPoint.y > 122 &&
_tPoint.y < 218 && _tPoint.x > 162);
    else
      click = _tPoint.x >= (uint16_t)x() && _tPoint.x <= (uint16_t)x() + 45 && _tPoint.y >= 22
&& _tPoint.y <= 22 + 197;

    if(!click)
      return false;

    this->select(true);
    //String ss = isFeed ? F("feedClick") : F("axisClick");

    Debug(F("Axis clicked: "), name().c_str());
    return true;
  }

  void setPos(int32_t newPos)
  {
    pos = newPos;
    setSeek(newPos);
  }

  void setSeek(int32_t newSeek)
  {
    if(seek == newSeek && initialized)
      return;

    if(resetCallback == resetJogging)
    {
      if(isFeed)
        drawFeed(newSeek);
      else
        drawSeek(newSeek);
    }

    seek = newSeek;
  }

  void confirmSeek()
  {
    if(seek == pos)
      return;

    setPos(seek);
  }
};
```

02f_Toolbar.ino

WireBender2\03_TFT\02f_Toolbar.ino

```
static Button toolbar[5 + 1];

void selectToolbar(int32_t val)
{
  if(val < 0)
    return resetEncoder(0);
  else if(val >= 5)
    return resetEncoder(5);

  for(uint8_t i = 0; i < 5; i++)
    if(toolbar[i]._info.selected)
      toolbar[i].select(false);

  if(val > 0 && val <= 5)
    toolbar[val - 1].select(true);
}

int8_t triggerToolbar()
{
  for(uint8_t i = 0; i < 5; i++)
    if(toolbar[i]._info.selected)
      toolbar[i].trigger();

  return 1;
}

int8_t clickToolbar()
{
  for (uint8_t i = 0; i < 5 + 1; i++)
    if (toolbar[i].clicked())
    return i;

  return -1;
}

int8_t resetToolbar()
{
  for (uint8_t i = 0; i < 5 + 1; i++)
    if (toolbar[i]._info.active)
      toolbar[i].draw(false);

  return 1;
}

void terminateToolbar()
{
  for(int i = 0; i < 5 + 1; i++)
    toolbar[i] = Button();
}
```

02g_LineButtons.ino

WireBender2\03_TFT\02g_LineButtons.ino

```
static Line lineButtons[VIEWLINES];

void terminateLineButtons()
{
  terminateLineButtons(true);
}

void terminateLineButtons(bool clear)
{
  if(clear)
    clearLineButtonRect();

  for(int i = 0; i < VIEWLINES; i++)
    lineButtons[i] = Line();
}

void resetLineButtons()
{
for (int i = 0; i < VIEWLINES; i++)
    if (lineButtons[i]._info.active)
      lineButtons[i].draw(false);
}

int8_t clickLineButtons()
{
  for (int i = 0; i < VIEWLINES; i++)
    if (lineButtons[i].clicked())
      return i;

  return -1;
}
```

02h_AxisButtons.ino

WireBender2\03_TFT\02h_AxisButtons.ino

```
static Axis xAxis;
static Axis yAxis;
static Axis zAxis;
```

04a_Helpers.ino

WireBender2\04_Dialog\04a_Helpers.ino

```
void dialogBtn1(const __FlashStringHelper* title, ButtonCallback c = NULL)
{
  if(String(title).length() > 0)
    toolbar[0] = Button(0, 200, 106, 40, title, true, true, c);
  else
    toolbar[0] = Button(0, 200, 106, 40, EMPTY, false, true, NULL);
}

void dialogBtn2(const __FlashStringHelper* title, ButtonCallback c = NULL)
{
  if(String(title).length() > 0)
    toolbar[1] = Button(107, 200, 106, 40, title, true, true, c);
  else
    toolbar[1] = Button(107, 200, 106, 40, EMPTY, false, true, NULL);
}

void dialogBtn3(const __FlashStringHelper* title, ButtonCallback c = NULL)
{
  if(String(title).length() > 0)
    toolbar[2] = Button(214, 200, 106, 40, title, true, true, c);
  else
    toolbar[2] = Button(214, 200, 106, 40, EMPTY, false, true, NULL);
}

void dialogBtn3ViewFile()
{
  toolbar[2] = Button(214, 200, 106, 40, OK, true, true, viewFile);
}

void dialogOK()
{
  dialogBtn3(OK, openJogging);
}
```

04b_Dialogs.ino

WireBender2\04_Dialog\04b_Dialogs.ino

```
void ConfirmDialog(String title, String text, ButtonCallback yes, ButtonCallback no, Button-
Callback cancel = NULL)
{
  initDialog(title, text);

  if(cancel == NULL)
  {
    dialogBtn1(YES, yes);
    dialogBtn2(EMPTY, NULL);
    dialogBtn3(NO, no);
  }
  else
  {
    dialogBtn1(YES, yes);
    dialogBtn2(NO, no);
    dialogBtn3(CANCEL, cancel);
  }
}

void ErrorDialog(String text)
{
  initDialog(F("Error"), EMPTY, text, true);

  dialogBtn1(EMPTY, NULL);
  dialogBtn2(EMPTY, NULL);
  if(errorCallback != NULL)
    dialogBtn3(OK, errorCallback);
  else
    dialogBtn3(OK, openJogging);
}

void InfoDialog(String title, String text, ButtonCallback ok = NULL)
{
  initDialog(title, EMPTY, text, false);

  dialogBtn1(EMPTY, NULL);
  dialogBtn2(EMPTY, NULL);
  if(ok == NULL)
    dialogBtn3(OK, openJogging);
  else
    dialogBtn3(OK, ok);
}

bool EmptyDialog(String title, String text)
{
  initDialog(title, EMPTY, text, false);
  dialogBtn1(EMPTY, NULL);
  dialogBtn2(EMPTY, NULL);
  dialogBtn3(EMPTY, NULL);
  return true;
}

void Dialog(String title, String text, ButtonCallback ok, bool warning = false )
{
  initDialog(title, EMPTY, text, warning);

  dialogBtn1(EMPTY, NULL);
  dialogBtn2(EMPTY, NULL);
  dialogBtn3( warning ? OK : CLOSE, ok);
}

void UsbDialog(String title, String text)
{
  initDialog(title, EMPTY, EMPTY, false);
  drawUsbText(text);
  bmpDraw(F("/GFX/USB.565"), 24, 78);

  dialogBtn1(EMPTY, NULL);
  dialogBtn2(EMPTY, NULL);
  dialogBtn3(CANCEL, cancelClick);
}
```

04c_Click.ino

WireBender2\04_Dialog\04c_Click.ino

```
int8_t clickDialog()
{
  for(int i = 0; i < 3; i++)
    if(toolbar[i].clicked())
      return i;

  return -1;
}
```

04d_Draw.ino

WireBender2\04_Dialog\04d_Draw.ino

```
void drawDialogTitle(String title, bool warning = false)
{
  uint16_t color = (warning) ? MAGENTA : BLUE;

  t.setFont(Calibri24);
  t.setTextColor(WHITE, color);
  t.setTextArea(0, 0, 200, 45);
  t.clearTextArea();
  t.printAligndOffseted(title, gTextAlignMiddleLeft, 10, 0);
  t.setFont(Calibri13);
}

void drawDialogSubtitle(String subtitle)
{
  t.setFont(Calibri24);
  t.setTextArea(200, 0, 120, 45);
  t.setTextColor(WHITE, BLUE);
  t.clearTextArea();
  t.printAligndOffseted(subtitle, gTextAlignMiddleRight, -10, 0);
  t.setFont(Calibri13);
}

void drawUsbText(String text)
{
  drawDialogText(text, 85, 60, 225, 95);
}

void drawDialogText(String text)
{
  drawDialogText(text, 10, 55, 300, 135);
}

void drawDialogText(String text, int16_t left, int8_t top, int16_t width, int16_t height)
{
  t.setFont(Calibri13);
  t.setTextColor(BLACK, WHITE);
  t.setTextArea(left, top, width, height);
  t.clearTextArea();

  while(text.startsWith(HASHTAG))
    text = text.substring(1);

  uint16_t start = text.indexOf(HASHTAG);
  if(start < 0)
  {
    t.printAligned(text, gTextAlignMiddleCenter);
    return;
  }

  uint8_t count = 1;
  while(true)
  {
    uint16_t end = text.indexOf(HASHTAG, start + 1);
    if(end < 0 || end >= text.length())
      break;
    else
    {
      start = end;
      count++;
    }
  }
  if(text.lastIndexOf(HASHTAG) < text.length())
    count++;

  start = 0;
  uint8_t offset = (135 - count * t.getFontHeight(Calibri13) + 1) / 2;
  for(uint8_t i = 0; i <= count; i++)
  {
    if(start != 0)
      start++;

    uint8_t end = text.indexOf(HASHTAG, start);
    if(end <= 0)
      end = text.length();

    t.printAlignedOffseted(text.substring(start, end), gTextAlignTopCenter, 0, offset + i *
(t.getFontHeight(Calibri13) + 1));

    if(end == text.length())
      break;

    start = end;
  }
}
```

04e_Init.ino

WireBender2\04_Dialog\04e_Init.ino

```
void initDialog(String title, String subtitle, String text, bool warning)
{
  ButtonCallback callbackCache = errorCallback;
  terminatePreviousView();

  clickCallback = clickDialog;
  resetCallback = resetDialog;
  errorCallback = callbackCache;

  t.fillScreen(WHITE);

  drawDialogTitle(title, warning);

  uint16_t color = (warning) ? MAGENTA : BLUE;
  if(subtitle.length() < 1)
    t.fillRect(200, 0, 120, 45, color);
  else
    drawDialogSubtitle(subtitle);

  if(String(text).length() > 0)
    drawDialogText(text);

  screensaverOff();
}

void initDialog(String title, String text)
{
  initDialog(title, EMPTY, text, false);
}

int8_t resetDialog()
{
  for (int i = 0; i < 3; i++)
    if(toolbar[i]._info.active)
      toolbar[i].draw(false);

  return 1;
}
```

05a_Touch.ino

WireBender2\05_General\05a_Touch.ino

```
Point2d getPoint()
```

```
2463 {
2464   TSPoint p = ts.getPoint();
2465   Point2d result;
2466   result.x = (uint16_t) map(constrain(p.y, 100, 900),100, 900, 300, 0);
2467   result.y = (uint8_t) map(constrain(p.x, 140, 900), 140, 900, 0, 240);
2468
2469   if(p.y < 100 || p.y > 900)
2470     result.x = 10000;
2471
2472   if(p.x < 140 || p.y > 900)
2473     result.y = 255;
2474
2475   result.z = p.z;
2476   return result;
2477 }
2478
2479 void touchLooper() {
2480   _tPoint = getPoint();
2481
2482   unsigned long elapsed = millis() - lastClick;
2483
2484   if (elapsed > 200 && config.touched)
2485   {
2486     if(resetCallback != NULL)
2487       resetCallback();
2488
2489     config.touched = false;
2490   }
2491
2492   if ( _tPoint.z > 100) {
2493     if(screensaverOff())
2494       return;
2495
2496     if(clickCallback != NULL)                    //Jogging, Viewer or Browser
2497       config.touched = clickCallback() > -1;
2498   }
2499 }
```

05b_bootUp.ino

WireBender2\05_General\05b_bootUp.ino

```
2500 void bootUp()
2501 {
2502   if(config.booted)
2503   {
2504     String msg = F("The driver module has restarted.");
2505     if(encoderSettings.b8)
2506       ErrorDialog(msg);
2507     else
2508       Debug(msg);
2509
2510     return;
2511   }
2512
2513   initTFT();
2514
2515   //***** SD Card *******
2516   if (!sd.begin(SD_CS, SD_SPI_SPEED))
2517   {
2518     ErrorDialog(F("_SD_INIT_ERROR#-#Please check if an SD-Card is inserted#and if it's
      formatted correctly."));          //return sdWarning();
2519     dialogBtn3(EMPTY, NULL);
2520     digitalWrite(LITE, HIGH);
2521     return;
2522   }
2523   //***** SD Card *******
2524
2525
2526   xAxis = Axis();
2527   yAxis = Axis();
2528   zAxis = Axis();
2529
2530   screenLoading();
2531   uint8_t success = loadSettings();
2532   screensaverOff();
2533
2534   if(success == 2)
2535     config.booted = true;
2536   else if(success == 1)
2537     return ConfirmDialog(F("WireBender"), F("Not all settings were transferred,#to the driver
      module.#Do you want to load the settings again?"), reloadSettings, openJogging);
2538   else
2539     return;
2540
2541   //initViewer();
2542   initJogging();
2543 }
2544
2545 void screenLoading()
2546 {
2547   if(terminateCallback != NULL)
2548   terminateCallback();
2549
2550   clickCallback = NULL;
2551
2552   t.fillScreen(WHITE);
2553
2554   //Logo & Schrift
2555   bmpDraw(F("/GFX/uibk.565"), 10, 10);
2556   t.setFont(Calibri24);
2557   t.setTextColor(BLACK, WHITE);
2558   t.setTextArea(92, 16, 220, 60);
2559
2560   t.printAligned(F("Universitaet"), gTextAlignTopLeft);
2561   t.printAligned0ffseted(F("Innsbruck"), gTextAlignTopLeft, 0, 28);
2562
2563
2564   t.setFont(Calibri13);
2565   t.setTextArea(92, 87, 220, 60);
2566   t.printAligned(F("Institut für"), gTextAlignTopLeft);
2567   t.printAligned0ffseted(F("experimentelle Architektur.Hochbau"), gTextAlignTopLeft, 0, 15);
2568   t.printAligned0ffseted(F("Student: Florian Frank"), gTextAlignTopLeft, 0, 45);
2569
2570   t.setTextArea(0, 0, 320, 240);
2571   t.printAligned0ffseted(F("WireBender v"), gTextAlignBottomRight, -35, -6);
2572   t.printAligned0ffseted(String(VERSION), gTextAlignBottomRight, -10, -6);
2573
2574   //bmpDraw(F("/GFX/logo.565"), 31, 75);
2575 }
```

05c_Calibration.ino

WireBender2\05_General\05c_Calibration.ino

```
2576 void Calibrate()
2577 {
2578   if(!Post(F("calibrate")))
2579   {
2580     ErrorDialog(eText(lastError)+String(F("#-#The calibration couldn't be started.")));
2581     return;
2582   }
2583
2584   initDialog(F("Kalibration"), F("The calibration started.#Please wait..."));
2585   dialogBtn1(EMPTY, NULL);
2586   dialogBtn2(EMPTY, NULL);
2587   dialogBtn3(EMPTY, NULL);
2588   lastAck = 0;
2589   receiveCallback = CalibrateCallback;
2590 }
2591
2592 void CalibrateCallback(String text)
2593 {
2594   drawDialogText(text);
2595
2596   if
2597   (
2598     text.indexOf(F("be reached")) >= 0 ||
2599     text.indexOf(F("unexpected emergency stop")) >= 0 ||
2600     text.indexOf(F("ended unexpected")) >= 0 ||
2601     text.indexOf(F("calibration was successful")) >= 0
2602   )
2603   {
2604     xAxis.setPos(0);
2605     yAxis.setPos(0);
2606     dialogOK();
2607     receiveCallback = NULL;
2608   }
2609 }
```

05d_Eastereggs.ino

WireBender2\05_General\05d_Eastereggs.ino

```
2610 void Easteregg(String type)
2611 {
2612   Easteregg(type, false);
2613 }
2614
2615 void Easteregg(String type, bool isPlayerInterpreter)
2616 {
2617   terminatePreviousView();
2618   t.fillScreen(WHITE);
2619
2620   clickCallback = clickEasteregg;
2621   resetCallback = NULL;
2622
2623   uint8_t one_width = 72;
2624   uint8_t one_height = 58;
2625   String one = F("/GFX/DAUMEN.565");
2626
2627   uint8_t two_width = 71;
2628   uint8_t two_height = 78;
2629   String two = F("/GFX/DAUMEN_HOCH.565");
2630
2631   if(type == F("wink"))
2632   {
2633     one_width = one_height = two_width = two_height = 80;
2634     one = F("/GFX/WINK_1.565");
2635     two = F("/GFX/WINK_2.565");
2636   }
2637
2638   bmpDraw(one, 160 - one_width / 2, 120 - one_height / 2);
2639   uint32_t start = millis();
2640   screensaverOff();
2641
2642   while(millis() - start < 1000 && clickEasteregg() == -1)
2643     delayMicroseconds(500);
2644
2645   bmpDraw(two, 160 - two_width / 2, 120 - two_height / 2);
2646
2647   if(isPlayerInterpreter)
2648     toolbar[5] = Button(0, 0, 320, 240, F("Close"), true, false, openViewer, NULL);
2649   else
2650     toolbar[5] = Button(0, 0, 320, 240, F("Close"), true, false, openJogging, NULL);
2651 }
2652
2653 int8_t clickEasteregg()
2654 {
2655   return clickToolbar();
2656 }
```

05e_Shutdown.ino

WireBender2\05_General\05e_Shutdown.ino

```
2657 void Shutdown()
2658 {
2659   ConfirmDialog(F("Shutdown"), F("Choose \"Cancel\" if you have to remove some wire!#Do
      you want to run to parking position#before shutdown?"), ParkingPosition, ImmediateShut-
      down, CancelShutdown);
2660
2661   receiveCallback = ShutdownCallback;
2662 }
2663
2664 void ShutdownCallback(String text)
2665 {
2666   if(text == F("PARKING_POSITION"))
2667     ImmediateShutdown(NULL);
2668   else if( text.startsWith(F("POS:")) )
2669     SavePositions(text);
2670 }
2671
2672 void ParkingPosition(ButtonInfo* i)
2673 {
2674   if(!Post(F("PARKING_POSITION")))
2675     ShutdownAgain();
2676
2677   EmptyDialog(F("Shutdown"), F("Moving to parking position.#Please wait..."));
2678 }
2679
2680 void ImmediateShutdown(ButtonInfo* i)
2681 {
2682   if( !Post(F("SHUTDOWN")) )
2683     return ShutdownAgain();
2684
2685   if( !Post(F("GET_POS")) )
2686     return ShutdownAgain();
2687 }
2688
2689 void CancelShutdown(ButtonInfo* i)
2690 {
2691   receiveCallback = NULL;
2692   openJogging(i);
2693 }
2694
2695 void ShutdownAgain()
2696 {
2697   drawDialogText(F("Current Position could not be saved. You will have to#recalibrate on the
      next start of the device.#Please click OK to try again."));
2698   dialogBtn3(OK, ImmediateShutdown);
2699 }
2700
2701 void SavePositions(String text)
2702 {
2703   drawDialogText(F("Saving current position..."));
2704
2705   axisData x = parseAxisCode(text, XCHAR);
2706   axisData y = parseAxisCode(text, YCHAR);
2707
2708   if(!x.isValid || !y.isValid)
2709     return ShutdownAgain();
2710
2711   if ( !SaveSetting(F("X Pos"), x.pos) || !SaveSetting(F("Y Pos"), y.pos) )
2712     return ShutdownAgain();
2713
2714   Post(F("SHUTDOWN_SUCCEEDED"));
2715
2716   dialogBtn1(EMPTY, NULL );
2717   dialogBtn2(EMPTY, NULL );
2718   dialogBtn3(EMPTY, NULL );
2719   bmpDraw(F("/GFX/UNPLUG.565"), 35, 55);
2720 }
```

05f_ManualCommand.ino

WireBender2\05_General\05f_ManualCommand.ino

```
2721 void ManualCommand(String command)
2722 {
2723   Log("Manual:"+command);
2724   LineInfo info;
2725   interpreteCodeLine(command, &info);
2726
2727   if(!info.isValid || info.isComment)
2728   {
2729     Log(F("The manual command is a not a valid command."));
2730     return;
2731   }
2732
2733   if(info.isCommand)
2734   {
2735     doCommand(command);
2736     Log(F("The manual command has executed successfully."));
2737     return;
2738   }
2739
2740   if(Post(!Zeros(0)+String(SPACE)+command) != 1)
2741     return ErrorDialog(String(F("The manual command:#"))+command+String(F("#couldn't be
      sent to the driver module.")));
2742
2743   receiveCallback = ManualAck;
2744   setPositions(command);
2745 }
2746
2747 void ManualAck(String text)
2748 {
2749   if(text != F("ACK0"))
2750   {
2751     Log(F("The manual command has failed."));
2752     return ErrorDialog(String(F("The manual command failed#-#"))+text);
2753   }
2754
2755   xAxis.confirmSeek();
2756   yAxis.confirmSeek();
2757   zAxis.confirmSeek();
2758
2759   Log(F("The manual command has executed successfully."));
2760   receiveCallback = NULL;
2761 }
```

06a_Interpreter.ino

WireBender2\06_SD_Player\06a_Interpreter.ino

```
2762 axisData parseAxisCode(String code, const __FlashStringHelper* name)
2763 {
2764   axisData result;
2765   code.trim();
2766   initAxisData(&result, name);
2767
2768   int16_t pos = code.indexOf(name);
2769   if( pos < 0 )
2770     return result;
2771
2772   int16_t len = (code.indexOf(SPACE, pos) > 0) ? code.indexOf(SPACE, pos) : code.length();
2773
2774   //Code f?r Achsparameter
2775   String sub = code.substring(pos + 1, len);
2776   sub.trim();
2777
2778   //Trennzeichen Indexe finden
2779   int16_t spd = sub.indexOf(DBLPT);
2780   int16_t acc = sub.indexOf(SLASH);
2781   len = sub.length();
2782
2783   //Position aus Achsparameter lesen
2784   int16_t start = 0;
2785   int16_t ende = spd;
2786
2787   if(ende < 0)
2788     ende = acc;
2789   if(ende < 0)
2790     ende = len;
2791
2792   if(!setValue(&result, 1, sub, start, ende, len))
2793     return result;
2794
2795   //Speed aus Achsparameter lesen
2796   start = spd + 1;
2797   ende = acc;
2798
2799   if(ende < 0)
2800     ende = len;
2801
2802   if(!setValue(&result, 2, sub, start, ende, len))
2803     return result;
2804
2805   //Acceleration aus Achsparameter lesen
2806   setValue(&result, 3, sub, acc + 1, len, len);
2807   return result;
2808 }
2809
2810 void interpreteCodeLine(String code, LineInfo* i)
2811 {
2812   code.toLowerCase();
2813   //Debug(String(F("interpreting: ")+code);
2814
2815   i->interpreted = true;
2816   i->isValid = i->isCommand = i->isComment = false;
2817   if( code.substring(0,1) == String(HASHTAG) )
2818   {
2819     i->isValid = i->isComment = true;
2820     return;
2821   }
2822
2823   if( isCommand(code) )
2824   {
2825     i->isValid = i->isCommand = true;
2826     return;
2827   }
2828
2829   axisData x = parseAxisCode(code, XCHAR);
2830   axisData y = parseAxisCode(code, YCHAR);
2831   axisData z = parseAxisCode(code, ZCHAR);
2832
2833   if(
2834     code.length() > 0 &&
2835     !x.isValid &&
2836     !y.isValid &&
2837     !z.isValid
2838   )
2839   return;
2840
2841   i->isValid = true;
2842 }
2843
2844 bool isCommand(String code)
2845 {
2846   if(code.startsWith(F("wait ")) && code.length() > 5 && isFloat(code.substring(5)))
2847     return true;
2848
2849   else if(code.startsWith(F("trigger 1")) || code.startsWith(F("trigger 2")))
2850     return true;
2851
2852   else if(code.startsWith(F("great")) || code.startsWith(F("wink")))
2853     return true;
2854
2855   return false;
2856 }
2857
2858 void doCommand(String code)
2859 {
2860   code.toLowerCase();
2861   if(!isCommand(code))
2862     return;
2863
2864   if(code.startsWith(F("wait ")) && code.length() > 5 && isFloat(code.substring(5)))
2865     Wait(code.substring(5).toFloat());
2866
2867   else if(code.startsWith(F("trigger 1")) || code.startsWith(F("trigger 2")))
2868     Trigger(code.substring(8).toInt());
2869
2870   else if(code == F("great") || code == F("wink"))
2871     Easteregg(code, true);
2872 }
2873
2874 void setPositions(String code)
2875 {
2876   if( code.substring(0,1) == String(HASHTAG) )
2877     return;
2878
2879   axisData x = parseAxisCode(code, XCHAR);
2880   axisData y = parseAxisCode(code, YCHAR);
2881   axisData z = parseAxisCode(code, ZCHAR);
2882
2883   if(x.isValid)
2884     xAxis.setPos(x.pos);
2885
2886   if(y.isValid)
2887     yAxis.setPos(y.pos);
2888
2889   if(z.isValid)
2890     zAxis.setPos(z.pos);
2891 }
```

06b_Helpers.ino

WireBender2\06_SD_Player\06b_Helpers.ino

```
2892 bool isReady()
2893 {
2894   if(config.programRunning)
2895     return false;
2896
2897   if(String(workingPath).length() < 1)
2898     return false;
```

```
    if(!sd.exists(workingPath))
      return false;

    return true;
}

void selectLine(int32_t lineNumber)
{
    if(lineNumber < 0)
      return resetEncoder(0, selectLine);

  if(lineNumber >= viewFileLines)
      return resetEncoder(viewFileLines - 1, selectLine);

    selectedLine = lineNumber;
    refreshPlayerInteractives();
}

uint32_t parseAck(String command)
{
    if(command.length() < 4)
      return ErrorLog(67);

    String numStr = command.substring(3);
    if(!isInt(numStr))
      return ErrorLog(68);

    return numStr.toInt();
}

void initAxisData(axisData* ad, const __FlashStringHelper* name)
{
    if(String(name) == String(F("X")))
      ad->axis = 1;
    else if(String(name) == String(F("Y")))
      ad->axis = 2;
    else if(String(name) == String(F("Z")))
      ad->axis = 3;
    else
      ad->axis = 0;
}

bool setValue(axisData* ad, uint8_t type, String sub, int16_t start, int16_t ende, int16_t len)
{
    String val = sub.substring(start, ende);
    ad->isValid = isInt(val) || isFloat(val);

    if(!ad->isValid)
      return false;

    if(type == 1)
      ad->pos = val.toInt();
    else if(type == 2)
      ad->speed = val.toFloat();
    else if(type == 3)
      ad->accel = val.toFloat();

    if(ende == len)
      return false;

    return true;
}
```

06c_Click.ino

WireBender2\06_SD_Player\06c_Click.ino

```
void clickPlayerLineBtn(LineInfo* i)
{
  if( !viewerSettings.b8 || config.programRunning )
    return;

  selectLine(i->lineNumber);
}

void clickPlayBtn(ButtonInfo* i)
{
  Play();
}

void clickStopBtn(ButtonInfo* i)
{
  Stop();
}
```

06d_Draw.ino

WireBender2\06_SD_Player\06d_Draw.ino

```
void drawPlayerLineBtn(LineInfo* i)
{
  // Wenn schon aktiv und selectedLine dieser LineBtn ist
  //if(i->active && (i->lineNumber == selectedLine))
    //return;

  uint16_t backColor = (i->active) ? GRAY : WHITE;
  uint16_t fontColor = BLACK;

  uint8_t y = i->lineNumber % VIEWLINES * 13 + 28;
  t.setTextArea(46, y, 274, 13);

  String code = EMPTY;
  SdFile file;
  if(!openFile(&file, currentPath))
    return;

  code = readLine(&file, i->seek);
  code.trim();

  //Play Modus: Ist die Codezeile Interpretiert?
  if(!i->interpreted)
    interpreteCodeLine(code, i);

  if( i->lineNumber == selectedLine )
  {
    i->active = true;
    backColor = config.programRunning ? MAGENTA : GRAY;
  }

  if( i->isComment )
    fontColor = ( i->active ) ? WHITE : GRAY;
  else if( i->isCommand )
    fontColor = BLUE;
  else if( !i->isValid )
    fontColor = RED;

  t.setTextColor(fontColor, backColor);
  t.clearTextArea();
  t.printAlignedOffseted(lZeros(i->lineNumber + 1)+SPACE+code, gTextAlignMiddleLeft, 4, 0);
  file.close();
}

void refreshPlayerLines()
{
  if( selectedLine < (uint16_t) viewerOffset * VIEWLINES ||
    selectedLine >= (uint16_t) ( viewerOffset + 1 ) * VIEWLINES )
    return;

  for(uint8_t i = 0; i < VIEWLINES; i++)
    if( i != selectedLine % VIEWLINES &&
      lineButtons[i]._info.active )
      lineButtons[i].draw(false);
    else if(i == selectedLine % VIEWLINES)
      lineButtons[i].draw(true);
}

void refreshPlayerInteractives()
{
  if(clickCallback == clickViewer && viewerSettings.b8) // Viewer Page
  {
    resetViewerToolbar();
    if( config.programRunning )
    {
      setViewerOffset( selectedLine / viewFileLines );
      lineButtons[selectedLine % VIEWLINES].draw(true);
    }
    refreshPlayerLines();
  }
  else if(clickCallback == clickJogging)
  // Jogging Page
    resetJoggingToolbar();
}
```

06e_Errors.ino

WireBender2\06_SD_Player\06e_Errors.ino

```
void NotExistingError(SdFile* file = NULL)
{
  if(file != NULL)
    file->close();

  setWorkingPath(EMPTY);

  String s1 = (config.programRunning) ?
    F("The program routine couldn't be started.") :
    F("The program routine has been stopped.");

  Stop();

  PlayerErrorDialog(
    s1+String(F("#Die Datei ,,"))+
    shortName(workingPath)+
    String(F("#existiert nicht."))
  );
}

void LineError(SdFile* file = NULL)
{
  if(file != NULL)
    file->close();


  String s1 = (config.programRunning) ?
    F("The program routine couldn't be started.") :
    F("The program routine has been stopped.");

  Stop();

  PlayerErrorDialog(
    s1+String(F("#The line ,,"))+
    String(selectedLine + 1)+
    String(F(",, couldn't be found in:#"))+
    shortName(workingPath)
  );
  selectedLine = 0;
}

void SendError(SdFile* file = NULL)
{
  if(file != NULL)
    file->close();

  String s1 = eText(lastError)+String(F("#-#"));

  Stop();

  PlayerErrorDialog(
    s1+
    String(F("The program routine has been stopped."))+
    String(F("#The line ,,"))+
    String(selectedLine + 1)+
    String(F(",, couldn't be sent to the driver module."))
  );
}

void AckError()
{
  String s1 = eText(lastError)+String(F("#-#"));

  Stop();

  PlayerErrorDialog(
    s1+
    String(F("The program routine has been stopped."))+
    String(F("#The command couldn't be acknowledged."))
    );
}
```



```
void PlayerErrorDialog(String text)
{
  setViewerPath(workingPath);
  Debug(text);
  Dialog(
    F("Player"),
    text,
    openViewer,
    true
  );
}
```

06f_Player.ino

WireBender2\06_SD_Player\06f_Player.ino

```
void playerLooper()
{
  if(!config.programRunning || lastAck <= selectedLine)
    return;

  // WAIT
  if(waitTime > 0)
  {
    if(millis() - lastWait < waitTime)
      return;

    waitTime = 0;
  }

  lastAck = 0;

  SdFile file;
  if(!openFile(&file, workingPath))
    return NotExistingError(&file);

  playerSeek = getLineSeek(&file, selectedLine);
  if(selectedLine > 0 && playerSeek < 1)
    return LineError(&file);

  String code = readLine(&file, playerSeek);
  if(code.length() < 1)
  {
    Debug(String(F("Line "))+String(selectedLine + 1)+String(F(" is EMPTY")));
  file.close();
    return Skip();
  }

  LineInfo info;
  interpreteCodeLine(code, &info);
  if(info.isComment || info.isCommand)
  {
    Debug(String(F("Line "))+String(selectedLine + 1)+String(info.isComment ? F(" is COMMENT") : F(" is COMMAND")));
    if(info.isCommand)
      doCommand(code);
    file.close();
    return Skip();
  }

  if(Post(lZeros(selectedLine + 1)+String(SPACE)+code) != 1)
    SendError();

  setPositions(code);

file.close();
}

void Skip()
{
if(Post(lZeros(selectedLine + 1)) != 1)
    SendError();

  return;

  String s1 =
  String(F("Line "))+
  String(selectedLine + 1)+
  String(F(" has been skipped."));

  if(Post(lZeros(selectedLine + 1)) != 1)
    SendError();
  else
    Debug(s1);
}

void AckLine(String text)
{
  uint16_t lineNumber = parseAck(text);
  if(lineNumber < 1)
    return AckError();

  xAxis.confirmSeek();
  yAxis.confirmSeek();
zAxis.confirmSeek();

  SdFile file;
  if(!openFile(&file, workingPath))
    return NotExistingError();


  playerSeek = getLineSeek(&file, lineNumber);
  if(playerSeek < 1 || playerSeek == file.fileSize())
  {
    Stop();

    if(lineNumber == selectedLine + 1 && lineNumber == countLines(&file))
    {
      setViewerPath(workingPath);
      InfoDialog(F("Player"), F("The program routine ended successfully."), openViewer);
    }

    selectLine(0);
    file.close();
    return;
  }
  file.close();

  lastAck = lineNumber + 1;
  //Debug(F("AckLine Seek: "), String(playerSeek).c_str());
  Debug(F("Line "), String(lineNumber).c_str(), String(F(" has been Acked")).c_str());
```

```
  selectLine(lineNumber);
  return;
}

void Play()
{
  if(config.programRunning)
    return;

  receiveCallback = NULL;
  if(String(workingPath).length() < 1)
  {
    resetCallback();
    Warning(F("No program routine loaded.#Please load a program routine and start again."));
    return;
  }

  SdFile file;
  if(!openFile(&file, workingPath))
    return NotExistingError();

  playerSeek = getLineSeek(&file, selectedLine);
  file.close();
  if(selectedLine > 0 && playerSeek < 1)
    return LineError();

  lastAck = selectedLine + 1;
  config.programRunning = true;
  refreshPlayerInteractives();

  Log(
    String(F("Program routine \"")+
    shortName(workingPath)+
    String(F("\" started on line "))+
    String(selectedLine + 1)+
    String(F(".."))
  );
}

void Stop()
{
  config.programRunning = false;
  lastSentTime = 0;
  lastAck = selectedLine + 1;
  Post("Reset");
  refreshPlayerInteractives();
  //resetEncoder(selectedLine, selectLine);
}
```

06g_Commands.ino

WireBender2\06_SD_Player\06g_Commands.ino

```
void Wait(float seconds)
{
  lastWait = millis();
  waitTime = floor(seconds * 1000.0);
}

void Trigger(int32_t pin)
{
  if(pin == 1)
    digitalWrite(TRIGGER1, HIGH);
  else if(pin == 2)
    digitalWrite(TRIGGER2, HIGH);
  else
    return;

  lastTrigger = millis();
  if(lastTrigger == 0)
    lastTrigger = 1;
}

void triggerLooper()
{
  if(lastTrigger == 0 || lastTrigger - millis() > TRIGGER_TIME)
    return;

if(digitalRead(TRIGGER1) == HIGH)
    digitalWrite(TRIGGER1, LOW);

  if(digitalRead(TRIGGER2) == HIGH)
    digitalWrite(TRIGGER2, LOW);

  lastTrigger = 0;
}

void initTriggers()
{
  pinMode(TRIGGER1, OUTPUT);
  digitalWrite(TRIGGER1, LOW);

  pinMode(TRIGGER2, OUTPUT);
  digitalWrite(TRIGGER2, LOW);
}
```

07aa_Click.ino

WireBender2\07_Jogging_P1\07a_Toolbar\07aa_Click.ino

```
void clickViewerBtn(ButtonInfo* i)
{
  initViewer();
}

void clickManualBtn(ButtonInfo* i)
{
  ManualCommand(String(F("Z"))+String(zAxis.getPos())+String(F(":2000/800")));
}

void openJogging(ButtonInfo* i)
{
  initJogging();
}

void ZeroPosition()
{
  String cmd = F("0 x0:2000/2000 y0:2000/2000");
```

```
3346    xAxis.setPos(0);
3347    yAxis.setPos(0);
3348    if(Post(cmd) != 1)
3349      SendError();
3350  }
3351
3352  void insertCallback(String text)
3353  {
3354    uint16_t lineNumber = parseAck(text);
3355    if(lineNumber > 1)
3356      return AckError();
3357
3358    if(lineNumber == 0)
3359    {
3360      //xAxis.confirmSeek();
3361      //yAxis.confirmSeek();
3362      String cmd = F("1 z13695:400/0");
3363      setPositions(cmd);
3364      if(Post(cmd) != 1)
3365        SendError();
3366    }
3367    else if(lineNumber == 1)
3368    {
3369      config.programRunning = false;
3370      resetCallback();
3371      receiveCallback = NULL;
3372      Debug(F("Wire insertion was successful."));
3373      zAxis.setPos(415);
3374      setActive(3);
3375    }
3376  }
3377
3378  void clickInsertBtn(ButtonInfo* i)
3379  {
3380    if(config.programRunning)
3381      return ErrorDialog(F("The insertion of wire while a program routine#is running, is not
     allowed."));
3382
3383    config.programRunning = true;
3384    receiveCallback = insertCallback;
3385    ZeroPosition();
3386  }
3387
3388  void removeCallback(String text)
3389  {
3390    uint16_t lineNumber = parseAck(text);
3391    if(lineNumber >= 1)
3392      return AckError();
3393
3394    xAxis.confirmSeek();
3395    yAxis.confirmSeek();
3396    config.programRunning = false;
3397    resetCallback();
3398    receiveCallback = NULL;
3399    setActive(3);
3400  }
3401
3402  void clickRemoveBtn(ButtonInfo* i)
3403  {
3404    if(config.programRunning)
3405      return ErrorDialog(F("Removing wire while a program routine#is running, is not allo-
     wed."));
3406
3407    config.programRunning = true;
3408    receiveCallback = removeCallback;
3409    ZeroPosition();
3410  }
```

07ab_Draw.ino

WireBender2\07_Jogging_P1\07a_Toolbar\07ab_Draw.ino

```
3411  void drawViewerBtn(ButtonInfo* i)
3412  {
3413    if(i->active)
3414      bmpDraw(F("GFX/Toolbar/Programm_Pink.565"), i->x+15, i->y+12);
3415    else
3416      bmpDraw(F("GFX/Toolbar/Programm_Blue.565"), i->x+15, i->y+12);
3417  }
3418
3419  void drawSettingsBtn(ButtonInfo* i)
3420  {
3421    if(i->active)
3422      bmpDraw(F("GFX/Toolbar/Settings_Pink.565"), i->x+11, i->y+13);
3423    else
3424      bmpDraw(F("GFX/Toolbar/Settings_Blue.565"), i->x+11, i->y+13);
3425  }
3426
3427  void drawInsert(ButtonInfo* i)
3428  {
3429    t.fillArc(i->x+34, i->y+17, 7, 3, 0, 360, WHITE);
3430    t.fillArc(i->x+34, i->y+48, 23, 3, -90, 45, WHITE);
3431    t.fillTriangle(i->x+15, i->y+20, i->x+23, i->y+24, i->x+15, i->y+29, WHITE);
3432    t.drawFastHLine(i->x, i->y+24, 16, WHITE);
3433    t.drawFastHLine(i->x, i->y+25, 16, WHITE);
3434    t.drawFastHLine(i->x, i->y+26, 16, WHITE);
3435  }
3436
3437  void drawRemove(ButtonInfo* i)
3438  {
3439    t.fillArc(i->x+10, i->y+17, 7, 3, 0, 360, WHITE);
3440    t.fillArc(i->x+10, i->y+48, 23, 3, -45, 89, WHITE);
3441    t.fillTriangle(i->x+35, i->y+20, i->x+43, i->y+24, i->x+35, i->y+29, WHITE);
3442    t.drawFastHLine(i->x+23, i->y+24, 13, WHITE);
3443    t.drawFastHLine(i->x+23, i->y+25, 13, WHITE);
3444    t.drawFastHLine(i->x+23, i->y+26, 13, WHITE);
3445  }
3446
3447  void drawManualBtn(ButtonInfo* i)
3448  {
3449    t.fillArc(i->x+11, i->y+16, 20, 3, 90, 190, WHITE);
3450    t.fillTriangle(i->x+24, i->y+18, i->x+34, i->y+18, i->x+29, i->y+10, WHITE);
3451  }
```

07ac_Init.ino

WireBender2\07_Jogging_P1\07a_Toolbar\07ac_Init.ino

```
3452  void initJoggingToolbar()
3453  {
3454    toolbar[0] = Button(0, F("ProgrammViewer"), true, clickViewerBtn, drawViewerBtn);
3455    toolbar[1] = Button(1, F("Insert"), !config.programRunning, clickInsertBtn, drawInsert);
3456    toolbar[2] = Button(2, F("Remove"), !config.programRunning, clickRemoveBtn, drawRemo-
     ve);
3457    toolbar[3] = Button(3, F("Settings"), true, initConfiguration, drawSettingsBtn);
3458    toolbar[4] = Button(4, F("Manual"), false, clickManualBtn, drawManualBtn);
3459    resetToolbar();
3460  }
3461
3462  int8_t resetJoggingToolbar()
3463  {
3464    resetToolbar();
3465    toolbar[1].enable(!config.programRunning);
3466    toolbar[2].enable(!config.programRunning);
3467    return 1;
3468  }
```

07b_Helpers.ino

WireBender2\07_Jogging_P1\07b_Helpers.ino

```
3469  void setActive(uint8_t                axis)
3470  {
3471    switch(axis)
3472    {
3473      case 1:
3474      xAxis.select(true);
3475        yAxis.select(false);
3476        zAxis.select(false);
3477        resetEncoder(xAxis.getPos(), setAxisPos, selectNextAxis);
3478        setEncoderIncrement(100);
3479        break;
3480      case 2:
3481        xAxis.select(false);
3482        yAxis.select(true);
3483        zAxis.select(false);
3484        resetEncoder(yAxis.getPos(), setAxisPos, selectNextAxis);
3485        setEncoderIncrement(100);
3486        break;
3487      case 3:
3488        toolbar[4].enable(true);
3489        xAxis.select(false);
3490        yAxis.select(false);
3491        zAxis.select(true);
3492        resetEncoder(zAxis.getPos(), setAxisPos, selectNextAxis);
3493        setEncoderIncrement(100);
3494        break;
3495      default:
3496        toolbar[4].enable(false);
3497        eraseArrow();
3498        xAxis.select(false);
3499        yAxis.select(false);
3500        zAxis.select(false);
3501        resetEncoder(0, NULL, selectNextAxis);
3502        break;
3503    }
3504  }
3505
3506  void setAxisPos(int32_t val)
3507  {
3508    if(xAxis.isSelected())
3509      xAxis.setPos(val);
3510    else if(yAxis.isSelected())
3511      yAxis.setPos(val);
3512    else if(zAxis.isSelected())
3513      zAxis.setPos(val);
3514  }
3515
3516  int8_t selectNextAxis()
3517  {
3518    if(xAxis.isSelected())
3519      setActive(2);
3520    else if(yAxis.isSelected())
3521      setActive(3);
3522    else if(zAxis.isSelected())
3523      setActive(0);
3524    else
3525      setActive(1);
3526
3527    return 1;
3528  }
```

07c_Click.ino

WireBender2\07_Jogging_P1\07c_Click.ino

```
3529  int8_t clickJogging()
3530  {
3531    int8_t clickIndex = clickToolbar();
3532
3533    if(clickIndex >= 0)
3534      return clickIndex;
3535
3536    /*
3537    if(xAxis.clicked())
3538      return 6;
3539
3540    if(yAxis.clicked())
3541      return 7;
3542    */
3543
3544    if(zAxis.clicked())
3545      return 8;
3546
3547    return -1;
3548  }
3549
3550  void clickInfo(ButtonInfo* i)
3551  {
3552    setActive(0);
3553  }
```

07d_Draw.ino

WireBender2\07_Jogging_P1\07d_Draw.ino

```
3554  void eraseArrow()
3555  {
3556    t.fillRect(45, 219, 175, 21, WHITE);
3557
3558    //Umrandung löschen
3559    t.drawFastHLine(0, 191, 47, BLUE);
3560    t.drawFastHLine(0, 192, 47, WHITE);
3561
3562    t.drawFastVLine(45, 191, 49, WHITE);
3563    t.drawFastVLine(46, 191, 49, WHITE);
3564  }
3565
3566  void drawArrow(int pos)
3567  {
3568    eraseArrow();
3569    //Vertikal
3570    t.drawFastVLine(pos, 219, 12, MAGENTA);
3571    t.drawFastVLine(pos + 1, 219, 12, MAGENTA);
3572
3573    //Horizontal
3574    t.drawFastHLine(58, 230, pos - 56, MAGENTA);
3575    t.drawFastHLine(58, 231, pos - 56, MAGENTA);
3576
3577    //Pfeilspitze
3578    t.fillTriangle(58, 224, 58, 236, 47, 230, MAGENTA);
3579
3580    //Umrandung ProttypBtn
3581    t.drawFastVLine(45, 191, 49, MAGENTA);
3582    t.drawFastVLine(46, 191, 49, MAGENTA);
3583    t.drawFastHLine(0, 191, 47, MAGENTA);
3584    t.drawFastHLine(0, 192, 47, MAGENTA);
3585  }
3586
3587  void drawInfoTitle()
3588  {
3589    t.setTextColor(WHITE, BLUE);
3590
3591    bool global = !xAxis.isSelected() && !yAxis.isSelected();
3592
3593    t.setTextArea(266, 4, 54, 16);
3594    t.clearTextArea();
3595    t.printAlignedOffseted(global ? F("Angle") : F("Position"), gTextAlignMiddleLeft, 10, 0);
3596
3597    t.setTextArea(266, 22, 54, 16);
3598    t.clearTextArea();
3599    t.printAlignedOffseted(global ? F("Pitch") : F("Pulses"), gTextAlignMiddleLeft, 10, 0);
3600
3601    t.setTextArea(266, 40, 54, 16);
3602    t.clearTextArea();
3603    t.printAlignedOffseted(F("Radius") , gTextAlignMiddleLeft, 10, 0);
3604
3605    drawInfo();
3606  }
3607
3608  void drawInfo()
3609  {
3610    t.setTextColor(BLACK, GRAY);
3611
3612    float xSeek = float(xAxis.getSeek()) / XSTEPS;
3613    float ySeek = float(yAxis.getSeek()) / YSTEPS;
3614
3615    float radius = (pow(xSeek, 2) + pow(deltaZ, 2))/(2 * xSeek);
3616
3617  // 3rd
3618    t.setTextArea(180, 40, 84, 16);
3619    t.clearTextArea();
3620    t.printAlignedOffseted((abs(radius) <= 999.9f ? String(radius)+String(MM) : String(F("Stra-
     ight"))), gTextAlignMiddleRight, -10, 0);
3621
3622    if(!xAxis.isSelected() && !yAxis.isSelected())
3623    {
3624      // 1st
3625      t.setTextArea(180, 4, 84, 16);
3626      t.clearTextArea();
3627      t.printAlignedOffseted(F("0"), gTextAlignMiddleRight, -10, 0);
3628
3629      // 2nd
3630      t.setTextArea(180, 22, 84, 16);
3631      t.clearTextArea();
3632      t.printAlignedOffseted(String(ySeek)+String(MM), gTextAlignMiddleRight, -10, 0);
3633      return;
3634    }
3635
3636    // 1st
3637    t.setTextArea(180, 4, 84, 16);
3638    t.clearTextArea();
3639    t.printAlignedOffseted(String(xAxis.isSelected() ? xSeek : ySeek)+String(MM), gTextAlign-
     MiddleRight, -10, 0);
3640
3641    // 2nd
3642    t.setTextArea(180, 22, 84, 16);
3643    t.clearTextArea();
3644    t.printAlignedOffseted(String(xAxis.isSelected() ? xAxis.getSeek() : yAxis.get-
     Seek())+String(F("p")), gTextAlignMiddleRight, -10, 0);
3645  }
```

07e_Init.ino

WireBender2\07_Jogging_P1\07e_Init.ino

```
3646  void initJogging()
3647  {
3648    terminatePreviousView();
3649
3650    clickCallback = clickJogging;
3651    resetCallback = resetJogging;
3652
3653    t.fillScreen(WHITE);
3654    initJoggingToolbar();
3655    initInfo();
3656    initAxes();
3657    resetEncoder(0, NULL, selectNextAxis);
3658  }
3659
3660  void initInfo(){
3661    toolbar[5] = Button(190, 0, 120, 56, F("Info"), true, false, clickInfo, NULL);
3662    drawInfoTitle();
3663  }
3664
```

```
3665  void initAxes()
3666  {
3667    if(!xAxis.isInitialized())
3668      xAxis = Axis(false, xPos);
3669    else
3670      xAxis.init();
3671
3672    if(!yAxis.isInitialized())
3673      yAxis = Axis(true, yPos);
3674    else
3675      yAxis.init();
3676
3677    if(!zAxis.isInitialized())
3678      zAxis = Axis(true);
3679    else
3680      zAxis.init();
3681  }
3682
3683  int8_t resetJogging()
3684  {
3685    resetJoggingToolbar();
3686    return 1;
3687  }
3688
3689  /*
3690  void initArc()
3691  {
3692    int target = feed;
3693    if (settings.b2) {
3694      int i = 1, step = 5, target = feed;
3695
3696      t.fillArc(fx, fy, 154, 45, -90, 0, GRAY);
3697      for (i = 1; i <= 100; i += step)
3698        posFeed(i);
3699
3700      for (i = 100; i >= target; i -= step)
3701        posFeed(i);
3702
3703      posFeed(target);
3704    }
3705    else {
3706      feed = 0;
3707      posFeed(target);
3708    }
3709
3710    Debug("Animate Feed done: " + String(target));
3711  }
3712  */
```

08aa_Click.ino

WireBender2\08_Browser_P2\08a_Toolbar\08aa_Click.ino

```
3713  void browseUp(ButtonInfo* i)
3714  {
3715    String p = String(currentPath);
3716    p = p.substring(0, p.lastIndexOf('/'));
3717    setBrowserPath(p);
3718  }
3719
3720  void browseHome(ButtonInfo* i)
3721  {
3722    setBrowserPath("");
3723  }
3724
3725  void browsePrev(ButtonInfo* i)
3726  {
3727    setBrowserOffset(browserOffset - 1);
3728  }
3729
3730  void browseNext(ButtonInfo* i)
3731  {
3732    setBrowserOffset(browserOffset + 1);
3733  }
```

08ab_Draw.ino

WireBender2\08_Browser_P2\08a_Toolbar\08ab_Draw.ino

```
3734  void drawHome(ButtonInfo* i)
3735  {
3736    t.fillRect(i->x+17, i->y+23, 11, 9, WHITE);
3737    t.fillTriangle(i->x+13, i->y+24, i->x+22, i->y+16, i->x+31, i->y+24, WHITE);
3738  }
3739
3740  void drawBrowseUp(ButtonInfo* i)
3741  {
3742    t.fillTriangle(i->x+18, i->y+26, i->x+23, i->y+16, i->x+28, i->y+26, WHITE);
3743  }
3744
3745  void drawPrev(ButtonInfo* i)
3746  {
3747    t.fillTriangle(i->x+17, i->y+22, i->x+27, i->y+17, i->x+27, i->y+27, WHITE);
3748  }
3749
3750  void drawNext(ButtonInfo* i)
3751  {
3752    t.fillTriangle(i->x+17, i->y+27, i->x+17, i->y+17, i->x+27, i->y+22, WHITE);
3753  }
3754
3755  void drawClose(ButtonInfo* i)
3756  {
3757    t.drawLine(i->x+15, i->y+31, i->x+28, i->y+17, WHITE);
3758    t.drawLine(i->x+16, i->y+31, i->x+29, i->y+17, WHITE);
3759
3760    t.drawLine(i->x+15, i->y+17, i->x+28, i->y+31, WHITE);
3761    t.drawLine(i->x+16, i->y+17, i->x+29, i->y+31, WHITE);
3762  }
```

08ac_Init.ino

WireBender2\08_Browser_P2\08a_Toolbar\08ac_Init.ino

```
3763  void initBrowserToolbar()
```

```
{
  Debug(F("Browser Toolbar"));
  toolbar[0] = Button(0, F("Home"), false, browseHome, drawHome);
  toolbar[1] = Button(1, F("browseUp"), false, browseUp, drawBrowseUp);
  toolbar[2] = Button(2, F("browsePrev"), false, browsePrev, drawPrev);
  toolbar[3] = Button(3, F("browseNext"), false, browseNext, drawNext);
  toolbar[4] = Button(4, F("Close"), true, openJogging, drawClose);
  resetBrowserToolbar();
}

uint8_t resetBrowserToolbar()
{
  resetToolbar();
  toolbar[0].enable(browserSettings.b1);
  toolbar[1].enable(browserSettings.b2);
  toolbar[2].enable(browserSettings.b3);
  toolbar[3].enable(browserSettings.b4);
  return 1;
}
```

08b_Helpers.ino

WireBender2\08_Browser_P2\08b_Helpers.ino

```
void setBrowserPath(String newPath)
{
  browserOffset = 0;
  newPath.toCharArray(currentPath, 255);
  Debug(String(F("setBrowserPath: ")), currentPath);
  //ramInfo();
  resetEncoder();
  drawFileListing();
}

void setBrowserOffset(int32_t offset)
{
  if(offset < 0 || (offset > browserOffset && !browserSettings.b4))
    return;

  browserOffset = offset;
  drawFileListing();
}

String getFolderItemName(uint16_t index)
{
  char name[40];

  SdFile myDir;
  if(!openFolder(&myDir, currentPath))
    return String(F("FEHLER"));

  SdFile myFile;
  uint16_t n = 0;
  while (n <= index && myFile.openNext(&myDir, O_READ))
  {
    if (!myFile.isHidden())
    {
      myFile.getName(name, 40);
      n++;
    }

  myFile.close();
  }
  myDir.close();
  return String(name);
}
```

08c_Click.ino

WireBender2\08_Browser_P2\08c_Click.ino

```
int8_t clickBrowser()
{
  int8_t clickIndex = clickToolbar();
  if(clickIndex >= 0)
    return clickIndex;

  clickIndex = clickLineButtons();
  if(clickIndex >= 0)
    return clickIndex;

  return -1;
}

void openDirFile(LineInfo* i)
{
  String name = getFolderItemName(i->lineNumber);
  String filePath = String(currentPath)+String(F("/"))+name;

  SdFile myFile;
  if(!myFile.open(filePath.c_str(), O_READ))
  {
    ErrorDialog(String(F("The path:#"))+filePath+String(F("#is invalid.")));
    myFile.close();
    return;
  }

  if(myFile.isDir())
    setBrowserPath(filePath);
  else if(myFile.isFile())
  {
    setViewerPath(filePath);

    String title = F("Read file");
    String text = String(String(F("Do you want to read and load:#"))+shortName(name)+String(F("#as a program routine?")));
    if( String(currentPath) == String(workingPath) )
      initViewer();
    else
      ConfirmDialog(title, text, loadFile, viewFile, openBrowser);
  }
  else
    Debug(filePath);

  myFile.close();
}

void viewFile(ButtonInfo* i)
{
  initViewer();
}

void loadFile(ButtonInfo* i)
{
  if(String(currentPath) != String(workingPath))
    viewerOffset = 0;

  setWorkingPath(currentPath);
  viewFile(i);
}
```

08d_Draw.ino

WireBender2\08_Browser_P2\08d_Draw.ino

```
void drawFolderItemName(LineInfo* i)
{
  uint8_t y = i->lineNumber % BROWSELINES * 24 + 28;
  uint16_t c = (i->active) ? GRAY : WHITE;
  t.setTextArea(46, y, 274, 24);
  t.setTextColor(BLACK, c);
  t.clearTextArea();
  t.printAlignedOffseted(getFolderItemName(i->lineNumber), gTextAlignMiddleLeft, 34, 0);
}

void drawFolderBtn(LineInfo* i)
{
  drawFolderItemName(i);

  uint8_t y = i->lineNumber % BROWSELINES * 24 + 28;
  if(i->active)
    bmpDraw(F("GFX/Browser/Folder_Gray.565"), 46 + 4, y + 4);
  else
    bmpDraw(F("GFX/Browser/Folder_White.565"), 46 + 4, y + 4);
}

void drawFileBtn(LineInfo* i)
{
  drawFolderItemName(i);

  uint8_t y = i->lineNumber % BROWSELINES * 24 + 28;
  if(i->active)
  bmpDraw(F("GFX/Browser/File_Gray.565"), 46 + 6, y + 1);
  else
    bmpDraw(F("GFX/Browser/File_White.565"), 46 + 6, y + 1);
}

bool drawFileListing()
{
  SdFile myDir;
  if(!openFolder(&myDir, currentPath))
  {
    ErrorDialog(String(F("The path:#"))+String(currentPath)+String(F("#doesn't exist.")));
    setViewerPath(SLASH);
    return false;
  }

  terminateLineButtons();
  t.setTextColor(WHITE, BLUE);
  t.setTextArea(46, 0, 274, 24);
  t.clearTextArea();
  t.printAlignedOffseted((myDir.isRoot()) ? F("SD-Card") : shortName(currentPath),
gTextAlignMiddleLeft, 35, 0);
  bmpDraw(F("GFX/Browser/Folder_Blue.565"), 50, 4);

  t.setTextColor(BLACK, WHITE);

  SdFile myFile;
  uint8_t n = 0;
  uint8_t offset = browserOffset + 1;
  while (n <= BROWSELINES * offset && myFile.openNext(&myDir, O_READ)) {
    // Skip directories and hidden files.
    if (!myFile.isHidden())
    {
      if(n >= BROWSELINES * browserOffset && n < BROWSELINES * offset)
        lineButtons[n % BROWSELINES] = Line(n, openDirFile, myFile.isSubDir() ? drawFol-
derBtn : drawFileBtn);

      n++;
    }
    myFile.close();
  }
  if(n == 0)
    displayText(46, 24, F("This folder is empty."));

  browserSettings.b1 = !myDir.isRoot();
  browserSettings.b2 = !myDir.isRoot();
  browserSettings.b3 = browserOffset > 0;
  browserSettings.b4 = n > BROWSELINES * offset;
  resetBrowserToolbar();

  myDir.close();

  resetEncoder(browserOffset, setBrowserOffset);
  Debug(F("Files found: "), String(n).c_str());
  return true;
}
```

08e_Init.ino

WireBender2\08_Browser_P2\08e_Init.ino

```
void initBrowser()
{
  terminatePreviousView();

  clickCallback = clickBrowser;
  resetCallback = resetBrowser;
  errorCallback = openBrowser;
  //sd.ls(LS_R); //Auflistung aller Dateien

  t.fillScreen(WHITE);

  if(!drawFileListing())
    return;

  initBrowserToolbar();
}

int8_t resetBrowser()
{
  resetBrowserToolbar();
  resetLineButtons();
  return 1;
}
```


09aa_Click.ino

WireBender2\09_Viewer_P3\09a_Toolbar\09aa_Click.ino

```
void toggleViewerToolbar(ButtonInfo* i)
{
  viewerSettings.b5 = !viewerSettings.b5;
  initViewerToolbar();
}

void prevPage(ButtonInfo* i)
{
  setViewerOffset(viewerOffset - 1);
}

void nextPage(ButtonInfo* i)
{
  setViewerOffset(viewerOffset + 1);
}

void ppToStart(ButtonInfo* i)
{
  setViewerOffset(0);
}

void ppToEnd(ButtonInfo* i)
{
  setViewerOffset(viewFileLines / getViewLines());
}

void openBrowser(ButtonInfo* i)
{
  initBrowser();
}
```

09ab_Draw.ino

WireBender2\09_Viewer_P3\09a_Toolbar\09ab_Draw.ino

```
void drawMenu(ButtonInfo* i)
{
  t.drawFastHLine(i->x+15, i->y+17, 16, WHITE);
  t.drawFastHLine(i->x+15, i->y+18, 16, WHITE);

  t.drawFastHLine(i->x+15, i->y+23, 16, WHITE);
  t.drawFastHLine(i->x+15, i->y+24, 16, WHITE);

  t.drawFastHLine(i->x+15, i->y+29, 16, WHITE);
  t.drawFastHLine(i->x+15, i->y+30, 16, WHITE);
}

void drawBrowse(ButtonInfo* i)
{
  t.fillRect(i->x+14, i->y+20, 17, 12, WHITE);
  t.drawFastHLine(i->x+16, i->y+18, 8, WHITE);
}

void drawStart(ButtonInfo* i)
{
  drawPrev(i);
  t.drawFastVLine(i->x+15, i->y+17, 11, WHITE);
  t.drawFastVLine(i->x+16, i->y+17, 11, WHITE);
}

void drawEnd(ButtonInfo* i)
{
  drawNext(i);
  t.drawFastVLine(i->x+28, i->y+17, 11, WHITE);
  t.drawFastVLine(i->x+29, i->y+17, 11, WHITE);
}

void drawPlay(ButtonInfo* i)
{
  t.fillTriangle(i->x+15, i->y+17, i->x+15, i->y+31, i->x+29, i->y+24, WHITE);
}

void drawStop(ButtonInfo* i)
{
  t.fillRect(i->x+15, i->y+17, 14, 14, WHITE);
}
```

09ac_Init.ino

WireBender2\09_Viewer_P3\09a_Toolbar\09ac_Init.ino

```
void initViewerToolbar()
{
  //Debug(F("Viewer Toolbar: "), String(viewerSettings.b5).c_str());
  if(!viewerSettings.b5)
  {
    toolbar[0] = Button(0, F("Toggle Menu"), true, toggleViewerToolbar, drawMenu);
    toolbar[1] = Button(1, F("PP to Start"), false, ppToStart, drawStart);
    toolbar[2] = Button(2, F("Prev Page"), false, prevPage, drawPrev);
    toolbar[3] = Button(3, F("Next Page"), false, nextPage, drawNext);
    toolbar[4] = Button(4, F("PP to End"), false, ppToEnd, drawEnd);

  }
  else
  {
    toolbar[0] = Button(0, F("Toggle Menu"), true, toggleViewerToolbar, drawMenu);
    toolbar[1] = Button(1, F("Browse"), true, openBrowser, drawBrowse);
    toolbar[2] = Button(2, F("Play"), false, clickPlayBtn, drawPlay);
    toolbar[3] = Button(3, F("Stop"), false, clickStopBtn, drawStop);
    toolbar[4] = Button(4, F("Close Viewer"), true, openJogging, drawClose);

  }
```

```
  resetViewerToolbar();
}

uint8_t resetViewerToolbar()
{
  resetToolbar();
  if(!viewerSettings.b5)
  {
    toolbar[1].enable(viewerSettings.b1);
    toolbar[2].enable(viewerSettings.b2);
    toolbar[3].enable(viewerSettings.b3);
    toolbar[4].enable(viewerSettings.b4);
  }
  else
  {
    toolbar[2].enable( isReady() && viewerSettings.b8 );
    toolbar[3].enable( config.programRunning && viewerSettings.b8 );
  }
  return 1;
}
```

09b_Helpers.ino

WireBender2\09_Viewer_P3\09b_Helpers.ino

```
uint8_t getViewLines()
{
  return viewerSettings.b8 ? VIEWLINES : VIEWLINES + 3;
}

void setViewerPath(String newPath)
{
  viewerOffset = 0;
  newPath.toCharArray(currentPath, 255);
  Debug(String(F("setViewerPath: ")), currentPath);
}

void setWorkingPath(String newPath)
{
  newPath.trim();
  viewerOffset = 0;
  selectedLine = 0;
  newPath.toCharArray(workingPath, 255);
newPath.toCharArray(currentPath, 255);
  Debug(String(F("setWorkingPath: ")), workingPath);
}

void setViewerOffset(int32_t offset)
{
  if(offset < 0 || offset > (int32_t)viewFileLines / getViewLines() || offset == viewerOffset)
  {
    if(offset < 0)
      resetEncoder(0, setViewerOffset);

    if(offset >= (int32_t)viewFileLines / getViewLines())
      resetEncoder(viewFileLines/getViewLines(), setViewerOffset);

    return;
  }

  viewerOffset = offset;
  drawFileContent();
}
```

09c_Click.ino

WireBender2\09_Viewer_P3\09c_Click.ino

```
int8_t clickViewer()
{
  int8_t clickIndex = clickToolbar();
  if(clickIndex >= 0)
    return clickIndex;

  if( viewerSettings.b8 )
  {
    clickIndex = clickLineButtons();
    if(clickIndex >= 0)
      return clickIndex;
  }

  return -1;
}
```

09d_Draw.ino

WireBender2\09_Viewer_P3\09d_Draw.ino

```
bool drawFileView()
{
  SdFile file;
  if(!openFile(&file, currentPath))
  {
    initBrowser();
    return false;
  }

  t.setTextColor(WHITE, BLUE);
  t.setTextArea(46, 0, 274, 24);
  t.clearTextArea();
  t.printAlignedOffseted(shortName(currentPath), gTextAlignMiddleLeft, 35, 0);
  bmpDraw(F("GFX/Browser/File_Blue.565"), 54, 1);

  if( viewerSettings.b8 ) //isPlayer
    t.fillCircle(306, 12, 7, MAGENTA);

  viewFileLines = countLines(&file);

  if(selectedLine >= viewFileLines)
    selectedLine = viewFileLines - 1;

  if(viewerSettings.b8)
  viewerOffset = selectedLine / getViewLines();
```

```
4175    //Debug(String(F("Datei mit ")+String(viewFileLines)+String(F(" Zeilen wird gelesen.")));
4176    file.close();
4177    return drawFileContent();
4178  }
4179
4180  bool drawFileContent()
4181  {
4182    SdFile file;
4183    if(!openFile(&file, currentPath))
4184      return false;
4185
4186    if( viewerSettings.b8 )
4187      terminateLineButtons();
4188    else
4189      clearLineButtonRect();
4190
4191
4192    uint16_t start = getViewLines() * viewerOffset;
4193    uint32_t startseek, seek = getLineSeek(&file, start);
4194    startseek = seek;
4195    for(uint16_t i = start; i < (uint8_t)(getViewLines() * (viewerOffset + 1)); i++)
4196    {
4197      if(i >= viewFileLines || (seek < 1 && start > 0))
4198        break;
4199
4200      if( viewerSettings.b8 )
4201        lineButtons[i % VIEWLINES] = Line(i, clickPlayerLineBtn, drawPlayerLineBtn, 1, seek);
4202
4203      seek = getNextSeek(&file, seek);
4204    }
4205
4206    if(!viewerSettings.b8)
4207    {
4208      seek = getNextSeek(&file, seek);
4209
4210      uint16_t len = seek-startseek;
4211      char buf[len+1];
4212      file.seekSet(startseek);
4213      file.read(buf, len);
4214      buf[len] = 0;
4215      t.setTextColor(BLACK, WHITE);
4216      t.setTextArea(50, 29, 270, 210);
4217      t.printAligned(String(buf), gTextAlignTopLeft);
4218    }
4219    file.close();
4220
4221    viewerSettings.b1 = viewerSettings.b2 = viewerOffset > 0;
4222    viewerSettings.b3 = viewerSettings.b4 = viewFileLines > (uint8_t) (getViewLines() * (viewer-
Offset + 1));
4223    resetViewerToolbar();
4224
4225    if(  viewerSettings.b8 )
4226      resetEncoder(selectedLine, selectLine);
4227    else
4228      resetEncoder(viewerOffset, setViewerOffset);
4229    return true;
4230  }
```

09e_Init.ino

WireBender2\09_Viewer_P3\09e_Init.ino

```
4231  void initViewer()
4232  {
4233    terminatePreviousView();
4234
4235    clickCallback = clickViewer;
4236    resetCallback = resetViewer;
4237    errorCallback = openViewer;
4238
4239    t.fillScreen(WHITE);
4240
4241    if(String(currentPath).length() < 1 || !sd.exists(currentPath))
4242      return initBrowser();
4243
4244    viewerSettings.b8 = String(currentPath) == String(workingPath);
4245    viewerSettings.b5 = viewerSettings.b8; // Toolbar #2
4246
4247    initViewerToolbar();
4248
4249    if(!drawFileView())
4250      return;
4251  }
4252
4253  void openViewer(ButtonInfo* i)
4254  {
4255    initViewer();
4256  }
4257
4258  int8_t resetViewer()
4259  {
4260    resetViewerToolbar();
4261    return 1;
4262  }
```

10a_Helpers.ino

WireBender2\10_Settings_P4\10a_Helpers.ino

```
4263  bool SaveSetting(String name, int32_t value)
4264  {
4265    String path = SETTINGS_PATH;
4266    if(path.length() < 1)
4267      return false;
4268
4269    SdFile file;
4270    if(!openFile(&file, path.c_str(), true))
4271      return false;
4272
4273    uint16_t i = 0;
4274    for(; i < countLines(&file); i++)
4275    {
4276      uint32_t seek = getLineSeek(&file, i);
4277      String line = readLine(&file, seek);
4278      if( line.startsWith(name) )
4279        break;
4280    }
4281
4282    if(i <= 3)
4283      return false;
4284
4285    selectedSetting = i - 3;
4286    settingBuffer = value;
4287
4288    uint8_t success = SaveFromBuffer();
4289    if(success == 5)
4290      Debug(F("SaveSetting: "), name.c_str(), String(value).c_str() );
4291
4292    return success == 5 || success == 2;
4293  }
4294
4295  uint8_t SaveFromBuffer()
4296  {
4297    String path = SETTINGS_PATH;
4298    if(path.length() < 1)
4299      return 0;
4300
4301    String bkpPath = String(F("/BKP_"))+path.substring(1);
4302    if(sd.exists(bkpPath.c_str()))
4303    sd.remove(bkpPath.c_str());
4304
4305    SdFile file;
4306    if(!openFile(&file, path.c_str(), true))
4307      return 0;
4308
4309    SdFile buffer;
4310    if(!openFile(&buffer, bkpPath.c_str(), true))
4311      return 0;
4312
4313    uint32_t seek = getLineSeek(&file, selectedSetting + 3);
4314    String line = readLine(&file, seek);
4315    Setting set = getSettingNameValue(line);
4316
4317    if( !set.isValid || !set.isNumber )
4318    {
4319      file.close();
4320      buffer.close();
4321      sd.remove(bkpPath.c_str());
4322      return 1;
4323    }
4324
4325    String name = String(set.name);
4326
4327    if(
4328    (set.isFloat && abs((float)settingBuffer / 100.0f - set.fVal) < 0.01) ||
4329    (!set.isFloat && settingBuffer == set.iVal)
4330    )
4331    {
4332      file.close();
4333      buffer.close();
4334      sd.remove(bkpPath.c_str());
4335      Debug(name+String(F(" not saved, because value didn't change.")));
4336      return 2;
4337    }
4338
4339    if( set.isFloat )
4340    {
4341      set.fVal = (float)settingBuffer / 100.0f;
4342      line = name+String(SEPARATOR)+String(set.fVal);
4343    }
4344    else
4345    {
4346      set.iVal = settingBuffer;
4347      line = name+String(SEPARATOR)+String(set.iVal);
4348    }
4349
4350    file.seekSet(0);
4351
4352    uint32_t written = 0;
4353    while(written < seek)
4354    {
4355      uint8_t len = min(seek - written, 255);
4356      char fBuf[len + 1];
4357      file.read(fBuf, len);
4358      fBuf[len] = 0;
4359      String result = String(fBuf);
4360      result.trim();
4361      buffer.print(result);
4362      written += len;
4363    }
4364
4365    buffer.println(EMPTY);
4366    buffer.println(line);
4367    uint32_t next = getNextSeek(&file, seek + 1);
4368
4369    written = next;
4370    while(written < file.fileSize())
4371    {
4372      uint8_t len = min(file.fileSize() - written, 255);
4373      char fBuf[len + 1];
4374      file.read(fBuf, len);
4375      fBuf[len] = 0;
4376      String result = String(fBuf);
4377      result.trim();
4378      buffer.print(result);
4379      written += len;
4380    }
4381
4382    file.close();
4383    buffer.close();
4384
4385    if(!sd.remove(path.c_str()))
4386      return 3;
4387
4388    if(!sd.rename(bkpPath.c_str(), path.c_str()))
4389      return 4;
4390
4391    doSetting(set);
4392
4393    if(name == F("Material") || name == F("Delta Z"))
4394      return 5;
4395    else if(Post(String(F("set:"))+line))
4396      return 5;
4397
4398    ErrorDialog(
4399      String(eText(lastError))+String(F("#-#"))+
4400      String(F("The setting:#"))+
4401      line+
4402      String(F(" couldn't be sent to the driver module."))
4403    );
4404    return 0;
4405  }
```


10ab_Helpers.ino

WireBender2\10_Settings_P4\10a_Settings\10ab_Helpers.ino

```
4406  void setSettingsOffset(uint8_t offset)
4407  {
4408    if( offset < 0 )
4409      return;
4410
4411    viewerOffset = offset;
4412    drawSettings();
4413  }
4414
4415  uint8_t getSettingsMaxPage()
4416  {
4417    uint8_t div = (uint8_t) (viewFileLines - 3) / SETTINGSLINES;
4418    return ( viewFileLines -3 ) % SETTINGSLINES > 0 ? div + 1 : div;
4419  }
```

10ac_Click.ino

WireBender2\10_Settings_P4\10a_Settings\10ac_Click.ino

```
4420  int8_t clickSettings()
4421  {
4422    int8_t clickIndex = clickToolbar();
4423    if(clickIndex >= 0)
4424      return clickIndex;
4425
4426    clickIndex = clickLineButtons();
4427    if(clickIndex >= 0)
4428      return clickIndex;
4429
4430    return -1;
4431  }
4432
4433  void clickSettingBtn(LineInfo* i)
4434  {
4435    if(!i->interpreted || !i->isValid || i->isComment)
4436    {
4437      toolbar[1].enable(false);
4438      selectedSetting = 255;
4439      refreshSettingsLines();
4440      t.fillTriangle(155, 190, 165, 190, 160, 200, WHITE);
4441      return;
4442    }
4443
4444    selectedSetting = i->lineNumber;
4445    refreshSettingsLines();
4446  }
4447
4448  void clickNextPageBtn(ButtonInfo* i)
4449  {
4450    uint8_t max = getSettingsMaxPage();
4451    if(viewerOffset + 1 < max)
4452      setSettingsOffset(viewerOffset + 1);
4453    else
4454      setSettingsOffset(0);
4455  }
```

10ad_Draw.ino

WireBender2\10_Settings_P4\10a_Settings\10ad_Draw.ino

```
4456  void drawSettingBtn(LineInfo* i)
4457  {
4458    String path = SETTINGS_PATH;
4459    if(path.length() < 1)
4460      return;
4461
4462    SdFile file;
4463    if(!openFile(&file, path.c_str(), true))
4464      return;
4465
4466    String text = readLine(&file, i->seek);
4467    file.close();
4468
4469    uint8_t y = i->lineNumber % SETTINGSLINES * 13 + 50;
4470
4471    if(selectedSetting == i->lineNumber)
4472    {
4473      toolbar[1].enable(true);
4474      t.fillTriangle(155, 190, 165, 190, 160, 200, BLUE);
4475      i->active = true;
4476    }
4477
4478    uint16_t fontColor = BLACK;
4479    uint16_t backColor = i->active ? BLUE : WHITE;
4480    t.setTextArea(4, y, 312, 13);
4481
4482    Setting set = getSettingNameValue(text);
4483    i->interpreted = true;
4484    i->isValid = set.isValid;
4485    i->isComment = !set.isNumber;
4486
4487    if(!i->isValid)
4488    {
4489      fontColor = RED;
4490
4491      t.setTextColor(fontColor, backColor);
4492      t.clearTextArea();
4493      t.printAligned(text, gTextAlignMiddleCenter);
4494    }
4495    else
4496    {
4497      String name = String(set.name);
4498      String val = set.isFloat ? String(set.fVal) : String(set.iVal);
4499
4500      if(i->isComment)
4501        fontColor = GRAY;
4502
4503      t.setTextColor(fontColor, backColor);
4504      t.clearTextArea();
4505
4506      t.printAlignedOffseted(name, gTextAlignMiddleRight, -170, 0);
4507
4508      if(i->isComment)
4509        t.printAlignedOffseted(text.substring(text.indexOf(SEPARATOR)+1), gTextAlignMiddle-
Left, 170, 0);
4510      else
4511        t.printAlignedOffseted(val, gTextAlignMiddleLeft, 170, 0);
4512    }
4513
4514  backColor = WHITE;
4515    if(selectedSetting != 255)
4516  {
4517    bool samePage = i->lineNumber / SETTINGSLINES == selectedSetting / SETTINGSLINES;
4518      backColor = (i->lineNumber >= selectedSetting && samePage) ? BLUE : WHITE;
4519    }
4520
4521    if(!i->isValid && backColor == WHITE)
4522      return;
4523
4524    t.drawFastVLine(159, y, 195 - y, backColor);
4525    t.drawFastVLine(160, y, 195 - y, backColor);
4526    t.drawFastVLine(161, y, 195 - y, backColor);
4527  }
4528
4529  void drawPageBtnOverlay()
4530  {
4531    uint16_t c = toolbar[0]._info.enabled ? BLUE : GRAY;
4532    t.setTextArea( 0, 200, 106, 40 );
4533    t.setTextColor( WHITE, c );
4534    t.printAligned(String(viewerOffset + 1)+String(F(" of "))+String(getSettingsMaxPage()),
gTextAlignMiddleCenter);
4535  }
4536
4537  void drawSettings()
4538  {
4539    String path = SETTINGS_PATH;
4540    if(path.length() < 1)
4541      return;
4542
4543    SdFile file;
4544    if(!openFile(&file, path.c_str(), true))
4545      return;
4546
4547    clearSettingsLines();
4548
4549    uint16_t start = SETTINGSLINES * viewerOffset;
4550    uint32_t seek = getLineSeek(&file, start + 3);
4551    for(uint16_t i = start; i < (uint8_t)(SETTINGSLINES * (viewerOffset + 1)); i++)
4552    {
4553      if(i >= viewFileLines || (seek < 1 && start > 0))
4554        break;
4555
4556      lineButtons[i % SETTINGSLINES] = Line(i, clickSettingBtn, drawSettingBtn, 2, seek);
4557      seek = getNextSeek(&file, seek);
4558    }
4559    file.close();
4560  }
4561
4562  void clearSettingsLines()
4563  {
4564    toolbar[1].enable(false);
4565    clearRect(0,45,320,155);
4566
4567    for(int i = 0; i < SETTINGSLINES; i++)
4568      lineButtons[i] = Line();
4569  }
4570
4571  void refreshSettingsLines()
4572  {
4573    if(
4574      selectedSetting < (uint16_t) viewerOffset * SETTINGSLINES ||
4575      selectedSetting >= (uint16_t) ( viewerOffset + 1 ) * SETTINGSLINES
4576    )
4577      return;
4578
4579    for(uint8_t i = 0; i < SETTINGSLINES; i++)
4580    {
4581      if(  i != selectedSetting % SETTINGSLINES && lineButtons[i]._info.active )
4582        lineButtons[i].draw(false);
4583    }
4584  }
```

10ae_Init.ino

WireBender2\10_Settings_P4\10a_Settings\10ae_Init.ino

```
4585  void initSettings(ButtonInfo* i)
4586  {
4587    initSettings();
4588  }
4589
4590  void initSettings()
4591  {
4592    terminatePreviousView();
4593
4594    Dialog( F("_Settings"), EMPTY, initConfiguration );
4595    dialogBtn2(F("_Edit"), initSetting);
4596    toolbar[1].enable(false);
4597
4598    clickCallback = clickSettings;
4599    resetCallback = resetSettings;
4600    errorCallback = initSettings;
4601
4602    selectedSetting = 255;
4603
4604    String path = SETTINGS_PATH;
4605    if(path.length() < 1)
4606      return;
4607
4608    SdFile file;
4609    if(!openFile(&file, path.c_str()))
4610      return;
4611
4612    viewFileLines = countLines(&file);
4613    file.close();
4614
4615    setSettingsOffset(0);
4616
4617    if(viewFileLines - 3 > SETTINGSLINES)
4618    {
4619      dialogBtn1( SPACE, clickNextPageBtn );
4620      drawPageBtnOverlay();
4621    }
4622  }
4623
4624  int8_t resetSettings()
```

```
{
  resetToolbar();
  resetLineButtons();
  drawPageBtnOverlay();
  return 1;
}
```

10bc_Click.ino

WireBender2\10_Settings_P4\10b_Setting\10bc_Click.ino

```
void clickSaveBtn(ButtonInfo* i)
{
  switch(SaveFromBuffer())
  {
    case 0:
      return;
    case 1:
      ErrorDialog( String(F("Invalid setting found in /Settings.s")) );
    case 2:
      return initSettings(i);
    case 3:
      return ErrorDialog(F("The old settings couldn't be deleted."));
    case 4:
      return ErrorDialog(F("The settings couldn't be copied from the buffer."));
    case 5:
      return  initSettings(i);
  }
}

void clickIncrementBtn(ButtonInfo* i)
{
  if(getEncoderIncrement() > 999)
    setEncoderIncrement();
  else
    setEncoderIncrement(getEncoderIncrement() * 10);
}
```

10bd_Draw.ino

WireBender2\10_Settings_P4\10b_Setting\10bd_Draw.ino

```
void drawSettingValue(int32_t val)
{
  settingBuffer = val;
  if(encoderSettings.b6)
    return drawSettingValue((float)val / 100.0f);

  t.setFont(Calibri24);
  t.setTextColor(BLACK, WHITE);
  t.setTextArea( 90, 107, 140, 24 );
  t.clearTextArea();
  t.printAligned( String(val), gTextAlignMiddleCenter );
  t.setFont(Calibri13);
}

void drawSettingValue(float val)
{
  t.setFont(Calibri24);
  t.setTextColor(BLACK, WHITE);
  t.setTextArea( 90, 107, 140, 24 );
  t.clearTextArea();
  t.printAligned( String(val), gTextAlignMiddleCenter );
  t.setFont(Calibri13);
}

void drawIncrementBtnOverlay()
{
  t.setTextArea( 107, 200, 106, 40 );
  t.setTextColor( WHITE, BLUE );
  t.clearTextArea();
  if(encoderSettings.b6)
  {
    float increment = 0.01f;
    if(getEncoderIncrement() != 1)
      increment = getEncoderIncrement() / 10.0f;

    t.printAligned(String(F("x ="))+String(increment), gTextAlignMiddleCenter);
  }
  else
    t.printAligned(String(F("x ="))+String(getEncoderIncrement()), gTextAlignMiddleCenter);
}
```

10be_Init.ino

WireBender2\10_Settings_P4\10b_Setting\10be_Init.ino

```
void initSetting(ButtonInfo* i)
{
  terminatePreviousView();

  String path = SETTINGS_PATH;
  if(path.length() < 1)
    return;

  SdFile file;
  if(!openFile(&file, path.c_str()))
    return;

  String text = readLine(&file, getLineSeek(&file, selectedSetting + 3));
  file.close();

  if(text.length() < 1)
    return ErrorDialog(F("Empty line in Settings file."));

  Setting set = getSettingNameValue(text);

  if(set.isNumber)
  {
    Dialog( set.name, EMPTY, NULL );
    encoderSettings.b6 = set.isFloat;
    int32_t val = set.isFloat ? int32_t(set.fVal * 100) : set.iVal;
    drawSettingValue(val);
    dialogBtn3(F("Cancel"), initSettings );
    dialogBtn2(SPACE, clickIncrementBtn);
    drawIncrementBtnOverlay();
    dialogBtn1(F("Save"), clickSaveBtn );
    resetEncoder(val, drawSettingValue);

    resetCallback = resetSetting;
    errorCallback = initSettings;
  }
  else
    Warning(String(F("Setting invalid: "))+text);
}

int8_t resetSetting()
{
  resetDialog();
  drawIncrementBtnOverlay();
return 1;
}
```

10c_Click.ino

WireBender2\10_Settings_P4\10c_Click.ino

```
void clickKalibrationBtn(ButtonInfo* i)
{
  Calibrate();
}

void clickDefaultsBtn(ButtonInfo* i)
{
  ConfirmDialog(                    F("Default Settings"),
        F("Are you sure, to reset all settings to default?*You will need to recalibrate afterwards."),
        defaultSettings,
        initConfiguration
  );
}
```

10d_Defaults.ino

WireBender2\10_Settings_P4\10d_Defaults.ino

```
void defaultSettings(ButtonInfo* i)
{
  String path = SETTINGS_PATH;
  if(path.length() < 1)
    return;

  SdFile file;
  if(!openFile(&file, path.c_str(), true))
    return ErrorDialog(F("The settings file couldn't be opened."));

  if(!file.remove())
  {
    file.close();
    return ErrorDialog(F("The old settings file couldn't be deleted."));
  }

  if(!openFile(&file, path.c_str(), true))
    return ErrorDialog(F("The new settings file couldn't be created."));

  int len = strlen_P(default_settings);
  char def[len+1];

  strcpy_P(def, default_settings);
  def[len] = 0;

  String c1 = F("/*************************/"), c2 = F("/*******  SETTINGS  ********/");

  file.seekSet(0);
  file.println(c1);
  file.println(c2);
  file.println(c1);
  file.print(String(def));
  file.close();
  Log(String(len)+F(" default settings restored."));
  initSettings();
}
```

10e_Init.ino

WireBender2\10_Settings_P4\10e_Init.ino

```
void initConfiguration(ButtonInfo* i)
{
  terminatePreviousView(false);

  Dialog( F("Configuration"), EMPTY, openJogging );
  toolbar[3] = Button(78, 53, 165, 40, F("Calibrate"), 1, 1, clickKalibrationBtn);
  toolbar[4] = Button(78, 101, 165, 40, F("Reset Defaults"), 1, 1, clickDefaultsBtn);
  toolbar[5] = Button(78, 148, 165, 40, F("Settings"), 1, 1, initSettings);

  clickCallback = clickToolbar;
  resetCallback = resetToolbar;
  errorCallback = initConfiguration;
}
```

10f_LoadSettings.ino

WireBender2\10_Settings_P4\10f_LoadSettings.ino

```
void reloadSettings(ButtonInfo* i)
{
  Post(F("restart"));
}

uint8_t loadSettings()
{
  const uint8_t sCount = 14;

  String path = SETTINGS_PATH;
  if(path.length() < 1)
    return 0;

  SdFile file;
  if(!openFile(&file, path.c_str()))
    return 0;

  uint32_t lastSeek = 0;
  uint32_t nextSeek = getNextSeek(&file, lastSeek);
  uint8_t settingCount = 0;
  if(nextSeek == 0)
  {
    file.close();
    return 0;
  }

  t.fillRect(31, 180, 259, 10, GRAY);

  while(nextSeek > lastSeek)
  {
    String line = readLine(&file, lastSeek, nextSeek);
    line.trim();

    lastSeek = nextSeek;
    nextSeek = getNextSeek(&file, lastSeek);
    //Debug(String(lastSeek)+" bis "+String(nextSeek));

    Setting set = getSettingNameValue(line);
    String name = String(set.name);
    if(!set.isNumber)
    {
      if(name.startsWith(F("Program")) && line.length() > name.length()+1)
      {
        String program = line.substring(name.length()+1);
        setWorkingPath(program);
      }

      continue;
    }

    doSetting(set);
    if(notSend(set))
      continue;

    if(Post(String(F("set:"))+line))
      settingCount++;

    delay(200);
    t.fillRect(31, 180, map(settingCount, 0, sCount, 0, 259), 10, BLUE);
  }
  Debug("File size:"+String(file.fileSize()));

  Debug(F("Settings SENT: "), String(settingCount).c_str());
  file.close();
  return settingCount == sCount ? 2 : 1;
}

Setting getSettingNameValue(String text)
{
  Setting result = Setting();
  uint8_t separator = text.indexOf("=");
  if(separator < 0 || separator >= 30)
    return result;

  result.isValid = true;

  String value = text.substring(separator + 1, text.length());
  strcpy(result.name, text.substring(0, separator).c_str());
  if(!isFloat(value) && !isInt(value))
    return result;

  result.isNumber = true;
  result.isFloat = !isInt(value);
  if(result.isFloat)
    result.fVal = value.toFloat();
  else
    result.iVal = value.toInt();

  return result;
}

bool notSend(Setting set)
{
  if(  String(set.name) == F("Program") ||
    String(set.name) == F("Selected Line") ||
    String(set.name) == F("Material") ||
    String(set.name) == F("Delta Z"))
      return true;

  return false;
}

bool doSetting(Setting set)
{
  if(!set.isNumber)
    return false;

String name = set.name;
  if(name == F("Selected Line"))
    selectedLine = set.iVal;
  else if(name == F("Debug"))
    config.debug = set.iVal != 0;
  else if(name == F("X Pos"))
  {
    if(xAxis.isInitialized())
      xAxis.setPos(set.iVal);
    else
      xPos = set.iVal;
  }
  else if(name == F("Y Pos"))
  {
    if(yAxis.isInitialized())
      yAxis.setPos(set.iVal);
    else
      yPos = set.iVal;
  }
  else if(name == F("Delta Z"))
  {
    deltaZ = set.fVal;
  }

  return true;
}
```

99a_Init.ino

WireBender2\99a_Init.ino

```
void setup()
{
  // TFT
  pinMode(LITE, OUTPUT);
  digitalWrite(LITE, LOW);

  // INIT SERIAL
  Serial.begin(19200);
  Log(String(F("Init WireBender "))+String(VERSION));
  initSPI();
  initEncoder();

  ramInfo(true);
}
```

99b_Loop.ino

WireBender2\99b_Loop.ino

```
void loop() {
  spiLooper();                                  //02bf_Looper
  playerLooper();                               //04f_Player
  encoderLooper();              //00a_Encoder
  serialLooper();                               //03a_Serial
  touchLooper();                                //05a_Touch
  screensaver();                                //02b_Screensaver
  triggerLooper();
}
```

Application.ino

WireBender2\Application.ino

```
/*********************************************/
/*        I N C L U D E S          */
/*********************************************/
#include <SPI.h>
#include <SdFat.h>
#include <SdFatUtil.h>
#include <LI9341_due.h>
#include <LI9341_due_config.h>
#include <TouchScreen.h>
#include <AccelStepper.h>
#include „Defines.h"
#include „Fonts/Calibri13.h"
#include „Fonts/Calibri24.h"


/*********************************************/
/*        T Y P E D E F          */
/*********************************************/
typedef int8_t (*Callback) (void);
typedef void (*ReceiveCallback) (String text);
typedef void (*EncoderCallback) (int32_t Value);
typedef void (*ButtonCallback) (ButtonInfo*);
typedef void (*LineCallback) (LineInfo*);
typedef struct boolMap {
  bool
    b1 : 1,
    b2 : 1,
    b3 : 1,
    b4 : 1,
    b5 : 1,
    b6 : 1,
    b7 : 1,
    b8 : 1;
} boolMap;
typedef struct ConfigMap {
  bool
    screenActive : 1,
    booted : 1,
    touched : 1,
    programRunning : 1,
    debug : 1,
    debugCache : 1,
    processIt : 1,
    requestIt : 1;
} ConfigMap;
//typedef struct Point2d{...};
  //Defines.h
//typedef struct axisData{...};
  //Defines.h
//typedef struct ButtonInfo{...};                                   //Defines.h


/*********************************************/
/*        G L O B A L S          */
/*********************************************/
static Callback terminateCallback = NULL;
static Callback resetCallback = NULL;
static Callback clickCallback = NULL;
static ReceiveCallback receiveCallback = NULL;
static ButtonCallback errorCallback = NULL;

volatile ConfigMap config = {
  true,  //b1   screenActive
  false,   //b2booted
  false,   //b3touched
  false,   //b4programRunning
  true,  //b5   debug
  false,   //b6debugCache
  false,   //b7processIt
  false  //b8   requestIt
};
volatile boolMap encoderSettings = {
  true,  //b1   firstEncoder
  false,   //b2btnState
  false,   //b310
  false,   //b4100
  false,   //b51000
  false,   //b6isFloat
  false,   //b7CANCEL_CLICK
  false  //b8   INIT_SCREEN
```

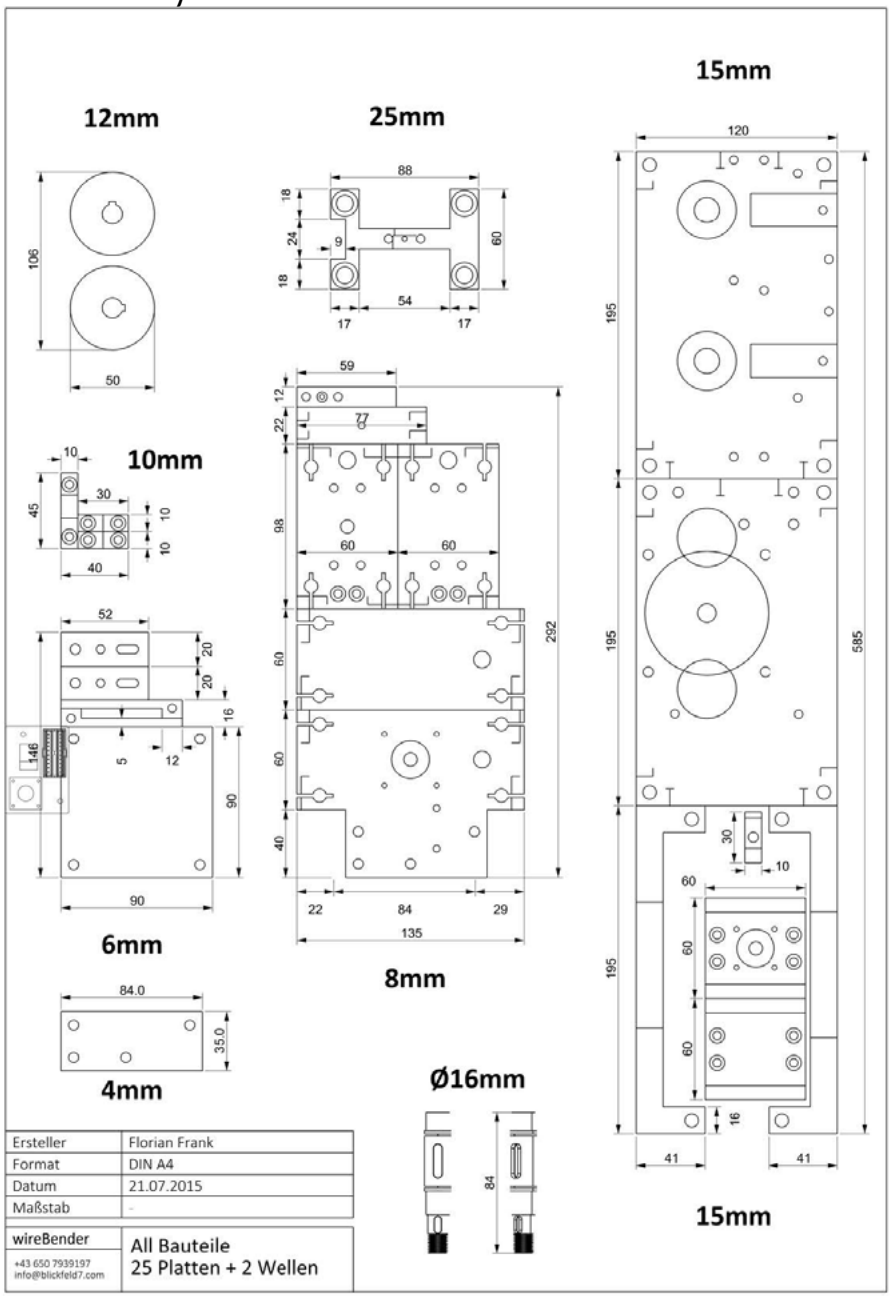
```
};

static uint32_t lastAction = 0;
static uint32_t lastTrigger = 0;
static uint32_t lastWait = 0;
static uint16_t waitTime = 0;

// Touch
static uint32_t lastClick;

// SD
static SdFat sd;
                    //591bytes

// TFT
static ILI9341_due t = ILI9341_due(TFT_CS, TFT_DC);
  //94bytes
  // For better pressure precision, we need to know the resistance
  // between X+ and X- Use any multimeter to read it
  // For the one we're using, its 300 ohms across the X plate
static TouchScreen ts = TouchScreen(XP, YP, XM, YM, 0);
  //6bytes
static Point2d _tPoint;

// Encoder
static EncoderCallback encoderCallback = NULL;
static Callback encoderBtnCallback = NULL;

volatile int32_t Encoder = 0;
static int32_t lastEncoder = 0;

volatile int32_t lastEncoded = 0;
static int32_t lastEncoderValue = 0; //The actual value

// SPI
char buf[SPI_BUFFER];
volatile uint8_t dataLength;
volatile uint8_t pos;
volatile uint8_t lastError = 0;
SPISettings _spiSettings = SPISettings(F_CPU / 8, MSBFIRST, SPI_MODE0);

// TFT
//static Button toolbar[5 + 1];
        //78bytes
//static Line lineButtons[VIEWLINES];
      //176bytes

// Jogging P1
static int16_t xPos, yPos;
static float deltaZ = 15.0f;
//Axis xAxis;
//Axis yAxis;
//Axis zAxis;

// Browser P2
static uint8_t browserOffset = 0;
static boolMap browserSettings;
static char currentPath[255];
        //255bytes

//  Viewer P3
static uint8_t viewerOffset = 0;
static uint16_t viewFileLines = 0;
static boolMap viewerSettings;

// Player P4
static char workingPath[255];
        //255bytes
static uint16_t selectedLine = 0;
static uint16_t lastAck = 0;
static uint32_t lastSentTime = 0;
static uint32_t playerSeek = 0;

// Settings P5
static uint8_t selectedSetting = 255;
static int32_t settingBuffer = 0;
```

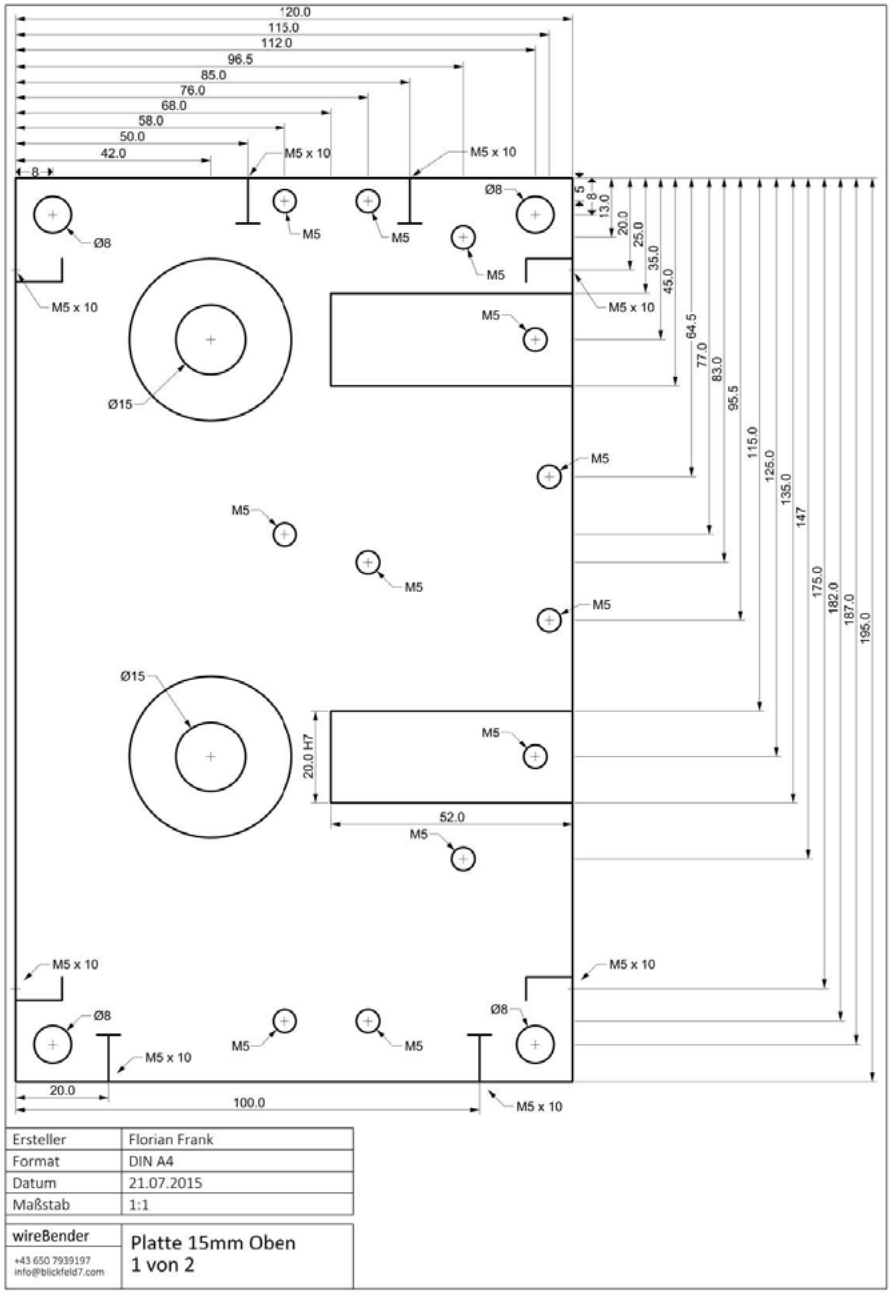
Defines.h

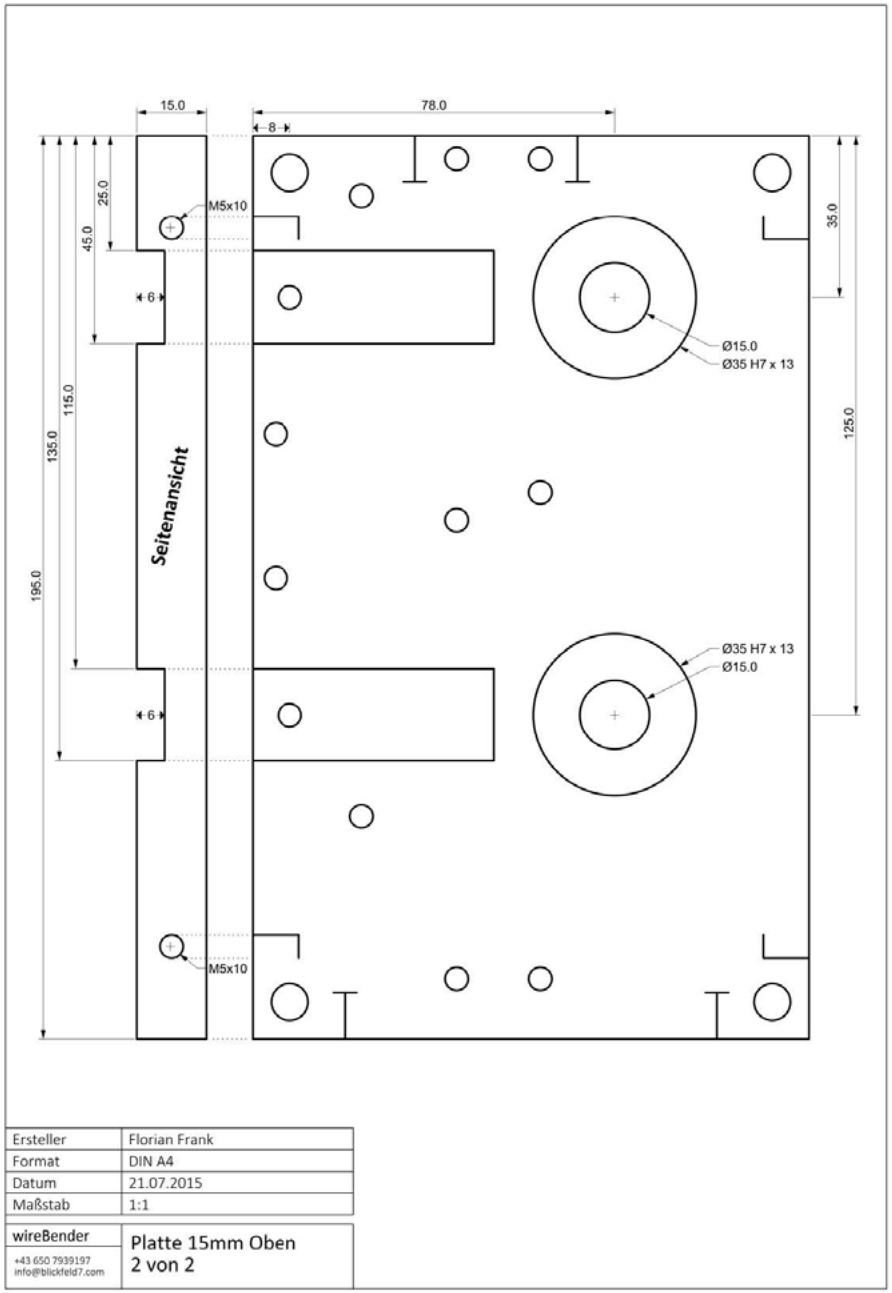
WireBender2\Defines.h

```
/*
 * Defines.h
 *
 * Created: 14.12.2015 07:59:22
 *  Author: Florian Frank
 */
#include <Arduino.h>
#define VERSION 2.41


/************************************************/
/*            PINS                              */
/************************************************/

// SPI
#define MS 18
#define CS 9

// SD
#define SD_CS 47  // Chip Select for SD card

// TFT
#define TFT_DC 48
#define TFT_CS 53
#define LITE 46

// Touchscreen
#define YP A15
#define XM A14
#define YM 49
#define XP 48

// Encoder
#define E1 3  //interrupt 0
#define E2 2  //interrupt 1
#define BTN 4 //push button

// Triggers
#define TRIGGER1 30
#define TRIGGER2 31
#define TRIGGER_TIME 200

/************************************************/
/*            SD                                */
/************************************************/
#define SD_SPI_SPEED SPI_HALF_SPEED // SD card SPI speed, try SPI_FULL_SPEED or SPI_HALF_SPEED
#define BUFFPIXELCOUNT 100  // size of the buffer in pixels



/************************************************/
/*            SPI                               */
/************************************************/
#define SPI_BUFFER 256
#define TIMEOUT 2000
#define WAIT 20
#define WAITLONG 150
#define MAX_SENDS 10

#define RESET 0
#define SOH 1
#define STX 2
#define EOT 4
#define ENQ 5
#define ACK 6
#define NAK 21
#define IDLE 22



/************************************************/
/*            TFT                               */
/************************************************/
#define WHITE 0xFFFF
#define BLACK 0x0000
#define GRAY 0xDEFB

#define BLUE 0x51A
#define MAGENTA 0xF8BB
#define RED 0xF800

#define BROWSELINES 9
#define VIEWLINES 16
#define SETTINGSLINES 11
#define LINEBUTTON F("LineButton")

#define ZEROING 4

#define SCREENSAVER 120
      //Sekunden bis der Bildschirmschoner aktiviert wird
#define NOTIMPLEMENTED F("This function is not implemented yet")


/************************************************/
/*      INTERPRETER                             */
/************************************************/
#define ZEROING 4
#define MINELAPSEDTIME 800

#define EMPTY F("")
#define SPACE F(" ")
#define DOT F(".")
#define ZERO F("0")
#define HASHTAG F("#")
#define SEPARATOR F("=")
#define SLASH F("/")
#define DBLPT F(":")
#define MM F("mm")
#define NL F("\n")

// Interpreter
#define VALID_CHARS F(" -.0123456789#:/ xyz")
#define XCHAR F("x")
#define YCHAR F("y")
#define ZCHAR F("z")


/************************************************/
/*            SETTINGS                                                  */
/************************************************/
#define XSTEPS 1269.8412698f
#define YSTEPS 629.92125984f
#define ZSTEPS 83.0025144f
#define XMAX 10222
#define YMAX 5071
#define ZMAX 83000

#define SETTINGS_PATH "/Settings.s"

const char default_settings[] PROGMEM  = {
  "Program=/teeee.w\n"
  "Selected Line=0\n"
  "Material=1\n"
  "Delta Z=15.00\n"
  "Debug=1\n"
  "X Origin=20518\n"
  "Y Origin=11383\n"
  "X Speed=2000.00\n"
  "X Max Speed=0.00\n"
  "X Acceleration=800.00\n"
  "Y Speed=2000.00\n"
  "Y Max Speed=0.00\n"
  "Y Acceleration=800.00\n"
  "Z Speed=2000.00\n"
  "Z Max Speed=0.00\n"
  "Z Acceleration=800.00\n"
  "X Pos=0\n"
  "Y Pos=0"
};


/************************************************/
/*            STRUCTS                                                   */
/************************************************/

typedef struct Point2d {
  uint16_t x;
  uint8_t y;
  uint16_t z;
} Point2d;

typedef struct axisData {
  bool
    isValid = false;
  int8_t
    axis = 0;
  int32_t
  pos = 0;
  float
    accel = 0,
    speed = 0;
} axisData;

typedef struct ButtonInfo {
  bool
    active : 1,
    enabled : 1,
    visible : 1,
    selected : 1,
    initialized : 1,
    b6 : 1,
    b7 : 1,
    b8 : 1;
  uint8_t
    y = 0,
    height = 0;
  int16_t
    x = 0,
    width = 0;
    const __FlashStringHelper*
    name;
} ButtonInfo;

typedef struct LineInfo {
  bool
    active : 1,
    sending : 1,
    isBrowserLine : 1,
    isViewerLine : 1,
    interpreted : 1,
    isComment : 1,
    isValid : 1,
    isCommand : 1;
  uint16_t
    lineNumber = 0;
  uint32_t
    seek = 0;
} LineInfo;

typedef struct Setting {
  bool
    isValid : 1,
    isNumber : 1,
    isFloat : 1;
  char
    name[30];
  int32_t
    iVal;
  float
    fVal;
} Setting;


/************************************************/
/*            LANGUAGE                                                  */
/************************************************/
#define YES F("Yes")
#define NO F("No")
#define OK F("OK")
#define CANCEL F("Cancel")
#define CLOSE F("Close")
```

7.3. Konstruktionspläne

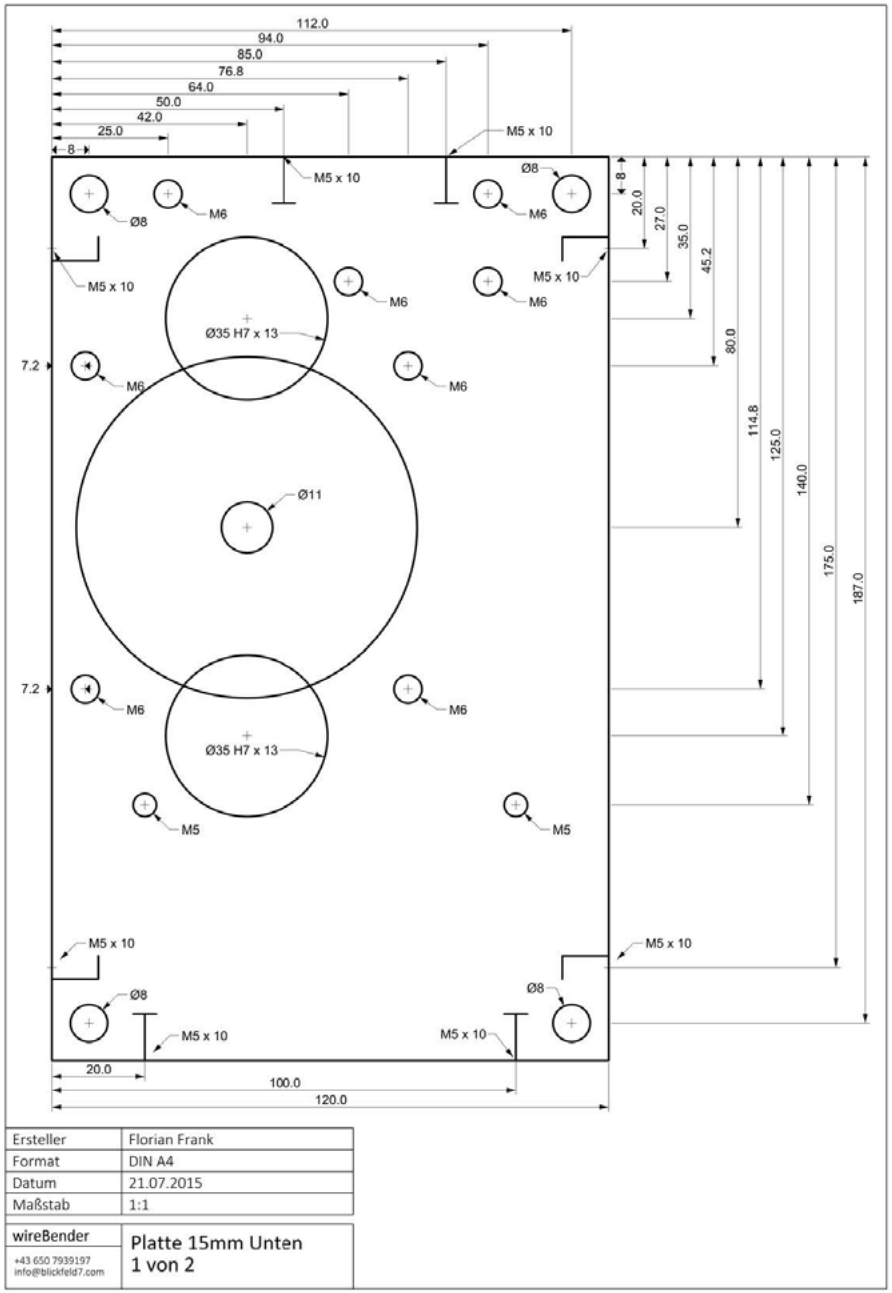






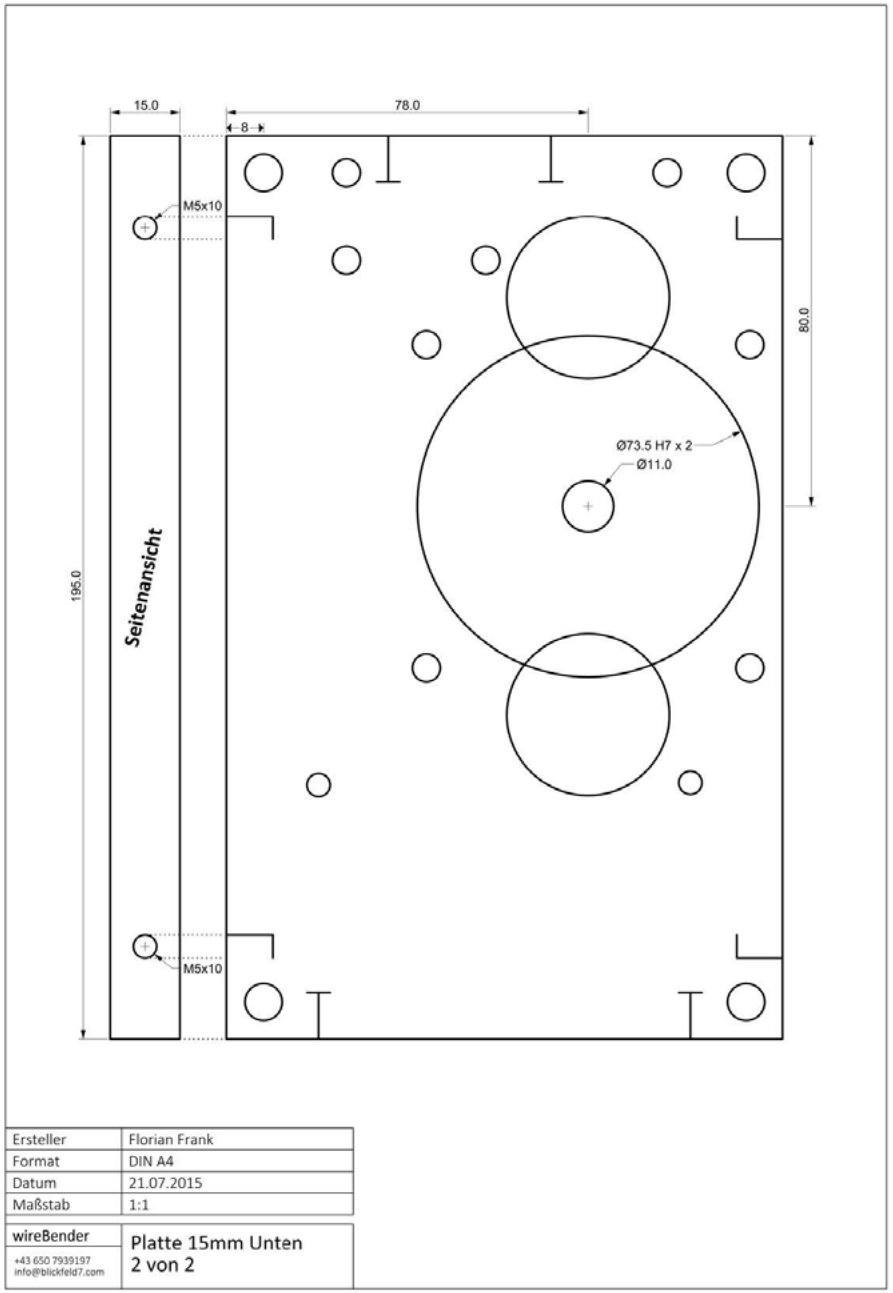
Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1

wireBender +43 650 7939197 info@blickfeld7.com	Platte 15mm Oben 2 von 2

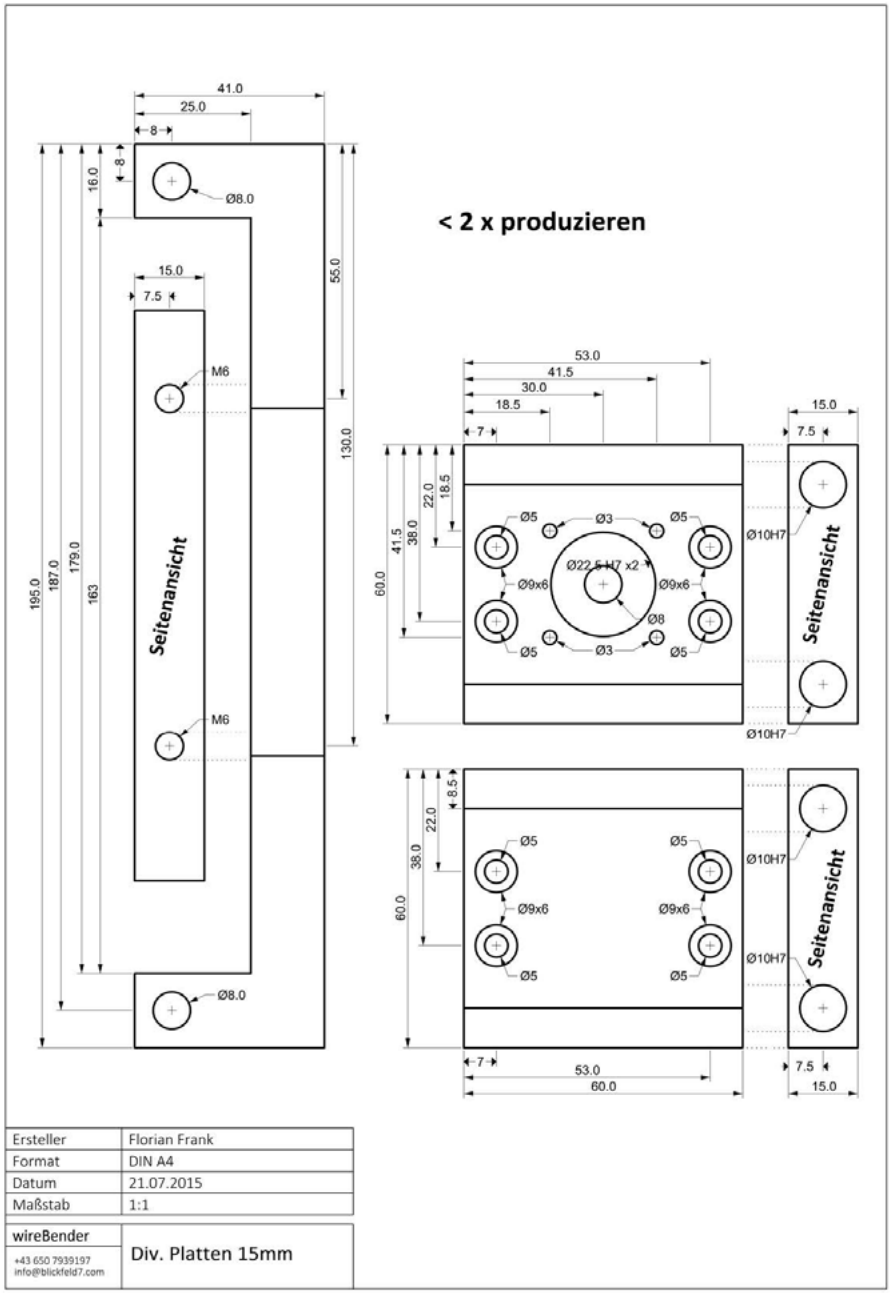


Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1

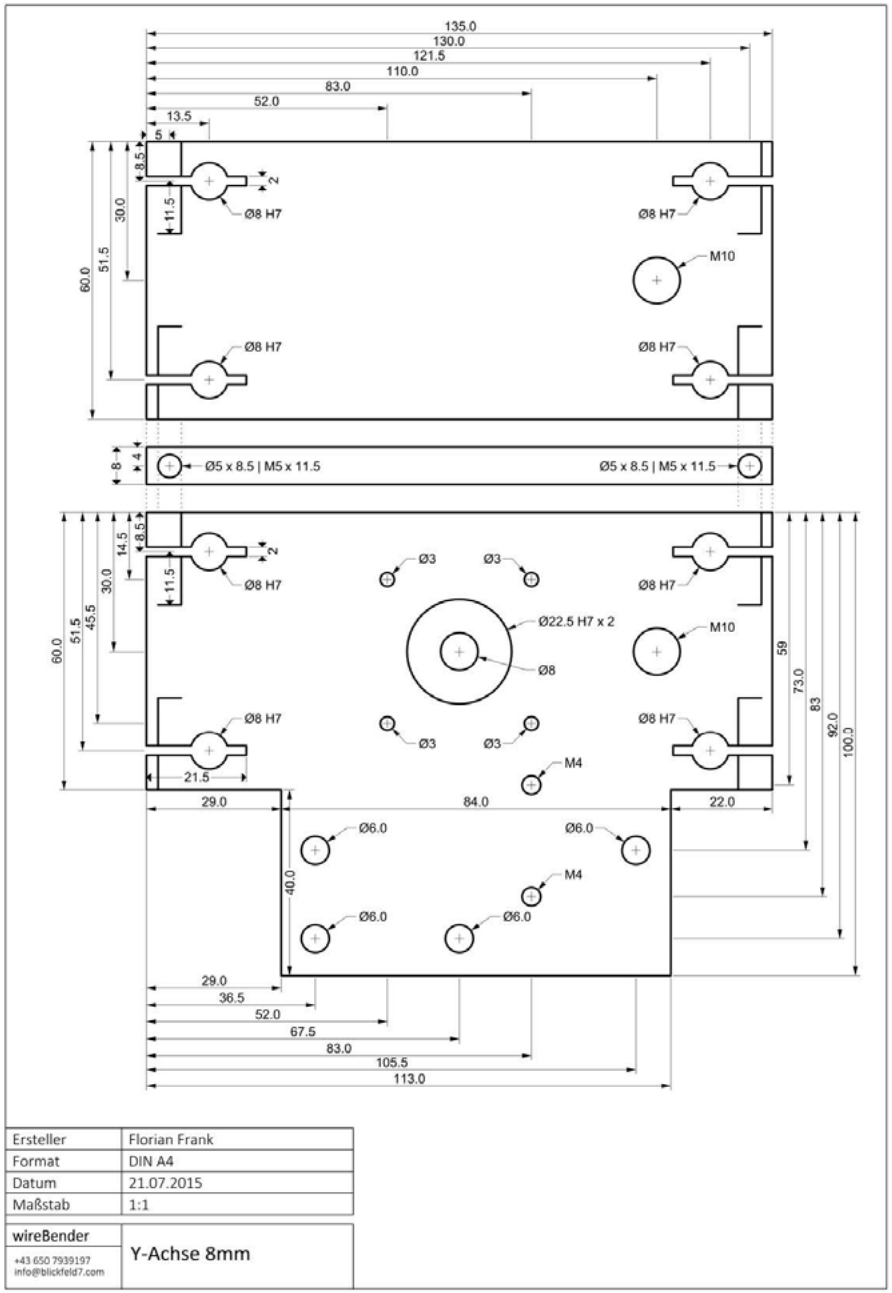
wireBender +43 650 7939197 info@blickfeld7.com	Platte 15mm Unten 1 von 2



Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1
wireBender +43 650 7939197 info@blickfeld7.com	Platte 15mm Unten 2 von 2

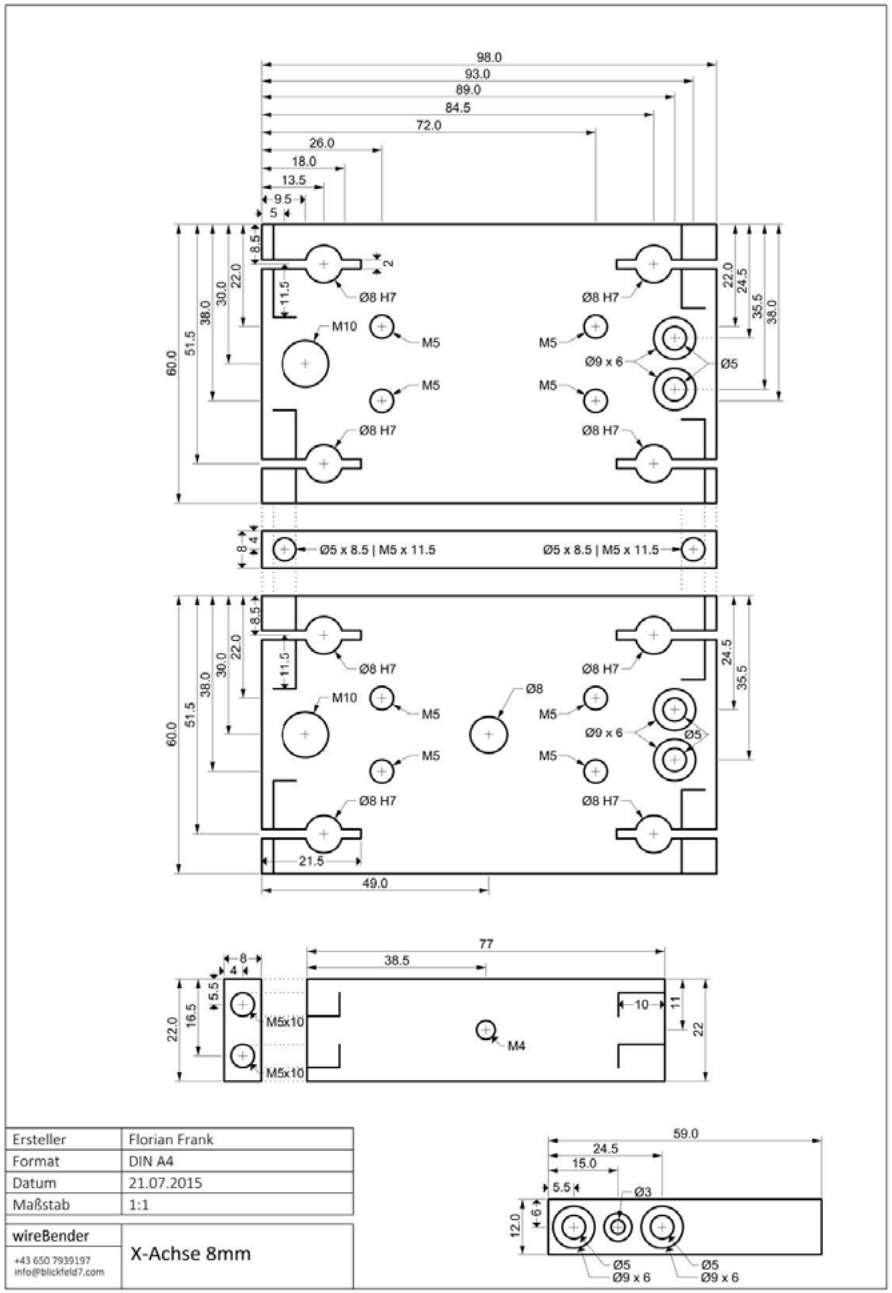


Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1
wireBender +43 650 7939197 info@blickfeld7.com	Div. Platten 15mm



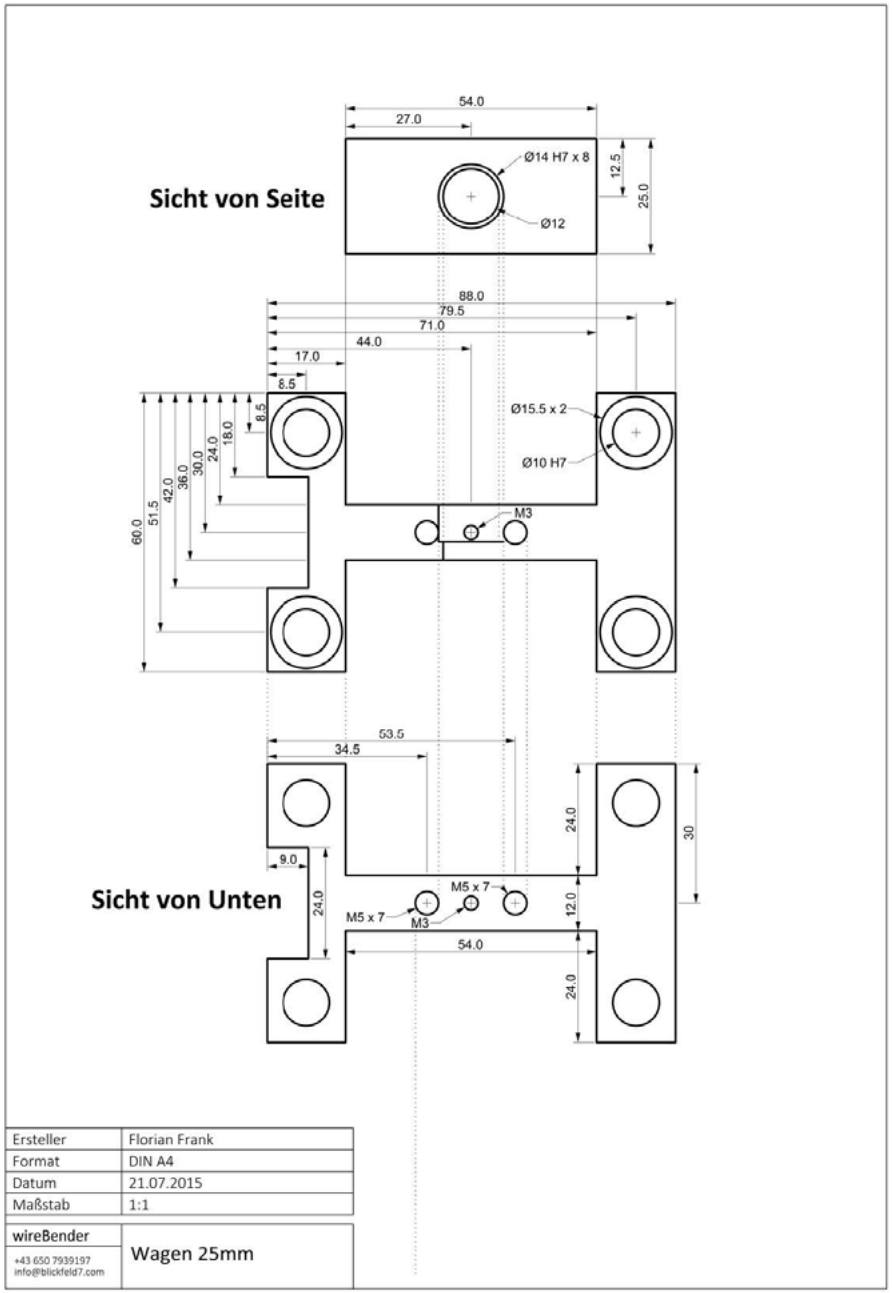
Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1

wireBender +43 650 7939197 info@blickfeld7.com	Y-Achse 8mm

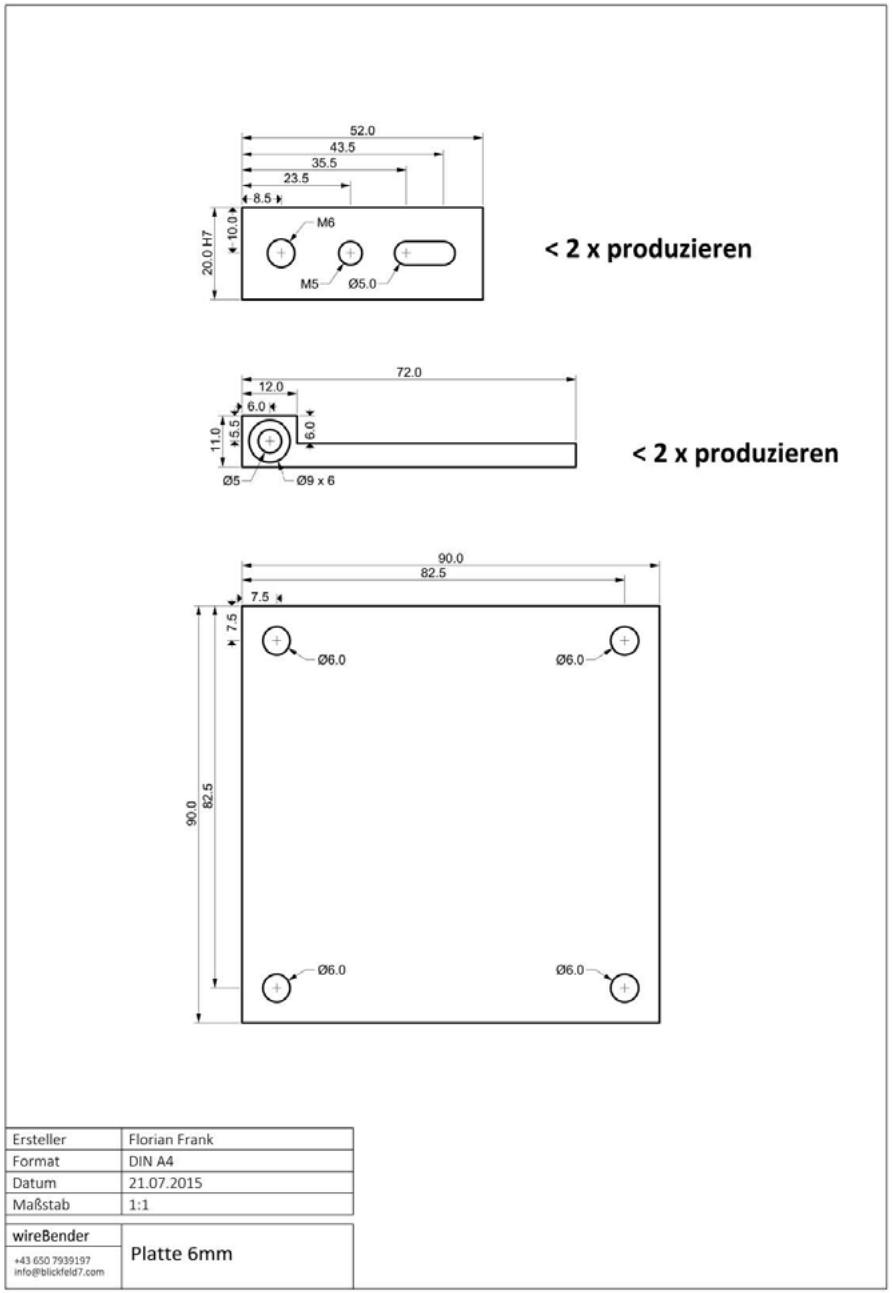


Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1

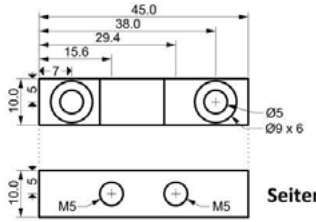
wireBender +43 650 7939197 info@blickfeld7.com	X-Achse 8mm



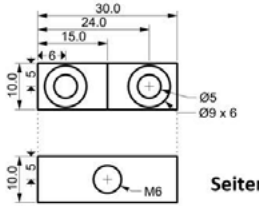
Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1
wireBender +43 650 7939197 info@blickfeld7.com	Wagen 25mm



Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1
wireBender +43 650 7939197 info@blickfeld7.com	Platte 6mm

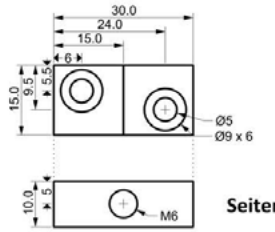


Seitenansicht

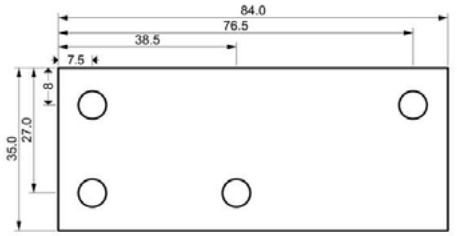


< 2 x produzieren

Seitenansicht



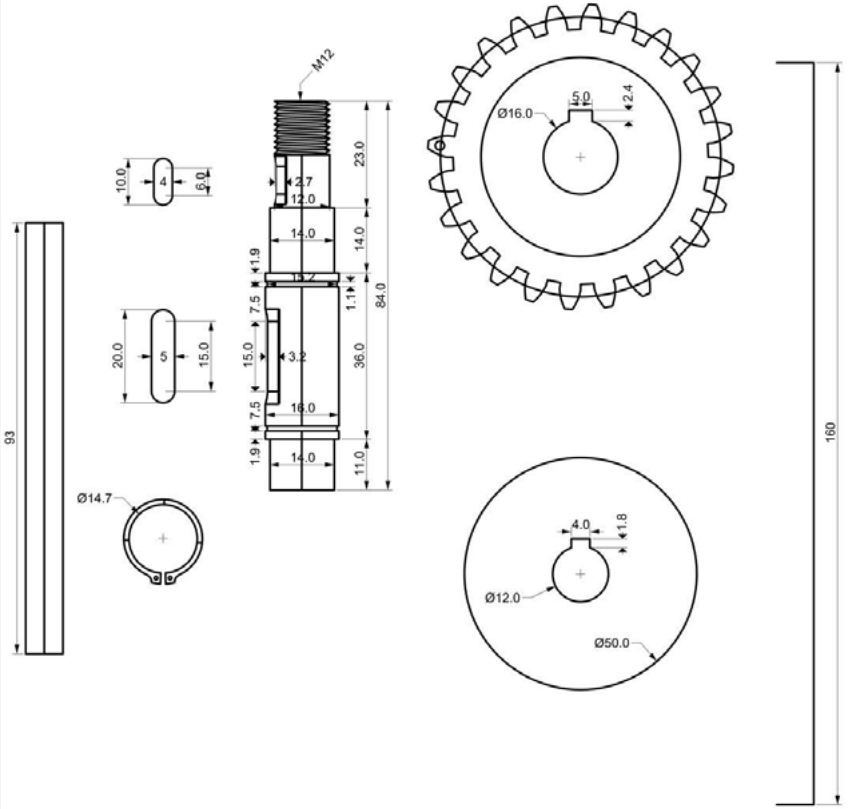
Seitenansicht



4mm !!!

Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1

wireBender +43 650 7939197 info@blickfeld7.com	Platte 10mm und 4mm



Ersteller	Florian Frank
Format	DIN A4
Datum	21.07.2015
Maßstab	1:1

wireBender +43 650 7939197 info@blickfeld7.com	Welle 16mm

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Innsbruck, Oktober 2016

Unterschrift

Danke

Erst durch die Unterstützung vieler Personen war diese Arbeit möglich. Ich bedanke mich bei allen, die mich während dieser Zeit begleitet und moralisch in meinem Vorhaben unterstützt haben.

Danke an meine Betreuer Marjan Colletti und Georg Grasser für das ausdauernde Interesse und Unterstützung in diesem Projekt.

Danke an das gesamte Team des REX |LAB am Institut für Experimentelle Architektur / Hochbau: Lukas Härtenberger, Johannes Ladinig und Pedja Gavrilovic.

Vielen Dank an meine lieben Eltern Marianne und Hanspeter, meiner Schwester Julia, die mich in jeder Lage unterstützt und gefördert und dies möglich gemacht haben.

Für die Unterstützung am Workshop „Resonance“: Shish-Yuan Wang, Yu-Ting Sheng und Yi-Chu.

Danke an folgende Unternehmen für die fachkundige Beratung und Unterstützung durch ihre Mitarbeiter: Koco-Motion, S&W Eloxal, SIKA Österreich, Stasto Automation, T&B Schrauben und Ortner & Stanger.

Ein großes Lob für die gelungene Zusammenarbeit und Danke für die aufopfernde Tätigkeit als Betonfachmann an Georg Breitenberger.

Danke für die Unterstützung in dieser Arbeit auch an Lisa Gummerer, Magdalena Wielander, Mirko Gamper, Christian Scheiber und Rupert Maleczek

Für die Hilfsbereitschaft in der Lösung von technischen Problemen: Tobias Strobl, Daniel Steffen, Simon Schiefer, den Mechatronikern David Lanthaler und Werner, Josef Schadlbauer, Gerhard Lener und Andreas Saxer

Besonderer Dank für die kräftige und unermüdliche Unterstützung fachlicher und persönlicher Natur bei der Planung und Umsetzung des Prototyps: Walter Windegger (Windegger Maschinen GmbH), Christoph Wegscheider und Ernest Hager.

und ein Danke an denen die die Zeit in Innsbruck zu etwas ganz Besonderes gemacht haben: Lisa, Kainz, Shifty, Motscho, Paolo, Benni, Magda, Jassi, Mac, Egon, Stegge, Thesi, Holzi, UTo, Armin, Schorsch, Zöschg, Faizan u.v.a.

Einen besonderen Dank auch an alle, die ich an dieser Stelle vergessen haben sollte. Ihre Hilfsbereitschaft und die der hier angeführten Personen verkörpert die Ziele dieser Arbeit und findet im Zitat einer bekannten österreichischen Schriftstellerin Genugtuung:

»Wenn jeder dem anderen helfen wollte, wäre allen geholfen.«
Marie Freifrau von Ebner-Eschenbach (1830 - 1916)

Gr

Mitgestalten!

Technologische Entwicklungen unserer Neuzeit scheinen keine allgemeine Vereinfachung der Dinge, sondern vielmehr eine Reizüberflutung und das gesellschaftliche Streben nach noch Mehr, Besser und Schneller, mitverursacht zu haben.

Die durch diese Entwicklung in berufliche und private Lebensbereiche hineingetragene Hast führt zum Ruf nach Entschleunigung. Dieser wird im Grundsatz der produktiven Zielsetzung einer gewinnorientierten Gesellschaft mehr und mehr verloren gehen. Solange es uns nicht gelingt die Eigendynamik, getrieben von Effektivität, Komplexität und dem hektischen Treiben nach noch Besserem zu durchbrechen und den technologischen Fortschritt zum Nutzen ALLER einzusetzen, wird diese Hast nicht abbrechen. Der „menschliche“ Fortschritt wird zukünftig einen Nutzen für Wenige, aber eine negative Auswirkung auf die Lebensbereiche Vieler haben. Prognosen für die kulturelle Entwicklung unserer Gesellschaft, unter der zunehmenden Automatisierung von Berufen führen bei einem Weiterfolgen der derzeitigen Ziele zu einem Überflüssigwerden für manch lebenserfüllende Aufgabe.

Gerade die Aufgabe der Kreativen besteht darin, in dieser Dynamik einen Weg zu gestalten, der die Fortschritte für Jeden zugänglich macht, um an der Gestaltung der kulturellen Identität mitzuwirken und sich von der Ohnmacht der scheinbar geistlosen Entwicklung zu lösen.

Mit dieser Arbeit hoffe ich Mut, Wille und Ehrgeiz zu wecken, sodass das Engagement für den einfachen und freien Zugang zu Wissen für eine Erweiterung von (Mit-)Gestaltungsmöglichkeit, auch über die Architektur hinaus, in allen Belangen unserer Kultur gesteigert wird.