Leuphana Universität Lüneburg

Operating Systems

Wintersemester 2019/2020

Plotten von vektorbasierten Grafiken in der Vertikalen Ebene durch Triangulation

Bruno Borchardt

51316

im Studium für B.Sc. Informatik-Ingenieurwesen

bruno.borchardt@tuhh.de

Sebastian Meyer

3030618

Ingenieurswissenschaften Automatisierungtechnik

Ingenieurswissenschaften Produktionstechnik

sebastian.meyer@stud.leuphana.de

31.03.2020

Gliederung

Einleitung

In der heutigen Leistungsgesellschaft kommen immer mehr Menschen an ihre psychischen Grenzen. Psychische Überlastung oder Burnout ist heute eine anerkannte Krankheit. Zum Burnout führt oft die fehlende Möglichkeit effektiv und schnell zu entspannen. Viele unterschätzen dabei die positive Auswirkung von Kunst auf die Psyche. Während das passive genießen von Kunst schon positive Auswirkungen haben kann, zeigt vor allem das aktive Gestalten eine deutliche Wirkung. Neben der Ablenkung von schwierigen Themen im Alltag, führt das eigenständige Gestalten auch zu einer stärkeren Persönlichkeit. (van der Vennet und Serice 2012)

Ein Problem, vor dem viele Menschen stehen, ist allerdings das fehlende künstlerische Talent. Das Ziel dieser Ausarbeitung ist das Bereitstellen von Werkzeugen zum eigenen Schaffen von Wandgemälden, um nicht auf große motorische oder zeichnerische Fähigkeiten angewiesen zu sein. Hier wird als Lösungsansatz die Automatisierung der Einteilung der Ebene in einzelne Bildelemente gewählt. Das Zeichen von Trennlinien zwischen zwei Flächen soll also nicht von Hand erfolgen, sondern vor Beginn des eigentlichen Malprozesses maschinell gelöst werden.

Die Vorgehensweise bei der Umsetzung ist die Folgende: Zu Beginn wird sich auf die Erfassung der gegenwärtigen Angebote konzentriert und auf bestehenden Lösungen aufbauend ein Grundkonzept entwickelt. Da das Problem auch eine Hardwarekomponente besitzt, ist auch eine zeitnahe Fertigung der notwendigen Elemente gewünscht, um die verschiedenen Teilsysteme besser aufeinander abstimmen zu können. Die daran anschließende Arbeit wird sich auf das Bereitstellen der Softwareseite fokussieren. Programmiert werden zwei große und prinzipiell voneinander unabhängige Teile. Während eine Aufgabe darin besteht, Bilddaten in Bewegungsbefehle für den Plotter zu übersetzen, muss auf der anderen Seite eine Möglichkeit gefunden werden, die Bewegungsbefehle des Plotters in Folgen von Motorpositionen darzustellen. Auch die Weitergabe von Daten zwischen beiden Hälften muss spezifiziert werden. Abschließend werden letzte Fehler behoben und eine Benutzeroberfläche geschaffen.

Planung

Für die Umsetzung eines solchen Vorhabens ist es notwendig, im Voraus zu planen. Teile dieses Planungsprozesses haben andere Personen bereits abgeschlossen, auf deren Arbeit wird nun aufgebaut. Starke Anlehnung findet das Projekt an den „Stringent, the $15 Wall Plotter“ welcher die Hardwareauswahl stark vereinfacht hat. (Stridsman 2018)

Anforderungen

Es gilt eine Apparatur, genannt Plotter, zu entwickeln, die in der Lage ist, sich auf einer vertikalen Ebene frei zu bewegen. Dabei muss sie präzise kartesische Koordinaten anfahren und während der Bewegung durch heben und senken eines Zeichengeräts Linien malen. Um eine gewisse Eigenständigkeit zu gewährleisten, müssen die Bewegungsanweisungen als Datei übergeben werden. Des Weiteren muss der Plotter kompakt, transportabel, autark als auch vielerorts einsetzbar sein.

Arbeitsweise und Vorstellung der Hardware

Erreicht werden die Anforderungen durch einen Aufbau, indem zwei 28BYJ-48 Schrittmotoren in einem Gehäuse jeweils eine Seiltrommel antreiben. Die Enden der Seile werden an zwei gleich hoch liegenden Punkten befestigt. Ebenfalls in dem Gehäuse befinden sich die ULN2003A Ansteuerungselektroniken der 28BYJ-48 Schrittmotoren, ein Servo-Motor zur Bewegung des Zeichengeräts, eine Anker Powercore 5 Ah Powerbank zur Stromversorgung und ein Raspberry Pi Zero W als Prozessor. Das Chassis selbst, sowie die Seilrollen, stammen aus dem 3D Drucker, das Modell des Chassis wurde dem Stringent Projekt entnommen. Lediglich die Seilrollen mussten neu in CAD gezeichnet werden, da die des Stringent Projektes zu groß für die Achsen unserer 28BYJ-48 Schrittmotoren war. Als vorrübergehendes Seil dient Zahnseide, da sie verfügbar war und den Ansprüchen gerecht wird dünn, unnachgiebig und belastbar zu sein.

Raspberry Pi

Der Raspberry Pi Zero W mit dem Raspbian Betriebssystem bietet eine gute Grundlage für das Projekt. Verwendet wird die Raspbian Buster Lite Edition, diese verzichtet auf eine grafische Benutzeroberfläche und unnötige damit verbundene Software und beschränkt sich somit auf das wesentliche. (Raspberry Pi Foundation 2020) Zudem wird keine grafische Oberfläche benötigt, sondern lediglich ein SSH basierter Fernzugriff. Da der Raspberry Pi Zero W durch eine Powerbank mit Strom versorgt werden soll, ist der geringere

Stromverbrauch der aus dem Headlessbetrieb resultiert ebenfalls wünschenswert. Raspbian als Betriebssystem stellt die grundlegenden Funktionen für die Entwicklung bereit, den Compiler für C++, die remote Shell, den SFTP Server zum Daten übertragen und die Schnittstelle zu den digitalen Ein- und Ausgängen des Raspberry Pi Zero W mit welchen die Aktoren des Systems gesteuert werden. Raspbian selbst basiert auf Debian, nutzt also den Linux Kernel. (raspbian.org 2020)

28BYJ-48 Schrittmotor und ULN2003A Darlington Array

Abbildung 1 28BYJ-48 und ULN2003A (eBay)

Der verwendete 28BYJ-48 Schrittmotor hat laut Datenblatt eine Betriebsspannung von 5v und 64 Schritte pro Umdrehung, diese werden durch das integrierte Getriebe auf 513 Schritte pro Umdrehung gesetzt. (28BYJ-48 Stepper Motor Datasheet) Die auf der Achse sitzende Spule ist vom Durchmesser so dimensioniert, dass der Umfang genau 100 mm beträgt was eine Auflösung von 0,2 mm bei sehr niedriger Toleranz pro Schritt gibt.

Mietgeliefert zu dem 28BYJ-48 Schrittmotor wird ein Treiber PCB, welches auf einem ULN2003A Darlington Array IC basiert. Dieser vereint 4 Darlington Transistor Paare in einem Gehäuse und schaltet die Spule des 28BYJ-48 Schrittmotor zum Ground durch. (STMICROELECTRONICS)

SG90 9g Micro Servo

Der SG90 9g Micro Servo wird verwendet um den gesamten Plotter von der Wand weg zu drücken.

Zusammenfassung

Die entstandene Schaltung kann im Anhang unter Abbildung 5 gefunden werden.

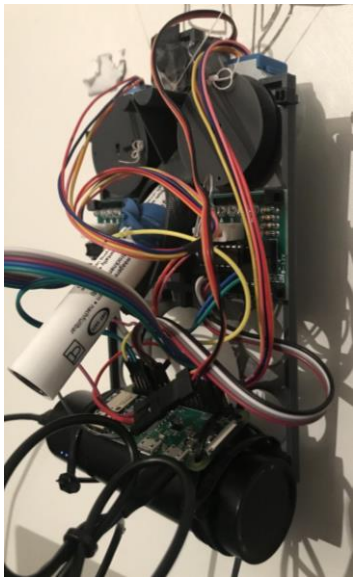

Abbildung 2 Fertiger Aufbau (eigene Aufnahme)

Systementwurf

Das grundlegende Problem des Plotters besteht darin, dass nicht flächig gemalt werden kann, sondern nur Linien gezeichnet werden. Entsprechend schwierig ist es, eine Pixelgrafik in Bewegungsbefehle zu übersetzen. Im Gegensatz dazu hat eine Vektorgrafik das ideale Format, um durch den Plotter dargestellt zu werden. Das hier genutzte Format ist Scalable Vector Graphics, kurz SVG. SVG wird als Textdatei gespeichert und ist in XML geschrieben. (Bellamy-Royds et al. 2018)

Das Projekt ist in C++ geschrieben. Der Quelltext übersteigt in seinem Umfang das hier darstellbare, dieses Kapitel soll aber eine grobe Übersicht geben und einige Details als Beispiel näher betrachten. Damit ist klar, dass dieses Kapitel ohne Anspruch auf Vollständigkeit verfasst wurde, der gesamte Code kann allerdings auf GitHub eingesehen werden. (Borchardt und Meyer 2020)

Die Programmierung wurde folgenderweise aufgeteilt: Bruno Borchardt hat das Einlesen einer SVG-Datei und Übersetzen in Bewegungsbefehle in der Ebene realisiert und Sebastian Meyer hat die Infrastruktur, diese Bewegungsbefehle in Folgen von Motorpositionen umzuwandeln, aufgebaut. Alle Aufgaben mit Hardwarebezug wurden von Sebastian Meyer übernommen.

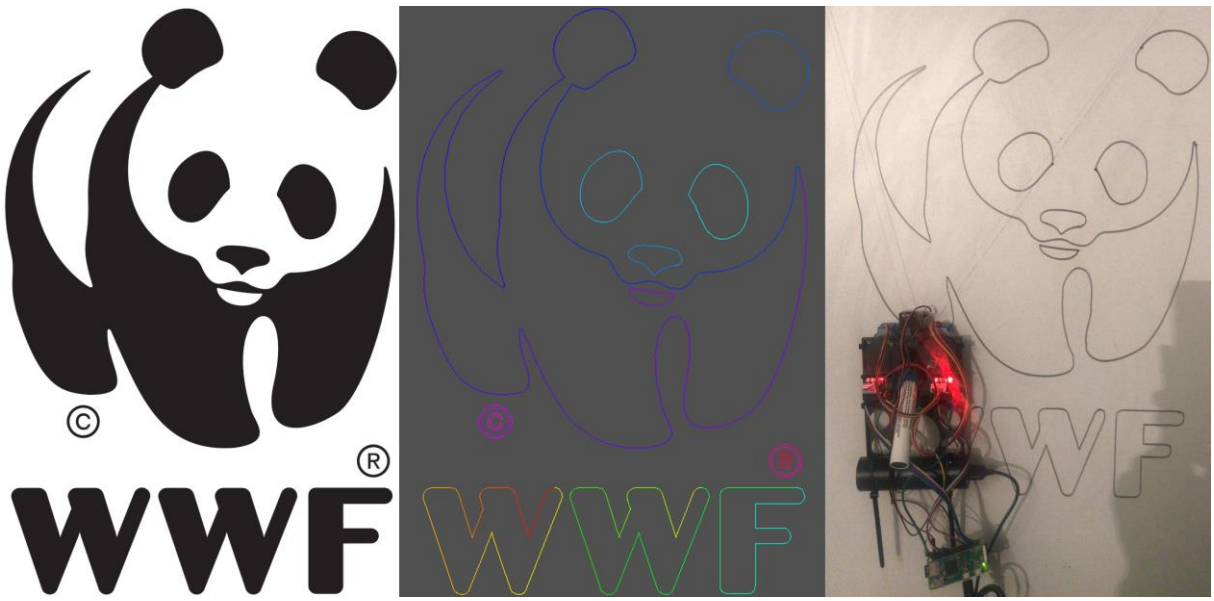

Abbildung 3 Vergleich von originaler Datei, daraus generierten Bewegungsinformationen und dem geplotteten Ergebnis (WWF)

Einlesen der SVG-Datei

Die hier näher erläuterten Funktionen und Strukturen sind im Anhang unter Auszüge des SVG Parser Codes aufgeführt.

Es wurde keine gute Bibliothek zum Einlesen von Vektorgrafiken für C/C++ gefunden, weswegen der Einleseprozess komplett selbst realisiert wurde. Dies ermöglicht, es direkt beim Parsen auch die Bewegungsbefehle zu generieren, anstatt als Zwischenergebnis eine Baumdarstellung der SVG-Datei zu erhalten. Wenn auch die Baumdarstellung einen flexibleren Umgang mit den Daten ermöglichen würde, wurde sie in diesem Projekt nicht als notwendig erachtet.

Der Grundaufbau des Parsers bildet die Dateistruktur eines SVG-Dokumentes nach, vereinfacht allerdings einen zentralen Aspekt: Während der SVG-Standard in einem SVG-Dokument das einbetten von anderen SVG-Dokumenten erlaubt und für diese jeweils die Definition eines neuen Gültigkeitsbereiches (View Box) vorsieht, gilt in diesem Programm immer ein globaler Gültigkeitsbereich, eingebettete SVG-Dateien werden nur um die spezifizierten Koordinaten verschoben, können jedoch überall innerhalb des globalen Gültigkeitsbereiches Objekte definieren. Die globale View Box wird am Anfang des Einleseprozesses einmal in der Funktion *read::evaluate_svg()* festgelegt, in Koordinaten der Zeichenfläche übersetzt und gespeichert. Am Ende von *read::evaluate_svg()* wird die Funktion *read::evaluate_fragment()* aufgerufen, welche das komplette Dokument übersetzt.

Der Parameter dieser Funktion *const Transform_Matrix& transform*, im Aufruf durch *read::evaluate_svg()* durch lokale Variable *to_board* besetzt, soll an dieser Stelle näher betrachtet werden. Das übersetzten von SVG-Koordinaten zu Kartesischen Koordinaten der Zeichenebene ist eine Lineare Transformation im dreidimensionalen Raum. Diese Transformation wird im Programm mittels des Types *Transform_Matrix* realisiert. Die Eleganz dieser Lösung besteht darin, dass auch die Verkettung zweier linearer Transformationen wieder eine lineare Transformation darstellt und damit auch als Transformationsmatrix gespeichert werden kann. Wenn einen Teilbereich der SVG-Datei also in einem anderen Koordinatensystem spezifiziert wurde, kann auch ein in diesem System dargestellter Vektor direkt in die Koordinaten der Zeichenebene übersetzt werden, wenn man ihn mit dem Produkt der Transformationsmatrix ins Zeichenebenensystem und der Transformationsmatrix ins Zwischensystem multipliziert.

Um zu garantieren, dass die Transformation ins Zeichenebenensystem an keiner Stelle vergessen wird, ist Ergebnis des Produktes aus Transformationsmatrix und Vektor in einem

SVG-System der Typ *Board_Vec*. Die unterschiedlichen Typen der sonst strukturgleichen *Board_Vec* und *Vec2D* verhindern das Vergessen einer Transformation.

Die Funktion *read::evaluate_fragment()* ist das Herzstück beim Einlesen. Hier wird die SVG elementweise eingelesen und erkannt, um was für Typen von Elementen es sich handelt. Ist der Typ unbekannt, etwa Metainformationen oder Anweisungen für Animationen, wird das Element ignoriert und das nächste Element eingelesen.

Beispielhaft wird jetzt für die Form des Kreises darauf eingegangen, wie aus den Daten des SVG-Elementes *circle* die Bewegungsbefehle für den Plotter in der Funktion *draw::circle()* generiert werden. Zu Beginn wird die tatsächliche Transformationsmatrix aus der Matrix des Systems außerhalb des Kreises und einer möglichen Transformation spezifisch für den Kreis gebildet. Die nächsten drei Zeilen lesen aus dem, den Kreis spezifizierenden String (genau genommen eine Referenz auf den entsprechenden Bereich im originalen String, also ein *std::string_view*), die Koordinaten des Mittelpunktes, sowie den Radius aus. Mit den jetzt gegebenen Daten wird zuerst ein Bewegungsbefehl an den Plotter zum Punkt auf dem Kreis mit größter x-Koordinate gegeben, Dann wird in der Schleife immer vom momentanen Punkt zum nächsten Punkt auf dem Kreis eine Gerade gezogen. Aus wie vielen Graden der Kreis dargestellt wird, bestimmt der Funktionsparameter *resolution*.

Die Funktion *draw::path()* muss etwas von dem Rezept, nachdem die *draw*-Funktionen für die einfachen Formen geschrieben wurden, abweichen, da der Pfad das Element mit den am Abstand meisten Möglichkeiten in einer SVG-Datei ist. Dementsprechend aufwändig ist auch die Implementierung dieses Programmteils, auch, da in größerem Umfang als an anderer Stelle auf Fehlerbehandlung Rücksicht genommen werden muss. Größere SVG-Dateien bestehen oft allerdings fast ausschließlich aus Pfaden, weswegen die nicht vollständige Implementierung eines Pfadinterpreters die Bildauswahl erheblich einschränken würde. Was für die gesamte SVG die Funktion *read::evaluate_fragment()* ist, übernimmt für Pfade die Funktion *draw::path()*. Hier wird immer der Typ des nächsten Pfadelementes erkannt. Für einfache Pfadelemente wie Geraden werden auch direkt die Bewegungsbefehle generiert. Bézier Kurven und Ellipsenbögen (*arc*) sind aufgrund derer Komplexität in jeweils eigene Funktionen ausgelagert, wobei aus dem die Bewegungsbefehle Ellipsenbogen am schwierigsten zu generieren sind, da der Mittelpunkt der Ellipse, genau wie die Begrenzungswinkel, nur implizit gegeben werden.

Testen

In einem prozeduralen Programm wie diesem ist das Testen einzelner Funktionen nicht immer effektiv, da in einer isolierten Testumgebung nicht auf Nebeneffekte Rücksicht genommen werden kann. Die Teststrategie hier ist deswegen, möglichst viele verschiedene SVG-Dateien einzulesen. Um schnell zu sehen, ob eine Datei korrekt interpretiert wurde, gibt es die Möglichkeit, eine Bitmap zu erstellen, die den Bewegungsablauf farbig darstellt. Die Farbwahl durchläuft für das gesamte Bild einmal alle Winkel (*Hue*) im HSV-Farbraum (Siehe Abbildung 3). Startfarbe ist dabei Rot, gefolgt von Gelb und so weiter. Die Bibliothek *libBMP.h* zum Export eines Arrays als Bitmap stammt dabei aus der Veranstaltung Prozedurale Programmierung der Technischen Universität Hamburg und ist kein Eigenwerk.

Das Bashey-Bruno-Format

Um die parallele Entwicklung von SVG Parser und Plotter Seite zu vereinfachen, Ist das Gesamtsystem in zwei Hälften aufgeteilt. Der SVG Parser übersetzt die SVG in ein Zwischenformat und speichert dieses als Textdatei ab. Die Sprache zur Beschreibung der Bewegungsbefehle ist das extra dafür entwickelte Bashey-Bruno-Formal, kurz bbf.

Das Format unterscheidet dabei nur zwei verschiedene Befehle, welche beide jeweils eine x-Koordinate als ersten Parameter und eine y-Koordinate als zweiten Parameter besitzen. Der *move_to* Befehl ist durch eine ‘0’ vor den mit Leerzeichen getrennten Parametern von dem *draw_to* Befehl mit einer führenden ‘1’ zu unterscheiden. Ein *move_to* wird vom Plotter ausgeführt, indem er sich mit gehobenem Stift von der momentanen Position zu den spezifizierten Koordinaten bewegt. Der *draw_to* Befehl funktioniert analog, allerdings mit abgesenktem Stift. Wichtig für *draw_to* ist zudem, dass die neue Position auf einem geradlinigen Weg von der momentanen Position aus erreicht wird.

Umsetzung von BBF in Bewegung

Der im Folgenden beschriebene Code ist im Anhang unter GitHub und Code

Der gesamte Code, als auch ergänzende Fotos und Videos sind ebenfalls in der GitHub Repository unter https://github.com/brunizzl/sooperDooperPlootter anzufinden.

Stepper08.cpp zu finden.

Um die Bewegungsanweisungen der .bbf umzusetzen bedarf es code für den Pi. In diesem werden mit der wiringPi Bibliothek die GPIOs des Raspberry Pis angesteuert um den G90 9g Micro Servo und die 28BYJ-48 Schrittmotoren zu steuern. Hierzu wird erst eine Schrittmotorklasse erstellt, diese enthält die aktuelle Schrittzahl, eine Funktion für das Setup der Pins und eine Funktion die genau einen Schritt in eine beliebige Richtung mit einer beliebigen anschließenden Verzögerung macht. Wie die Eingänge des ULN2003A geschaltet werden müssen, um den Schrittmotor zu bewegen, ist in einen 4x8 Array festgelegt. Es wurde sich für Halbschritte entschieden, die Pin-Zustände für den ULN2003A werden dann aus dem Modulo 8 der aktuellen Schrittzahl und dem Array entnommen. Beim Konstruieren eines Objektes der Klasse werden dann die angeschlossenen Pins definiert. Es gibt nur zwei statische Objekte der Klasse, den linken und den rechten Schrittmotor.

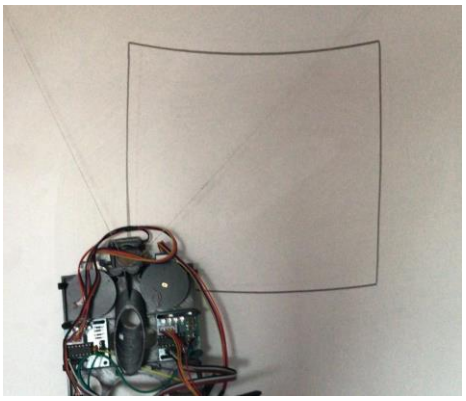

Abbildung 4 Fehler bei linearer Interpolation zwischen zwei Punkten (eigene Aufnahme)

Dann wird eine Plotterklasse definiert, sie hat nur ein Objekt und enthält die beiden Objekte der Schrittmotorklasse. In der *setup()* Methode dieser Klasse werden die *setup()* Methoden der Schrittmotoren ausgeführt, aber unter anderem auch die geometrischen Informationen über den Abstand der beiden Aufhängungspunkte als auch die aktuelle Seillänge der Schrittmotoren ausgelesen, letztere werden dann in die Schrittzahl der Schrittmotoren geschrieben. Diese Klasse besitzt einige Funktionen, zum Beispiel eine, die die aktuelle Position des Plotters ausgibt. Dazu gibt es eine *penMode()* Funktion, mit welcher der Zustand des Stiftes, beziehungsweise die Position des Servos verändert wird. *Penmode = 0* bedeutet kein Zeichnen, 1 heißt der Stift zeichnet. In einer späteren Version können dann beliebig viele weitere Stifte definiert werden.

Die wohl wichtigsten Funktionen sind *go2()* und *draw_bbf().* Die erste nimmt eine Zielkoordinate, dann errechnet sie sie benötigten Seillängen und die Differenz zur aktuellen Seillänge. Aus den beiden Differenzen wird dann ein Verhältnis gebildet in dem die Schrittmotoren nun Schritte machen, also alle X Schritte von Schrittmotor Links soll Schrittmotor Rechts einen Schritt tätigen. Dadurch bewegt sich der Plotter auf einer annähernd geraden Linie (siehe Abbildung 4). Diese Annäherung reicht jedoch, da in der .bbf Linien in viele einzelne Koordinaten aufgelöst sind. Die Funktion *draw_bbf()* liest nun die .bbf ein, dafür geht sie erst einmal alle Zeilen durch und überprüft, ob sich Punkte außerhalb des möglichen Druckbereich befinden, sprich das Objekt zu weit links oder rechts ist oder ob es zu breit oder zu weit oben ist. Wenn es keine Fehler gegeben hat, startet der Druckprozess

und die Anweisungen der .bbf werden Zeile für Zeile ausgeführt. Sie hat noch Parameter für die Verschiebung des Objektes in x- und y-Richtung. In der *end()* Funktion werden dann die aktuellen Seillängen gespeichert und alle Pins wieder auf LOW geschaltet.

Um das Neukalibrieren der Seillängen nach einem Programmabbruch zu verhindern, wurde ein Handler für das Str+C Kommando geschrieben. Der Handler stellt sicher das die *end()* Funktion korrekt ausgeführt und erst dann das Programm beendet wird.

Zusammenfassung

Das Interpretieren der SVG-Datei ist selbst umgesetzt und verzichtet auf übersetzten in eine Baumstruktur. Die zentrale Funktion für das Parsen ist *read::evaluate_fragment()*. Ein großer Teil des Programmes ist die Interpretation von Pfad-Befehlen, da diese einen sehr großen Funktionsumfang haben. Zum Testen können die Bewegungsabläufe auf einer Bitmap dargestellt werden, wobei ein Farbübergang die Reihenfolge visualisiert, in der die Zeichenbefehle gestellt werden. Die so generierten Bewegungsanweisungen im Zwischenformat bbf werden vom Pi eingelesen und in GPIO Signale umgewandelt.

Fazit und Ausblick

Alles in allem war das Projekt ein voller Erfolg. Das Ziel, ein Gerät für das automatische Bemalen von Wänden zu entwickeln, ist geglückt. Die Präzession, mit der ein Bild auf die Zeichenebene übertragen werden kann ist nicht nur besser als erwartet, sondern es sind mit bloßem Auge so gut wie keine Ungenauigkeiten zu erkennen.

Für die Zukunft ist denkbar, die Hardware weiter zu optimieren, zum Beispiel ist die Ergänzung mit weiteren Stiften das nächste Ziel. Des Weiteren wäre es eventuell sinnvoll, den Raspberry Pi Zero W durch einen Mikrocontroller mit weniger Leistung auszutauschen, zum Beispiel durch einen ESP32. Dieser hätte den Vorteil, weniger Strom zu verbrauchen, und die Anschaffungskosten sind geringer.

Um das ganze benutzerfreundlicher zu machen, ist angedacht, ein Webpage-basiertes Interface zu bauen, in dem man den Plotter kalibrieren und Dateien zum Zeichnen hochladen kann.

Auch das Verarbeiten der SVG kann noch erweitert werden. Das programminterne Darstellen der SVG als Baum wäre dabei der erste Schritt. Durch den direkten Zugriff auf die Struktur des Bildes ist dann nicht mehr nur das wiederholte Einlesen möglich, sondern auch eine Anpassung der Daten an die besonderen Bedürfnisse des Plotters. An erster Stelle steht dabei eine Änderung der Reihenfolge, in der einzelne Bildelemente gezeichnet werden. Die Wegstrecke zwischen allen Elementen kann dabei minimiert und die Druckzeit signifikant verkürzt werden.

Eine weitere Möglichkeit in der Verarbeitung einer SVG-Datei ist ab einer gewissen Liniendicke nicht mehr die Mitte der Linie als Pfad auszugeben, sondern die beiden Ränder. Mit Abstand am aufwändigsten, allerdings auch mit einer deutlichen Qualitätszunahme des Ergebnisses, wäre die Analyse der gegenseitigen Überdeckung von Geometrieelementen. Zeichenbefehle würden dann nur für die sichtbaren Bereiche generiert werden. Dafür ist allerdings nicht nur das Bestimmen von Schnittpunkten zwischen z.B. Bézierkurven und Ellipsenbögen notwendig, sondern es muss determiniert werden können, ob sich ein Objekt geometrisch innerhalb eines anderen befindet.

Querverweise

Literaturverzeichnis

28BYJ-48 Stepper Motor Datasheet. Online verfügbar unter https://www.st.com/resource/en/datasheet/uln2001.pdf, zuletzt geprüft am 29.03.2020.

Bellamy-Royds, Amelia; Brinza, Bogdan; Lilley, Chris; Schulze, Dirk; Storey, David; Williger, Eric (2018): Scalable Vector Graphics (SVG) 2. W3C. Online verfügbar unter https://www.w3.org/TR/SVG/Overview.html, zuletzt aktualisiert am 02.10.2018, zuletzt geprüft am 29.03.2020.

Borchardt, Bruno; Meyer, Sebastian (2020): brunizzl/sooperDooperPlootter. Online verfügbar unter https://github.com/brunizzl/sooperDooperPlootter, zuletzt aktualisiert am 29.03.2020, zuletzt geprüft am 29.03.2020.

Raspberry Pi Foundation (Hg.) (2020): Raspbian - Raspberry Pi Documentation. Online verfügbar unter https://www.raspberrypi.org/documentation/raspbian/, zuletzt aktualisiert am 30.03.2020, zuletzt geprüft am 30.03.2020.

raspbian.org (2020). Online verfügbar unter https://www.raspbian.org/RaspbianAbout, zuletzt aktualisiert am 30.03.2020, zuletzt geprüft am 30.03.2020.

STMICROELECTRONICS: ULN200 Datasheet, zuletzt geprüft am 29.03.2020.

Stridsman, Fredrik (2018): Stringent, the $15 Wall Plotter - Hackster.io. Online verfügbar unter https://download.mikroe.com/documents/datasheets/step-motor-5v-28byj48-datasheet.pdf, zuletzt aktualisiert am 29.03.2020, zuletzt geprüft am 29.03.2020.

van der Vennet, Renée; Serice, Susan (2012): Can Coloring Mandalas Reduce Anxiety? A Replication Study. In: *Art Therapy* 29 (2), S. 87–92. DOI: 10.1080/07421656.2012.680047.

Abbildungsverzeichnis

Anhang

Schaltplan

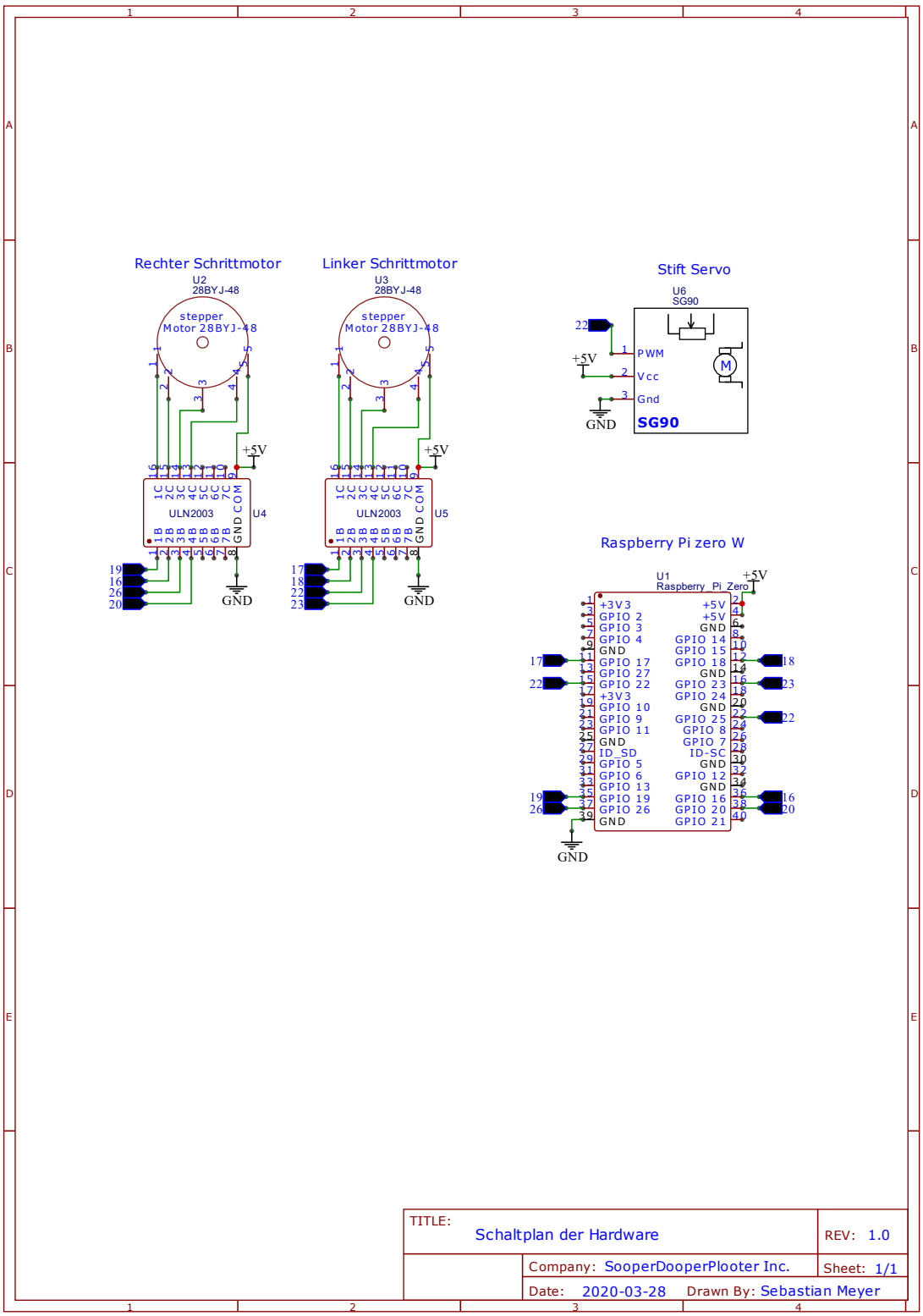

Abbildung 5 Schaltplan des Aufbaus (erstellt in easyEDA)

GitHub und Code

Der gesamte Code, als auch ergänzende Fotos und Videos sind ebenfalls in der GitHub Repository unter https://github.com/brunizzl/sooperDooperPlootter anzufinden.

Stepper08.cpp

```
#include <wiringPi.h>

          #include <stdio.h>

          #include <cmath>

          #include <iostream>
          #include <fstream>

          #include <signal.h>

          #include <chrono>

          #define steps_per_mm 40

          //#define width 577
          //#define width_s width * steps_per_mm

          #define servo_pin 13
          #define pd 15 // pen up val
          #define pu 165 // pen down val

          //using namespace std;

          // Sequenz in der die Spulen des Schrittmotors angeschaltet werden müssen
          const bool sequence[][4] =
          {
              { LOW,  LOW,  LOW,  HIGH },
              { LOW,  LOW,  HIGH, HIGH },
              { LOW,  LOW,  HIGH, LOW },
              { LOW,  HIGH, HIGH, LOW },
              { LOW,  HIGH, LOW,  LOW },
              { HIGH, HIGH, LOW,  LOW },
              { HIGH, LOW,  LOW,  LOW },
              { HIGH, LOW,  LOW,  HIGH }
          };

          struct Coord_mm
          {
              double x;
              double y;
          };

          struct Coord_steps
          {
              int x;
              int y;
```

```
};

int mm2steps(double mm)
{
    return std::round(mm * steps_per_mm);
}

double steps2mm(int steps)
{
    return double(steps) / double(steps_per_mm);
}

// Schrittmotor Klasse
class stepper_motor
{
public:
    stepper_motor (char, char, char, char);
    void setup();
    void step(int, int);
    unsigned int current_step = 0;
    char pins[4];
};



// initialisierung mit Anschluss-Pins
stepper_motor::stepper_motor(char IN1, char IN2, char IN3, char IN4)
{
    pins[0] = IN1;
    pins[1] = IN2;
    pins[2] = IN3;
    pins[3] = IN4;
}

// setup der IOs
void stepper_motor::setup()
{
    for(int i = 0; i < 4; i++)
    {
        pinMode(pins[i], OUTPUT);
    }
}

// Mach schritte
void stepper_motor::step(int direction, int speed = 6)
{
    if (direction > 0)
    {
        this->current_step++;
        int seq = this->current_step % 8;
        for(int p = 0; p < 4; p++)
        {
            digitalWrite(this->pins[p], sequence[seq][p]);
        }
        delay(speed);
    }
    else if (direction < 0)
```

```
    {
        this->current_step--;
        int seq = this->current_step % 8;
        for(int p = 0; p < 4; p++)
        {
            digitalWrite(this->pins[p], sequence[seq][p]);
        }
        delay(speed);
    }
}


// Plotter Class-------------------------------------------------------
class plotter
{
    stepper_motor &l;
    stepper_motor &r;
    int mode = 1;
public:
    plotter (stepper_motor &, stepper_motor &);
    void setPen(int);
    void go2(double, double, int);
    Coord_mm get_position_mm();
    void draw_bbf(std::string, double, double, int);
    double width;
    void setup();
    void end();
};

plotter::plotter(stepper_motor &A, stepper_motor &B)
    : l(A), r(B)
{
}

void plotter::setup()
{
    this->l.setup();
    this->r.setup();
    char k;
    double W, L, R;
    std::ifstream configfile("config.txt");
    configfile >> k >> W >> k >> L >> k >> R;
    configfile.close();
    this->l.current_step = mm2steps(L);
    this->r.current_step = mm2steps(R);
    this->width = W;
    std::cout << W << "mm width " << L << "mm L " << R << "mm R\n";
}

void plotter::end()
{
    std::ofstream output("config.txt");
    output << "W " << width
           << " L " << steps2mm(this->l.current_step)
           << " R " << steps2mm(this->r.current_step);
    output.close();
    for(int p = 0; p < 4; p++)
```

```
    {
        digitalWrite(this->l.pins[p], LOW);
        digitalWrite(this->r.pins[p], LOW);
    }
    this->setPen(0);
    digitalWrite(servo_pin, LOW);
    std::cout << "Config Gespeichert\n";
}

void plotter::setPen(int penMode)
{
    if(penMode == 0)
    {
        //stift hoch
        pwmWrite(servo_pin, pu);
        if (this->mode != penMode) delay(500);
    }
    else
    {
        //stift runter
        pwmWrite(servo_pin, pd);
        if (this->mode != penMode) delay(500);
    }
    this->mode = penMode;
}

void plotter::go2(double x, double y, int penMode = 0)
{
    std::cout << "PenMode " << penMode << " to x: " << x << " y: " << y << "\n";
    this->setPen(penMode);

    int steps_left = mm2steps(sqrt(x * x + y * y)); //total cable step
    int steps_right = mm2steps(sqrt((this->width - x) * (this->width - x) + y * y));
    int delta_l = steps_left - this->l.current_step;
    int delta_r = steps_right - this->r.current_step;

    //double ratio = std::abs(double(delta_l)) / std::abs(double(delta_r));

    if(std::abs(delta_l) > std::abs(delta_r))
    {
        double ratio = std::abs(double(delta_l)) / std::abs(double(delta_r));
        double counter = 0;
        while(this->l.current_step != steps_left)
        {
            this->l.step(delta_l);
            counter++;

            if (counter > ratio)
            {
                r.step(delta_r, 0);
                counter -= ratio;
            }
        }
    }
    else
    {
        double ratio = std::abs(double(delta_r)) / std::abs(double(delta_l));
```

```
            double counter = 0;
            while(this->r.current_step != steps_right)
            {
                r.step(delta_r);
                counter++;

                if (counter > ratio)
                {
                    l.step(delta_l, 0);
                    counter -= ratio;
                }
            }
        }
    }

    Coord_mm plotter::get_position_mm()
    {
        double S1 = steps2mm(this->l.current_step);
        double S2 = steps2mm(this->r.current_step);
        double b = this->width;
        double x = (b * b + S1 * S1 - S2 * S2) / (2 * b);
        double y = sqrt(S1 * S1 - x * x);

        //std::cout << x << " x mm\n" << y << " y mm\n";
        return {x, y};
    }


    void plotter::draw_bbf(std::string path, double offset_x = 0, double offset_y = 0,
int start_line = 0)
    {
        auto start = std::chrono::steady_clock::now();
        std::cout << "draw " << path << "\n"
                  << "x offset: " << offset_x << "\n"
                  << "y offset: " << offset_y << "\n"
                  << "starting at Line " << start_line << "\n";

        std::ifstream bbf(path); // bbf Bruno Bashi Format
        int g;
        double x, y;
        unsigned int current_line = 0;
        bool run = true;

        // check if bbf and offset conf is in boundaries
        std::cout << "checking for boundarie issues\n";
        while (bbf >> g >> x >> y)
        {
            x += offset_x;
            y += offset_y;
            if(x > this->width - 5 | x < 5)
            {
                std::cout << "Error: Objekt überschreitet X Grenzen\n";
                run = false;
                break;
            }
            if(y < width / 4)
            {
```

```
                    std::cout << "Error: Objekt zu weit oben\n";
                    run = false;
                    break;
                }
                current_line++;
            }

    bbf.clear();
            bbf.seekg(0);
            current_line = 0;
            if(run)
            {
                while (bbf >> g >> x >> y)
                {
                    if(current_line >= start_line)
                    {
                        x += offset_x;
                        y += offset_y;

                        std::cout << "line: " << current_line << " ";
                        if(current_line == start_line) g = 0;
                        this->go2(x, y, g);
                    }
                    current_line++;
                }

                auto end = std::chrono::steady_clock::now();

                std::cout << "Elapsed time in seconds : "
                          << std::chrono::duration_cast<std::chrono::seconds>(end -
start).count() << " sec\n";
            }
            else{
                std::cout << "Error: mission abborted due to boundary issues\n";
            }
            bbf.close();
        }


        stepper_motor right (19, 16, 26, 20);
        stepper_motor left (17, 18, 22, 23);
        plotter pltr (left, right);


        void sighandler(int sig)
        {
            std::cout << "\nProgram abgebrochen vom Nutzer" << std::endl;

            pltr.end();
            pwmWrite(servo_pin, 165);

            exit(3);
        }


        int main(int argc, char *argv[])
        {
```

```
signal(SIGABRT, &sighandler);
signal(SIGTERM, &sighandler);
signal(SIGINT, &sighandler);

wiringPiSetupGpio(); // Initalize Pi GPIO
printf("Hello world!\n");

pltr.setup();

pinMode (servo_pin, PWM_OUTPUT) ;
pwmSetMode (PWM_MODE_MS);
pwmSetRange (2000);
pwmSetClock (192);

std::cout << steps2mm(left.current_step) << " S1 mm\n";
std::cout << steps2mm(right.current_step) << " S2 mm\n";

Coord_mm pos = pltr.get_position_mm();
std::cout << "x: " << pos.x << "\n";
std::cout << "y: " << pos.y << "\n";


if (argc == 1)
{
    std::cout << "Error: Keine Datei ausgewählt\n";
    exit(3);
}
else if (argc == 2)
{
    const std::string path = argv[1] + std::string(".bbf");
    pltr.draw_bbf(path);
}
else if (argc == 3)
{
    const std::string path = argv[1] + std::string(".bbf");
    double x_offset = std::strtod(argv[2], nullptr);
    pltr.draw_bbf(path, x_offset);
}
else if (argc == 4)
{
    const std::string path = argv[1] + std::string(".bbf");
    double x_offset = std::strtod(argv[2], nullptr);
    double y_offset = std::strtod(argv[3], nullptr);
    pltr.draw_bbf(path, x_offset, y_offset);
}
else if (argc == 5)
{
    const std::string path = argv[1] + std::string(".bbf");
    double x_offset = std::strtod(argv[2], nullptr);
    double y_offset = std::strtod(argv[3], nullptr);
                                                unsigned int start_line =
                                          std::atoi(argv[4]);
    pltr.draw_bbf(path, x_offset, y_offset, start_line);
}
else if (argc > 5)
{
    std::cout << "Error: What do all your parameters even mean?\n";
```

```
                exit(3);
            }


            pltr.end();
        }
```

Auszüge des SVG Parser Codes

```
        //aus Quelldatei svgHandling.cpp
        void read::evaluate_svg(std::string_view svg_view, double board_width, double
board_height)
        {
                la::Transform_Matrix to_board = la::in_matrix_order(1, 0, 0,
                                                                    0, 1, 0);
                const std::string_view view_box_data = read::get_attribute_data(svg_view,
"viewBox=");
                if (view_box_data != "") {
                        to_board = View_Box::set(view_box_data, board_width,
board_height);
                }
                else {
                        const double width =
read::to_scaled(read::get_attribute_data(svg_view, "width="));
                        const double height =
read::to_scaled(read::get_attribute_data(svg_view, "height="));
                        to_board = View_Box::set(width, height, board_width,
board_height);
                }

                read::evaluate_fragment(svg_view, to_board);
        }

        //aus Quelldatei svgHandling.cpp
        void read::evaluate_fragment(std::string_view fragment, const la::Transform_Matrix&
transform)
        {
                Elem_Data next = take_next_elem(fragment);      //function also removes
prefix belonging to next
                while (next.type != Elem_Type::end) {
                        switch (next.type) {
                        case Elem_Type::svg:
                                {
                                        const std::size_t nested_svg_end =
find_closing_elem(fragment, { "<svg" }, { "</svg>" });
                                        const std::string_view nested_svg_fragment = {
fragment.data(), nested_svg_end };

                                        const double x_offset =
to_scaled(get_attribute_data(next.content, { "x=" }), 0.0);
                                        const double y_offset =
to_scaled(get_attribute_data(next.content, { "y=" }), 0.0);
                                        const la::Transform_Matrix nested_matrix =
transform * la::translate({ x_offset, y_offset });
```

```
                                        evaluate_fragment(nested_svg_fragment,
nested_matrix);
                                        fragment.remove_prefix(nested_svg_end +
std::strlen("</svg>"));
                                }
                                break;

                        case Elem_Type::g:
                                {
                                        const std::size_t group_end =
find_closing_elem(fragment, { "<g" }, { "</g>" });
                                        const std::string_view group_fragment = {
fragment.data(), group_end };

                                        const la::Transform_Matrix group_matrix =
transform * get_transform_matrix(next.content);

                                        evaluate_fragment(group_fragment,
group_matrix);
                                        fragment.remove_prefix(group_end +
std::strlen("</g>"));
                                }
                                break;

                        case Elem_Type::line:                draw::line(transform,
next.content);              break;
                        case Elem_Type::polyline:    draw::polyline(transform,
next.content);     break;
                        case Elem_Type::polygon:     draw::polygon(transform,
next.content);              break;
                        case Elem_Type::rect:                draw::rect(transform,
next.content);              break;
                        case Elem_Type::ellipse:     draw::ellipse(transform,
next.content);              break;
                        case Elem_Type::circle:              draw::circle(transform,
next.content);              break;
                        case Elem_Type::path:                draw::path(transform,
next.content);              break;

                        case Elem_Type::unknown: break;       //just read in the next
element and do nothing
                        }
                        next = take_next_elem(fragment);      //function also removes
prefix belonging to next
                }
        }


        //aus Quelldatei linearAlgebra.hpp und Quelldatei linearAlgebra.cpp
        struct Transform_Matrix
        {
                double a, b, c, d, e, f;
                //corresponds to Matrix
                // a c e
                // b d f
                // 0 0 1
```

```
                //as specified here:
https://www.w3.org/TR/SVG11/coords.html#TransformMatrixDefined
        };

        //matrix * matrix, as in math
        constexpr Transform_Matrix operator*(const Transform_Matrix& fst, const
Transform_Matrix& snd)
        {
                const double& A = fst.a, B = fst.b, C = fst.c, D = fst.d, E = fst.e, F =
fst.f,
                              a = snd.a, b = snd.b, c = snd.c, d = snd.d, e = snd.e, f =
snd.f;

                return in_matrix_order(A * a + C * b, A * c + C * d, E + C * f + A * e,
                                       B * a + D * b, B * c + D * d, F + D * f + B * e);
        }

        Board_Vec operator*(const Transform_Matrix& matrix, Vec2D vec)
        {
                const double& a = matrix.a, b = matrix.b, c = matrix.c, d = matrix.d, e =
matrix.e, f = matrix.f,
                              x = vec.x, y = vec.y;

                return Board_Vec(a * x + c * y + e,
                                 b * x + d * y + f);
        }



        //aus Quelldatei linearAlgebra.hpp
        struct Vec2D
        {
                double x;
                double y;

                //constant to indicate a Vec2D has not yet been set
                static const Vec2D no_value;
        };

        //all operators are intended to work as in math.
        constexpr Vec2D operator+(Vec2D a, Vec2D b) { return { a.x + b.x, a.y + b.y }; }
        constexpr Vec2D operator-(Vec2D a, Vec2D b) { return { a.x - b.x, a.y - b.y }; }
        constexpr Vec2D operator-(Vec2D a) { return { -a.x, -a.y }; }
        constexpr Vec2D operator*(double factor, Vec2D vec) { return { factor * vec.x,
factor * vec.y }; }
        constexpr bool operator==(Vec2D a, Vec2D b) { return (a.x == b.x && a.y == b.y); }

        //exactly analogous to Vec2D, but different type to make sure,
        //only transformed vectors are drawn
        struct Board_Vec
        {
                double x;
                double y;

                explicit Board_Vec(double x_, double y_);       //do not allow brace
enclosed initialisation for Board_Vec
        };
```

```
        //aus Quelldatei svgHandling.cpp
        void draw::circle(la::Transform_Matrix transform_matrix, std::string_view
parameters, std::size_t resolution)
        {
                transform_matrix = transform_matrix *
read::get_transform_matrix(parameters);

                const double cx = read::to_scaled(read::get_attribute_data(parameters, {
"cx=" }), 0.0);
                const double cy = read::to_scaled(read::get_attribute_data(parameters, {
"cy=" }), 0.0);
                const double r  = read::to_scaled(read::get_attribute_data(parameters, {
"r=" }), 0.0);

                save_go_to(transform_matrix * (la::Vec2D{ cx, cy } +la::Vec2D{ r, 0.0 }));
        //intersection of positive x-axis and circle is starting point
                for (std::size_t step = 1; step <= resolution; step++) {
                        const double angle = 2 * la::pi * (resolution - step) /
static_cast<double>(resolution);
                        save_draw_to(transform_matrix * la::Vec2D{ cx + std::cos(angle) *
r, cy + std::sin(angle) * r });
                }
        }
```

Eidesstattliche Erklärung

Sebastian Meyer

Matrikelnummer: 3030618

Ich erkläre hiermit an Eides statt, dass − die vorliegende Arbeit – bei einer Gruppenarbeit den entsprechend gekennzeichneten Teil der Arbeit – selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, − alle Stellen der Arbeit, die ich wortwörtlich oder sinngemäß aus anderen Quellen übernommen habe, als solche kenntlich gemacht habe und − die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

31.3.2020

Bruno Borchardt

Matrikelnummer: 51316

Ich erkläre hiermit an Eides statt, dass − die vorliegende Arbeit – bei einer Gruppenarbeit den entsprechend gekennzeichneten Teil der Arbeit – selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, − alle Stellen der Arbeit, die ich wortwörtlich oder sinngemäß aus anderen Quellen übernommen habe, als solche kenntlich gemacht habe und − die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

31.3.2020