Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Computergestützte Modellierung und Simulation
Prof. Dr.-Ing. André Borrmann

Interaktiver Achsenentwurf

Approximation mit Hilfe einer B-Splinekurve

Christoph Kaiser

Bachelorthesis
für den Bachelor of Science Studiengang Bauingenieurwesen

Autor: Christoph Kaiser
Matrikelnummer: [redacted]
Betreuer: Štefan Jaud, M.Sc.

Ausgabedatum: 11. Mai 2020
Abgabedatum: 11. Oktober 2020

Zusammenfassung

Im Lauf des vergangenen Jahrzehnts haben sich Geräte mit großformatigen Touch-Oberflächen stark im alltäglichen Leben etabliert. Dennoch finden sie insbesondere im Bereich der Planung von Trassierungen der Infrastruktur noch kaum Anwendung.
Diese Arbeit beschäftigt sich daher mit der Entwicklung eines Algorithmus und dessen Implementierung zum interaktiven Zeichnen von Achsverläufe der Infrastruktur, beispielsweise einer Starßenplanung. Aufgrund der Störeinflüsse, welche beim freihändigen Zeichnen auf einem Touch-Gerät auftreten, erfolgt im vorgeschlagenen Algorithmus eine Filterung mithilfe einer B-Splinekurve. Davon abgeleitete Werte der Krümmung und Krümmungsänderung bilden die Grundlage zur Identifikation der Trassierungselement, welche sich als Gerade, Kreisbogen und Übergangskurve definieren.

Abstract

Over the past decade, devices with large-format touch surfaces have become firmly established in everyday life. However, especially in the planning of infrastructure routes, they are still rarely used.
This work therefore deals with the development of an algorithm and its implementation for the interactive drawing of axis courses of the infrastructure, for example street planning. Due to the interference that occurs when drawing freehand on a touch device, the proposed algorithm uses a B-spline curve to filter. Values of curvature and change of curvature derived from this form the basis for identifying the alignment elements, which are defined as straight lines, circular arcs and transition curves.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CAD	Computer Aided Design
DB	Deutsche Bahn
GPS	Global Positioning System
IFC	Industry Foundation Classes
LGS	lineares Gleichungssystem
OIP	TUM Open Infra Platform
RAA	Richtlinien für die Anlage von Autobahnen
RAL	Richtlinien für die Anlage von Landstraßen
RASt	Richtlinien für die Anlage von Stadtstraßen

Kapitel 1

Einleitung

1.1 Motivation

Im Lauf des vergangen Jahrzehnts haben sich Geräte mit großformatigen Touch-Oberfläche stark im alltäglichen Leben etabliert. Entsprechen der guten Verfügbarkeit solcher Geräte ist auch die Zahl der dazugehörigen Anwendungen enorm. Diese profitieren von den neuen Eingabeoptionen, welche nicht mehr indirekt über Maus und Tastatur erfolgen, sondern direkt. Dadurch ist es möglich, seine Gedanken unmittelbar im Eingabegerät grafisch umzusetzen.

Selbst in den Bereich der technischen und konstruktiven Anwendungen dringt diese Technologie inzwischen vor. Dennoch wird gerade im Bereich der Infrastrukturplanung immer noch oft und viel manuell in althergebrachter Weise in den CAD-Programmen konstruiert (Coetzee, 2019). Hierbei werden zunächst meist Skizzen auf Papier angefertigt. Die Varianten, welche dem Designteam am besten zusagen, werden schließlich in Computer Aided Design (CAD) konstruiert – mit unter können dies jedoch bis zu fünf verschiedene Versionen sein, wovon am Ende des Planungsprozesses nur eine übrig bleibt. Folglich ist hier der Personal- und Ressourceneinsatz sehr groß.

Entsprechend besteht hier ein umfangreiches Optimierungspotenzial, diese drei Komponenten miteinander zu verbinden: Direktes Zeichnen per Hand, Nutzung der verfügbaren Touch-Eingabe, und die unmittelbare Erzeugung von CAD-Daten. Dadurch kann das bislang zeitaufwendige manuelle Konstruieren von Trassierungselementen in CAD-Systemen vermieden werden.

1.2 Stand der Forschung

Im Forschungsbereich der Skizzenerkennung existieren eine Vielzahl von Arbeiten, welche sich mit diesem Themenkomplex auseinandersetzten. So gibt Cruz & Velho (2010) beispielsweise einen generellen Überblick über die Teilprozesse der Skizzenerkennung und geht dabei auf spezifische Störquellen ein. Han *et al.* (2005) hingegen stellt ein Verfahren vor, bei dem als Approximation der Eingabepunkte neue Kontrollpunkten einer B-Splinekurve berechnet werden. Diese Arbeiten haben jedoch gemein, dass sie nicht auf die Erkennung von Trassierungen der Infrastruktur spezialisiert sind.

Diese Spezialisierung ist jedoch beispielsweise in der Arbeit von Coetzee (2019) enthalten, welche eine freihändige Touch-Eingabe in eine Abfolge von Trassierungselementen konvertiert. Coetzee stellte dabei in ihrer Schlussfolgerung jedoch selbst fest, dass die Abfolge der Segmente keine C^2-Kontinuität aufweist; das heißt, dass Sprünge im Krümmungsverlauf auftreten. Eine weitere Arbeit in diesem Segment stammt von Gikas & Stratakos (2012), in welcher die aufgezeichneten Daten eines Global Positioning System (GPS)-Empfängers zu Trassierungsinformationen umgewandelt werden. Aufgrund seiner Datenquelle beschäftigt er sich jedoch nicht mit der Erkennung von Skizzen und den dort auftretenden Charakteristiken

1.3 Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung eines Algorithmus zur computergestützten Konstruktion eines Achsenentwurfs mit kontinuierlicher Krümmung, basierend auf einer digitalisierten Handskizze.

Eine Folge von Punkten soll mit Hilfe einer B-Spline approximiert und das Resultat in eine technisch korrekte Abfolge von Trassierungselementen konvertiert werden.

Kapitel 2

Grundlagen

Hauptgegenstand dieser Arbeit sind Trassierungen der Infrastruktur, mit Straßen und Eisenbahnen insbesondere aus dem Bereich des Verkehrswegebaus. Eine Einführung in diese mit einer kurzen Definition der wichtigsten Elemente und Eigenschaften wird in Kapitel 2.1 gegeben. Aus Kapitel 2.2 können die Grundlagen zur B-Splinekurve entnommen werden, welche hier zum Approximieren der grafischen Eingabedaten verwendet wird. Das Kapitel der Grundlagen schließt mit Teilabschnitt 2.3, wo einer Vorstellung des Gleitenden Mittelwerts unter Einbeziehung des Semivariogramms erfolgt; diese Technik findet später Anwendung zum Filtern respektive Glätten der Zwischenergebnisse.

2.1 Trassierung

Die Trassierung beschreibt den Verlauf von linearen Bauwerksstrukturen der Infrastruktur bezüglich ihrer Lage im dreidimensionalen Raum; als Linie stellt sie dabei die größtmögliche Abstraktion dar (Amann *et al.*, 2014). Solche Bauwerksstrukturen sind beispielsweise Straßen und Eisenbahntrassen.

Als Randbedingung der Planung sind topographische Zwangspunkte zu beachteten, dies sind beispielsweise Siedlungen, Naturschutzgebiete oder Bereiche schlechter Bodeneigenschaften (Freudenstein, 2018). Diese sollten in der Regel vermieden und mit Abstand umplant werden; entsprechend müssen sie im Lageplan - also der Horizontalen - berücksichtigt werden. Weiter soll die Herstellung der Bauwerke hinsichtlich möglicher Erdbauarbeiten kostengünstig erfolgen (Freudenstein, 2018). Hierzu ist eine enge vertikale Anpassung an das Gelände erforderlich.

Dementsprechend wird die dreidimensionale Kurve der Trassierung beschrieben durch die Überlagerung einer horizontalen Trassierung mit einer vertikalen Trassierung; dies sind jeweils zweidimensionale Kurven (Amann *et al.*, 2014). Die horizontale Trassierung beschreibt dabei

die Positionierung in der xy-Ebene und besteht in der Regel aus geraden Liniensegmenten, Kreisbögen und Übergangskurven. Die Höhe z entlang der Kilometrierung s wird durch die vertikale Trassierung definiert, diese setzt sich meist aus geraden Linien und parabolischen Kreisbögen zusammen (Amann *et al.*, 2014). In Abbildung 2.1 visualisiert Wijnholts (2016) die Überlagerung der beiden Kurventypen sehr anschaulich. Die untere breite Linie stellt dabei die horizontale Trassierung dar, exemplarisch mit einer Geraden, einer Klothoide als Übergangsbogen und einem Kreisbogen (von links). Als Abwicklung entlang der horizontalen Trassierung repräsentiert die obere breite schwarze Linie den vertikalen Verlauf der Trasse. Betrachtet man diese Linie nicht mehr in der Ebene der Abwicklung, sondern bezüglich eines dreidimensionalen Koordinatensystems, so wird die dreidimensionale Kurve der Trassierung deutlich.

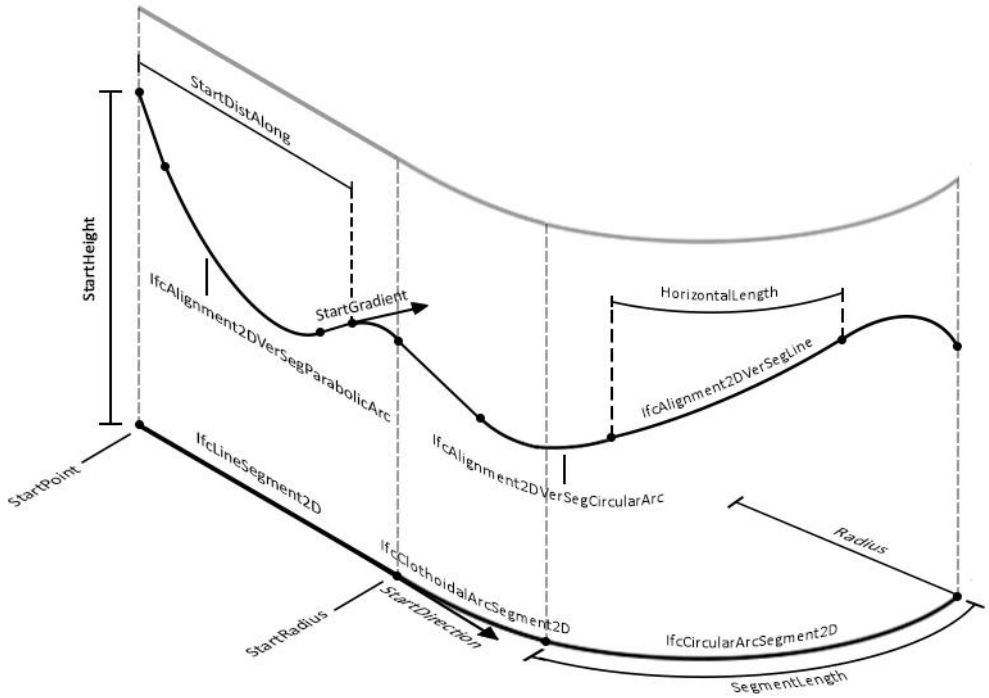

Abbildung 2.1: Überlagerung der horizontalen und vertikalen Trassierung. Abbildung übernommen aus Wijnholts (2016).

2.1.1 Trassierungselemente

Nachfolgend werden die oben genannten Elemente der horizontalen Trassierung näher definiert. Mit Blick auf die Eingrenzung dieser Arbeit wird auf eine detailliertere Beschreibung der vertikalen Trassierung verzichtet.

Alle Elemente können nach buildingSmart International (2020b) mit den Eigenschaften *StartPoint*, *StartDirection* und *SegmentLength* beschrieben werden. *StartPoint* – al-

so Startpunkt – ist ein Punkt mit kartesischen x- und y-Koordinaten. Die Startrichtung *StartDirection* ist ein Winkel gemäß mathematischer Definition, 0° blickt dabei in Richtung der positiven x-Achse. Die Kurvenlänge des Elements ist in *SegmentLength* abgelegt. Die Elementtypen Kreisbogen und Übergangsbogen werden nachfolgend mit weiteren Eigenschaften definiert.

Gerade

Geraden erscheinen zunächst als simples Trassierungselement, welches zwei Punkt auf kürzester Strecke verbindet (Schlenger, 2018). Bei Straßen sollten sie jedoch vermieden werden, unter Einhaltung enger Grenzwerte sind sie jedoch in einigen Fällen möglich. Weiter ist aus Schlenger (2018) zu entnehmen, dass sie hingegen bei Bahnstrecken das optimale Trassierungselement sind. Zu den oben für alle Elementtypen genannten Eigenschaften sieht (buildingSmart International, 2020c) für Geraden keine weiteren von.

Kreisbogen

Zur Richtungsänderung werden Kreisbögen verwendet, womit die oben genannten Zwangspunkte umplant werden können. Ebenso ist eine bessere Anpassung an den Verlauf des Geländes möglich (Schlenger, 2018). Die Definition des Kreisbogens wird in buildingSmart International (2020a) durch die Attribute *Radius* und $IsCCW$ ergänzt. $IsCCW$ steht dabei für „is counter-clockwise“ und gibt an, ob der Kreisbogen bezüglich der Kilometrierung eine Linkskurve (counter-clockwise) oder eine Rechtskurve (clockwise) beschreibt.

Übergangskurve

Die bereits vorgestellten Elemente Gerade und Kreisbogen weisen beide über ihre Elementlänge hinweg einen konstanten Wert des Radius R auf, wobei für die Gerade unendlich angenommen wird. Nach Kobryń (2017) ist jedoch auch die Krümmung κ eine wichtige Kenngröße, welche wie folgt berechnet wird:

$$\kappa = \frac{1}{R} \tag{2.1}$$

Weiterführend wird eine kontinuierliche Änderung der Krümmung, respektive des Radius, gefordert. Dies führt zu einer sich ebenfalls allmählich ändernden Zentrifugalkraft und verhindert somit einen plötzlichen Querruck (Kobryń, 2017). Im Fall des Straßenverkehrs erhöht sich laut Kobryń (2017) zudem die Verkehrssicherheit, da die Bewegung des Lenkrads in einer Übergangskurve als kontinuierlichen Bewegung erfolgen kann.

Die Klothoide, auch bekannt als Spiralkurve, erfüllt diese Bedingung und ist einer der wichtigsten Übergangskurven, welche durch die sogenannte natürliche Gleichung

$$A^2 = R \cdot L \tag{2.2}$$

beschrieben wird (Kobryń, 2017). A ist der Klothoidenparameter, R der Radius an einem Auswertepunkt und L die Kurvenlänge vom Ursprung bis zu diesem Auswertepunkt. Abbildung 2.2 zeigt eine vollständige Spiralkurve im ersten und dritten Quadranten des Koordinatensystems. Die Spiralen könnten in ihrem Inneren noch weiter verlängert werden, bis sie sich dem gekennzeichneten Mittelpunkt annähern.

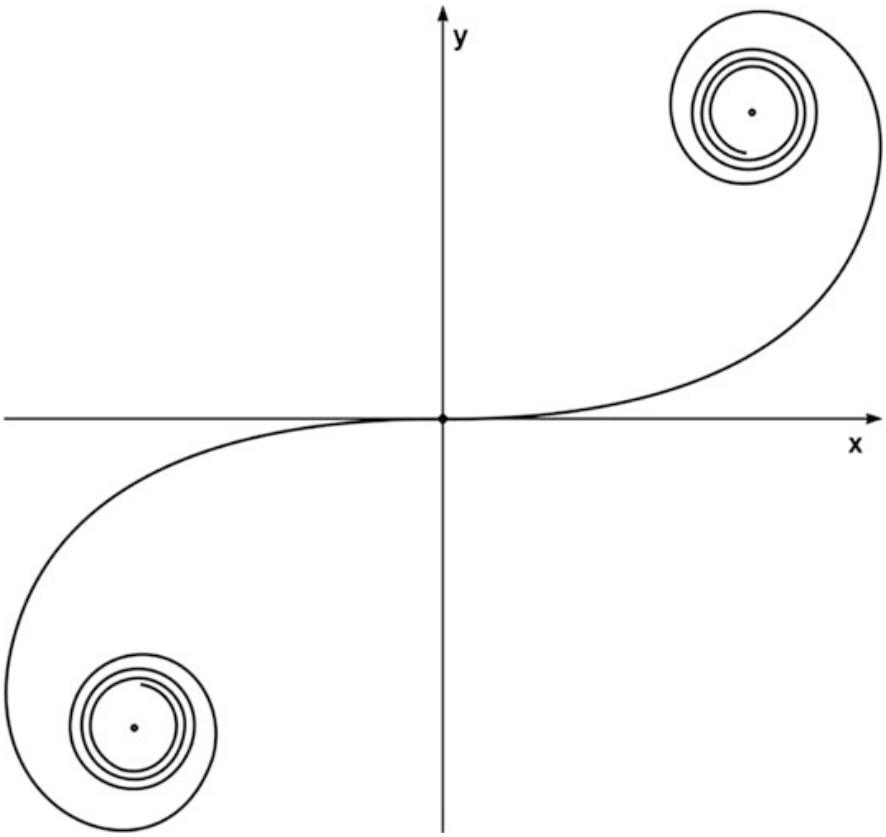

Abbildung 2.2: Vollständige Spiralkurve. Als Übergangsbogen wird im Weiteren lediglich ein Teil im ersten Quadranten benutzt. Abbildung übernommen aus Kobryń (2017).

Im konkreten Anwendungsfall des Übergangsbogens im Verkehrswegebau wird von der Spiralkurve jedoch nur ein kleiner Abschnitt im ersten Quadrant verwendet. Mathematisch kann dieser als Kurve $c(t)$ mit Hilfe des Fresnelintegrals

$$c(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \int_0^t \cos\frac{\pi}{2}u^2 du \\ \int_0^t \sin\frac{\pi}{2}u^2 du \end{pmatrix} \tag{2.3}$$

beschrieben werden (Meek & Walton, 1992). Eine anwendungsspezifische Auswertung bzw. Näherung des Integrals als Reihenentwicklung gibt Freudenstein (2018) an:

$$c(t) = \begin{pmatrix} t \cdot \left(1 - \frac{t^4}{40 \cdot A^4} + \frac{t^8}{3.456 \cdot A^8} - \frac{t^{12}}{599.040 \cdot A^{12}} + \frac{t^{16}}{175.472.640 \cdot A^{16}}\right) \\ t \cdot \left(\frac{t^2}{6 \cdot A^2} - \frac{t^6}{336 \cdot A^6} + \frac{t^{10}}{42.240 \cdot A^{10}} - \frac{t^{14}}{9.676.800 \cdot A^{14}}\right) \end{pmatrix} \tag{2.4}$$

Die in Freudenstein (2018) genannte Substitution $l = \frac{L}{A}$ wurde hier bereits als $l = \frac{t}{A}$ angewendet, damit ist Gleichung 2.4 auch äquivalent zu den Angaben in Kobryń (2017). A ist wiederum der Klothoidenparameter; t ist eine Laufvariable im Intervall $[0, L]$, wobei L die Kurvenlänge am Ende der Klothoide darstellt. Die Reihenentwicklung der Auswertung könnte zwar noch fortgesetzt werden, gemäß Freudenstein (2018) erreicht die gegebene Gleichung jedoch bereits eine ausreichend hohe Genauigkeit für die Anwendung im Verkehrswegebau.

Abbildung 2.3 (Freudenstein, 2018) zeigt den relevanten Ausschnitt der Klothoide im ersten Quadranten. Die Klothoide erstreckt sich vom Ursprung bis zum Punkt P; ebenfalls ist der anschließende Kreisbogen sowie relevante Bemaßungen und Bezeichnungen dargestellt. Neben den bereits genannten Größen hat auch der Winkel τ eine große Bedeutung, welcher die Richtung der Tangente am Klothoidenende P bezüglich der x-Achse angibt. Nach Kobryń (2017) wird er im Bogenmaß unter anderem wie folgt berechnet:

$$\tau = \frac{L}{2 \cdot R} \tag{2.5}$$

L bezeichnet die Kurvenlänge, an welcher der Radius R vorzufinden ist.

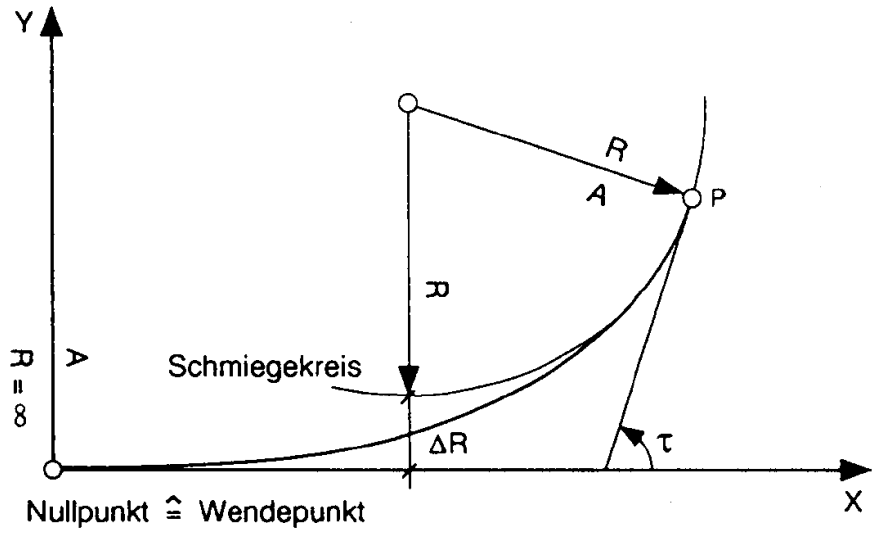

Abbildung 2.3: Klothoide im verwendeten Bereich mit Bezeichnung relevanter Abstände und Winkel. Abbildung übernommen aus Freudenstein (2018).

Um die Übergangskurve zu beschreiben, sieht buildingSmart International (2020d) zu den für alle Trassierungselemente oben genannten Eigenschaften zusätzlich die Parameter $StartRadius$, $EndRadius$, $IsStartRadiusCCW$, $IsEndRadiusCCW$ und $TransitionCurveType$ vor. Wird für den anschließenden Start- oder Endradius kein Zahlenwert übergeben, wird der Radius unendlich angenommen – also eine Gerade. In gleicher Weise wie beim Kreisbogen geben die Parameter $IsStartRadiusCCW$ und $IsEndRadiusCCW$ an, ob sich der Radius nach links oder rechts entwickelt. In $TransitionCurveType$ wird der Typ der Übergangskurve angegeben, im Fall der hier gezeigten Klothoide $CLOTHOIDCURVE$.

2.1.2 Grenzwerte der Trassierung

Die oben vorgestellten Parameter der einzelnen Elemente werden in ihrem Wertebereich beschränkt von den Richtlinien für die Anlage von Autobahnen (RAA), den Richtlinien für die Anlage von Landstraßen (RAL) und den Richtlinien für die Anlage von Stadtstraßen (RASt), sowie für Schienenwege durch DIN EN 13803 (2017). Die im nachfolgenden Kapitel angegebenen Werte stammen aus den Schneider–Bautabellen (Albert, 2016). Hiervon werden ihm Rahmen des Programmentwurfs im Kapitel 4 Programmentwurf Schwellenwerte der Kurveninterpretation abgeleitet. Da der Schwerpunkt dieser Arbeit auf der Erkennung der horizontalen Trassierung liegt, wird im folgenden lediglich auf diese eingegangen; Grenzwerte der vertikalen Linienführung können gegebenen Falls aus den genannten Quellen entnommen werden. Zudem liegt der Fokus auf den jeweiligen Minimal- und Maximalwerten, weshalb hier nur eine grobe Zusammenfassung über die Gesamtheit der Grenzwerte beabsichtigt ist.

Gerade

Bei den Geraden von Straßen gelten als Richtwerte die minimalen und maximalen Längen nach Tabelle 2.1 (Albert, 2016). Zudem sind bei Wendeklothoiden Zwischengeraden mit eine Länge von $L_Z \leq 0{,}08 \cdot (A_1 + A_2)$ möglich, wobei A_1 und A_2 die zugehörigen Klothoidenparameter repräsentieren. Entsprechend des bei Übergangsbogen größten genannten Wert von $A = 300$ m (siehe unten) ergibt sich hierfür die maximale Längen von 48 m.

	Autobahnen	Landstraßen
maximale Länge	2000 m	1500 m
minimale Länge zwischen gleichsinnigen Kurven	400 m	600 m bei EKL 1 bis EKL 3 400 m bei EKL 4

Tabelle 2.1: Grenzwerte der Geradenlänge bei Straßen. Inhalt übernommen aus Schneider–Bautabellen (Albert, 2016)

Schienenwege der Deutsche Bahn (DB) werden laut Albert (2016) meist nur noch mit angestrebten Entwurfsgeschwindigkeiten v von 120 bis 300 km/h geplant. Bei Geraden ergeben sich aufgrund der für diese Geschwindigkeiten gegebenen Bedingung $l_g \geq 0{,}2 \cdot v$ (nach Mindestwerte) Längen von 24 bis 60 m. Die Bedingung für die primär angestrebten Regelwerte lautet jedoch $l_g \geq 0{,}4 \cdot v$, wodurch sich die Mindestlänge von 48 bis 120 m erstreckt. Maximallängen sind nicht angegeben.

Kreisbogen

Für Kreisbögen der Autobahnen sieht die RAA Mindestradien in Abhängigkeit der Entwurfsklasse von 280 bis 900 m vor; bei Geraden von mehr als 500 m Länge sollen die angrenzenden Bögen sogar einen Mindestradius von 1300 m aufweisen (Albert, 2016).
Zudem müssen die Kreisbögen eine Mindestlängen von 55 bis 75 m haben. Maximalbeschränkungen bei Radius und Länge sind nicht genannt.

Aus den Schneider–Bautabellen ist für Landstraßen gemäß RAL ein Mindestradius von 200 bis 500 m zu entnehmen, abhängig von der Entwurfsklasse. Maximale Radien erstrecken sich von 400 bis 900 m, wobei die Entwurfsklasse EKL 1 keine Beschränkung nach oben hat.
Die Mindestlängen erstrecken sich von 40 bis 70 m.

Bei anbaufreien Hauptverkehrsstraßen ist nach RASt lediglich ein Mindestradius von 80 bzw. 190 m genannt, abhängig von der zugelassenen Höchstgeschwindigkeit (Albert, 2016). Weiter schreibt Albert, dass bei angebauten Hauptverkehrsstraßen diese Mindestradien nicht gelten.

Bei Schienenverkehrswegen der DB beträgt die Herstellungsgrenze einen Radius von 25.000 m (Albert, 2016). Weiter wird dort der Mindestradius auf 150 bis 500 m definiert, wobei dieser vom Anwendungsbereich abhängt; bei Straßenbahnen beträgt er sogar nur 25 m.
Die minimale Bogenlänge ist analog zur Geradenlänge definiert (siehe oben); entsprechend gelten auch hier Mindestlängen von 24 bis 60 m bzw. Längen von 48 bis 120 m für angestrebte Regelwerte.

Übergangskurve

In den Schneider–Bautabellen von Albert (2016) ist der allgemeinen Beschreibung der Wendeklothoide für Straßen zu entnehmen, dass die beiden Klothoidenparameter A_1 und A_2 die Bedingung $A_1 \leq 1{,}5 \cdot A_2$ erfüllen müssen, wobei $A_2 \leq 300$ m der kleinere Parameter ist. Weiterführend ist sogar wünschenswert, dass beide Parameter den gleichen Wert aufweisen.

Die RAA schreibt für Klothoiden als Übergangsbogen der Autobahn in Abhängigkeit der Entwurfsklasse den Mindestparameter A_{min} von 90 bis 300 m vor (Albert, 2016).

Zur Wahl des Klothoidenparameters A für Landstraßen schreibt die RAL die Bedingung $R/3 \leq A \leq R$ vor, wobei R dem Radius des anschließenden Kreisbogens entspricht (Albert, 2016).

Für anbaufreie Hauptverkehrsstraßen nach RASt ist in Abhängigkeit von der zugelassenen Höchstgeschwindigkeit aus den Schneider–Bautabellen der Klothoidenmindestparameter A_{min} von 50 bzw. 90 m zu entnehmen.

Die Länge des Übergangsbogens bei Schienenwegen ist abhängig von der zu erreichenden Überhöhung u [mm], welche bei fester Fahrbahn im Ermessensfall bis zu 170 mm beträgt, als auch von der vorgesehenen Entwurfsgeschwindigkeit v [km/h] (siehe oben). Bei einer geraden Überhöhungsrampe, welche in der Lage deckungsgleich mit dem Übergangsbogen liegt, ergibt sich nach der Formel

$$l_R[\mathrm{m}] = \frac{8 \cdot v[\mathrm{km/h}] \cdot u[\mathrm{mm}]}{1000} \tag{2.6}$$

eine Mindestlänge l_R von 163 bis 408 m (Albert, 2016).

2.2 B-Splinekurve

2.2.1 Definition der B-Splinekurve

Mathematische Kurven des Typs Spline sind stückweise Polynome (Prautzsch *et al.*, 2002). Nach Rogers (2001) definieren die Punkte P_i eines Kontrollpolygons die Kurve; mit Hilfe eines Interpolations- oder Approximationsschemas werden die Punkte und die Kurve miteinander in Beziehung gesetzt. Die B-Splinekurve ist dabei ein konkreter Fall der Splines, welche sich mit den B-Spline-Basisfunktionen $N_{i,k}$ als Approximationsschema wie folgt berechnet:

$$c(t) = \sum_{i=1}^{n+1} P_i \cdot N_{i,k}(t) \tag{2.7}$$

mit den rekursiv definierten Basisfunktionen:

$$N_{i,k}(t) = \frac{(t - x_i) \cdot N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t) \cdot N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \tag{2.8a}$$

$$\text{mit der Konvention } \frac{0}{0} = 0$$

$$N_{i,1}(t) = \begin{cases} 1 \text{ für } x_i \leq t < x_{i+1} \\ 0 \text{ sonst} \end{cases} \tag{2.8b}$$

Hierbei ist t der Kurvenparameter aus dem rechtsoffenen Intervall $[t_{\min};\ t_{\max})$. Das Kontrollpolygon wird durch $n+1$ Punkte beschrieben. k wird als Ordnung der B-Splinekurve bzw. derer Basisfunktionen bezeichnet. Für den Wert von k gilt: $k \leq n+1$. Die Ordnung unterscheidet sich vom Polynomgrad der Basisfunktionen (in Rogers als *degree* bezeichnet) durch die Vorschrift $degree = k-1$.

Für die Einträge x_i eines sogenannten Knotenvektors X gilt die Bedingung $x_i \leq x_{i+1}$. Ein Knotenvektor besteht immer aus $n+k+1$ Elemente. Die Vektoren werden unterschieden als gleichmäßig oder ungleichmäßig, sowie als periodisch oder offen.
Bei gleichmäßigen Knotenvektoren weisen alle Einträge den gleichen Abstand auf, davon ausgenommen sind ggf. abweichende Anfangs- und Endbedingungen des offenen Knotenvektors (siehe unten). Ein Beispiel hierfür stammt aus Rogers (2001): $X = (0;1;2;3;4)$. Bei ungleichmäßigen Knotenvektoren sind die Einträge entsprechend nicht gleichmäßig verteilt; ein eigenes Beispiel hierfür ist $X = (0;1;2{,}5;3;4)$.
Bei periodischen Knotenvektoren erhöht sich bereits der zweite Vektorwert, dies setzt sich bis zum letzten Wert fort (Rogers, 2001); das oben bereits gezeigte Beispiel $X = (0;1;2;3;4)$ ist daher ein periodischer gleichmäßiger Knotenvektor. Im Gegensatz dazu weisen bei offenen Knotenvektoren die ersten k Elemente einen identischen Wert auf; gleiches gilt auch für die letzten k Elemente. Für die Ordnung $k = 2$ zeigt Rogers den Vektor $X = (0;0;1;2;3;4;4)$ als Beispiel eines offenen gleichmäßigen Knotenvektors.

Unter den beiden Bedingungen, dass der offene gleichmäßige Knotenvektor mit dem Wert 0 beginnt, sowie die weiteren Werte ganzzahlig erhöht werden, ist aus Rogers (2001) folgende Bildungsvorschrift für die einzelnen Vektorwerte zu entnehmen:

$$x_i = 0 \qquad \text{wenn} \qquad 1 \leq i \leq k \tag{2.9a}$$

$$x_i = i - k \qquad \text{wenn} \qquad k+1 \leq i \leq n+1 \tag{2.9b}$$

$$x_i = n - k + 2 \qquad \text{wenn} \qquad n+2 \leq i \leq n+k+1 \tag{2.9c}$$

Die Werte t des oben bereits erwähnten Intervalls $[t_{\min};\ t_{\max})$ sind für periodische Knotenvektoren, welche mit dem Wert 0 starten und ganzzahlig hoch zählen, wie folgt definiert (Rogers, 2001):

$$k - 1 \leq t < n + 1 \tag{2.10}$$

Aufgrund dieser Vorschrift sind bei periodischen Knotenvektoren erhebliche Teile nahe der ersten und letzten Kontrollpunkte nicht Bestandteil der B-Splinekurve.
Für offene Knotenvektoren, welche nach den Gleichungen 2.9a, 2.9b und 2.9c gebildet wurden, wird der Intervall definiert als:

$$0 \leq t < n - k + 2 \tag{2.11}$$

Dadurch sind auch der erste und letzte Kontrollpunkt geometrischer Bestandteil der B-Splinekurve.

Abbildung 2.4 zeigt eine B-Splinekurve der Ordnung $k = 4$, welche auf Grundlage eines offenen gleichmäßigen Knotenvektors berechnet wurde. Die in der Abbildung dargestellten Punkte B_i entsprechen den im Text beschriebenen Kontrollpunkten P_i. In der Graphik wird deutlich, dass die Kurve grundsätzlich der Form des Kontrollpolygons folgt – insbesondere an den drei verschiedenen Lagen des Kontrollpunkts B_5 zu erkennen. (Rogers, 2001)

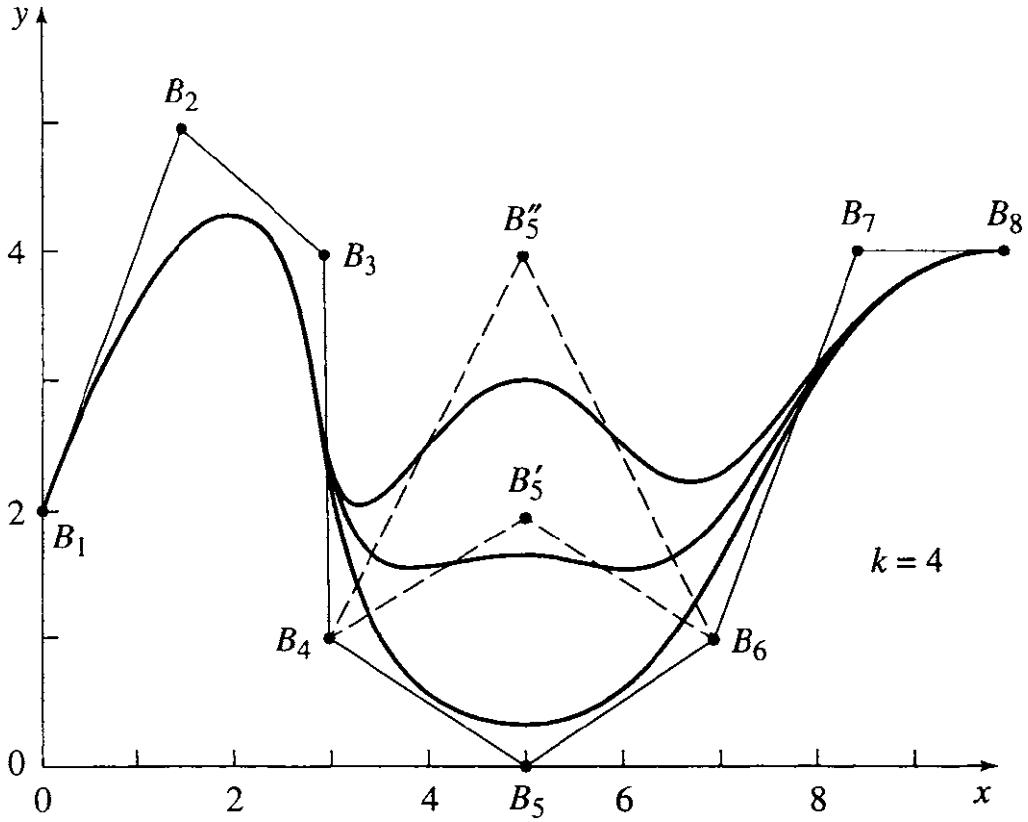

Abbildung 2.4: Beispiel einer B-Splinekurve. Veranschaulichung des Einflussbereichs des Kontrollpunks B_5. Die dargestellten Punkte B_i entsprechen den im Text beschriebenen Kontrollpunkten P_i. Abbildung übernommen aus Rogers (2001).

In diesem Zusammenhang wird auch der Einflussbereich eines Kontrollpunkts auf den Verlauf der Kurve deutlich. Nach Rogers (2001) wirkt sich ein Kontrollpunkt P_i auf den Kurvenverlauf zwischen den Punkten $P_{i-k/2}$ und $P_{i+k/2}$ aus. Aufgrund dieser Eigenschaft haben bei einer höheren Ordnung k auch eine größere Anzahl Kontrollpunkte Einfluss auf eine Stelle t der Kurve, dies äußert sich ein einem weicheren Verlauf der B-Splinekurve.

2.2.2 Ableitungen und Krümmung der B-Splinekurve

In Rogers (2001) wird die erste Ableitung der B-Splinekurve definiert als:

$$c'(t) = \sum_{i=1}^{n+1} P_i \cdot N'_{i,k}(t) \tag{2.12}$$

mit der ersten Ableitung der Basisfunktionen:

$$\begin{aligned} N'_{i,k}(t) = &\frac{N_{i,k-1}(t) + (t - x_i) \cdot N'_{i,k-1}(t)}{x_{i+k-1} - x_i} \\ &+ \frac{(x_{i+k} - t) \cdot N'_{i+1,k-1}(t) - N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \\ &\text{mit der Konvention } \tfrac{0}{0} = 0 \end{aligned} \tag{2.13a}$$

$$N'_{i,1}(t) = 0 \quad \text{für alle t} \tag{2.13b}$$

Weiterführend ist dort auch die zweite Ableitung zu entnehmen:

$$c''(t) = \sum_{i=1}^{n+1} P_i \cdot N''_{i,k}(t) \tag{2.14}$$

mit der zweiten Ableitung der Basisfunktionen:

$$\begin{aligned} N''_{i,k}(t) = &\frac{2 \cdot N'_{i,k-1}(t) + (t - x_i) \cdot N''_{i,k-1}(t)}{x_{i+k-1} - x_i} \\ &+ \frac{(x_{i+k} - t) \cdot N''_{i+1,k-1}(t) - 2 \cdot N'_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \\ &\text{mit der Konvention } \tfrac{0}{0} = 0 \end{aligned} \tag{2.15a}$$

$$N''_{i,1}(t) = N''_{i,2}(t) = 0 \quad \text{für alle t} \tag{2.15b}$$

Darüber hinaus ist in Bronstein *et al.* (2016) für parametrische Kurven des Typs $c(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$ eine Formel zur Berechnung der Krümmung gegeben:

$$K = \frac{\begin{vmatrix} x' & y' \\ x'' & y'' \end{vmatrix}}{(x'^2 + y'^2)^{3/2}}$$

Nach Anwendung der Determinante erhält man folgende Formel zur Berechnung der Krümmung:

$$K = \frac{x' \cdot y'' - x'' \cdot y'}{(x'^2 + y'^2)^{3/2}} \tag{2.16}$$

2.3 Gleitender Mittelwert mit Semivariogramm

Der gleitende Mittelwert wird als Filter zum Glätten von Rauschen in Datenerhebungen wie folgt definiert (Vamos & Craciun, 2012):

$$\vartheta_x(n) = \frac{1}{2 \cdot K + 1} \cdot \sum_{k=-K}^{K} x_{n+1} \tag{2.17}$$

mit der Stelle bzw. dem Index n in der Datenreihe x, den Werten x_{n+1} der Datenreihe und der halben Anzahl K der Punkte im Fenster des gleitenden Mittelwerts. Die hier gezeigte Formel ist bereits der Spezialfall des zentrierten gleitenden Mittelwerts, bei welchem auf beiden Seiten der betrachteten Stelle n jeweils K Werte in die Berechnung einfließen. Für Stellen, welche nahe am Anfang oder Ende der Datenreihe liegen, gibt Vamos & Craciun (2012) Formeln an, bei denen das Fenster einseitig verkleinert wird; beispielsweise für die erste Stelle eines eins-basierten Vektors erstreckt sich das Fenster über die Stellen 1 bis $1 + K$.

Zur dynamischen Bestimmung der Fenstergröße greift Gikas & Stratakos (2012) mit dem Semivariogramm ein Verfahren auf, welches fraktales Rauschen aus Messdaten entfernt, wobei die Daten eine Gaußsche Charakteristik aufweisen – entsprechend seiner gezeigten Testdaten also feine hochfrequente Störungen auf Daten, welche sich dennoch um einen Mittelwert herum anordnen. Hierzu wird die so genannte Semivariogrammvariable

$$\gamma(h) = \frac{1}{2 \cdot \sigma_a \cdot N} \cdot \sum_{i=1}^{N} [a(s_i) - a(s_i + h)]^2 \tag{2.18}$$

berechnet, wobei die Datenwerte bzw. hier die Krümmung a an den Stellen s_i respektive $s_i + h$, die Varianz σ_a und die Gesamtanzahl N der Datenwerte eingehen. Der Versatz h beschreibt, wie weit die jeweils betrachteten Werte a in der Datenreihe voneinander entfernt liegen. Wird die gegebene Formel für verschiedene Werte von h ausgewertet, ergibt sich ein geordneter Datensatz der Semivarianzen (Oliver & Webster, 2015).

Um diese auszuwerten, kann ein Diagramm in doppellogarithmischer Skalierung verwendet werden (Gikas & Stratakos, 2012); Abbildung 2.5 zeigt dieses in idealisierter Darstellung. Auf der horizontalen Achse „Correlation Length“ ist der Versatz h angetragen, auf der vertikalen Achse „Variance“ der dazugehörige Wert der Semivariogrammvariable $\gamma(h)$. Der Verlauf von $\gamma(h)$ ist zunächst linear steigend. Die charakteristische Stelle, an welcher der Übergang in einen konstanten Verlauf erfolgt, ist in der Abbildung als „Cross Over Length“ gekennzeichnet; allgemein wird in der Literatur diese Stelle auch als „Range“ bezeichnet, beispielsweise in Vamos & Craciun (2012). Der Wert der Semivariogrammvariable $\gamma(h)$, welcher sich ab dieser Stelle als konstant einstellt, wird als „Sill“ bezeichnet (Vamos & Craciun, 2012).

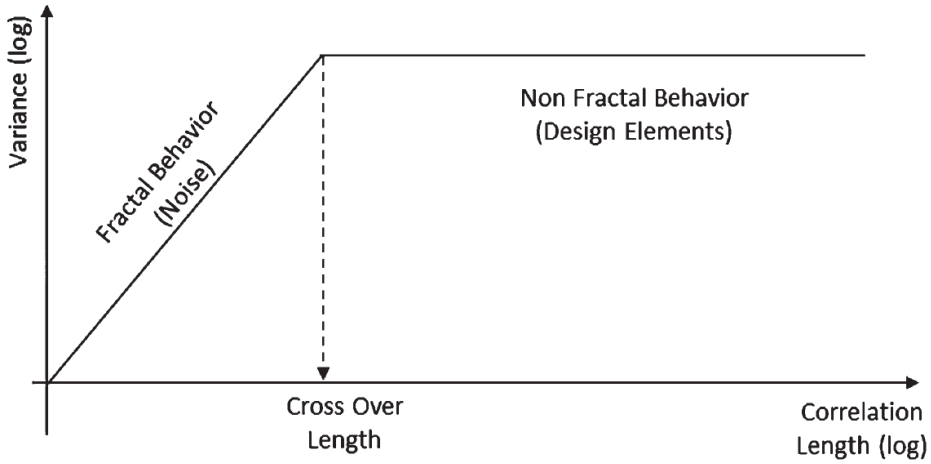

Abbildung 2.5: Semivariogramm in idealisierter Darstellung, mit Kennzeichnung der charakteristischen Stellen. Abbildung übernommen aus Gikas & Stratakos (2012).

Für Versatzgrößen h, welche größer sind als die Cross Over Length, hat das fraktale Rauschen kaum Einfluss auf den tatsächlichen Datenwert (Gikas & Stratakos, 2012). Darauf basierend wird festgestellt, dass die Cross Over Length für die gesamte Fenstergröße w des gleitenden Mittelwerts angesetzt werden kann.

Kapitel 3

Methodik

Kapitel 3.1 erläutert zunächst die Teilprozesse und Charakteristik einer grafischen Zeicheneingabe, welche beispielsweise auf einem Touchscreen oder an einem Beamer mit Berührungserkennung erfolgt. Um die dort beschriebenen Störungen zu kompensieren, wird in Kapitel 3.2 ein Verfahren zum Approximieren und Filtern der Eingabedaten mittels einer B-Splinekurve vorgestellt. Ebenso werden die Eigenschaften des für diese Anwendung gewählten B-Splinetyps aufgezeigt. Abschließend wird die Beurteilung der Krümmung mit Krümmungsänderung als Methodik der Elementidentifikation in Kapitel 3.3 vorgestellt.

3.1 Graphische Eingabe

Die grafische Eingabe einschließlich der Verarbeitung der Daten kann grundsätzlich in folgende Teilprozesse unterteilt werden (Cruz & Velho, 2010):

Zeichnen: Der Benutzer zeichnet physikalisch mit beispielsweise dem Finger oder einem Eingabestift auf einem Touchscreen.

Datenaufzeichnung: Das Gerät nimmt die Informationen des Zeichnens auf, für jeden Datenpunkt meist in festen Zeitintervallen die Position und den dazugehörigen Zeitpunkt, geräteabhängig manchmal auch weitere Informationen wie Anpressdruck oder Beschleunigung.

Pfadbildung: Die zeitliche Abfolge der einzelnen Datenaufzeichnungen kann als Pfad zusammengefasst werden und erhält dabei die Gestalt einer Punktfolge.

Modellierung: Die erhaltene Punktfolge wird zu einer Kurve verallgemeinert, welche oft durch eine mathematische Beschreibung abgebildet werden kann.

Interpretation: Die modellierten Kurven werden in Abhängigkeit der Anwendung gedeutet, beispielsweise für die Konstruktion geometrischer Objekte oder zur Steuerung des Programms.

Wie oben erwähnt, erfolgt die Datenaufzeichnung durch kontinuierliche Messungen in festen Zeitintervallen des Eingabegeräts, beispielsweise dem Touchscreen eines Tablets oder auf der Projektionsfläche eines Beamers mit Berührungserkennung. Nach Cruz & Velho (2010) sind im Allgemeinen dabei zwei Charakteristiken festzustellen: Infolge einer sich variierenden Zeichengeschwindigkeit liegen die erfassten Punkte nicht in gleichmäßigen geometrischen Abständen vor. Zudem ergibt sich bei einer sehr feinen Intervallgröße eine große Anzahl aufgenommener Punkte.

Um mit diesen beiden Eigenschaften akkurat umzugehen, erfolgt eine Filterung der aufgezeichneten Datenpunkte. Eine einfache Umsetzung eines solchen Filters ist es, lediglich Punkte in einem definierten Mindestabstand auszuwählen (Cruz & Velho, 2010). Damit wird zugleich das in Han *et al.* (2005) genannte Problem von zeitlich aufeinanderfolgend doppelt bzw. mehrfach aufgezeichneter Punktkoordinaten kompensiert. Auch in Abhängigkeit der weiteren Verwendung möglicherweise zu enge Abstände, wie sie in Han *et al.* (2005) erwähnt sind, können so ausgeschlossen werden.

Trotz der Filterung können jedoch immer noch zwei Arten von Störungen das Resultat beeinflussen. Die erste Art bezeichnet Cruz & Velho (2010) als Störung der Informationen. Sie tritt auf, wenn der Benutzer die gewünschte Linie nicht exakt zeichnet – beispielsweise wenn eine gerade Linie doch ein wenig krumm wird. Störungen des Geräts sind die zweite Art, welche laut Cruz & Velho (2010) vorkommen kann, wenn durch das Eingabegerät die Zeichenbewegung nicht präzise oder mit Abweichung aufgenommen wird.

3.2 Approximation der Eingabepunkte

Um die beiden am Ende des vorherigen Kapitels erläuterten Störungen kompensieren zu können, erfolgt hier eine Approximation mittels einer B-Splinekurve. Die vollständige Punktfolge der Eingabe fließt hierbei als Vektor der B-Spline-Kontrollpunkte ein. Die Approximation entspricht zugleich dem Teilprozess Modellierung der grafischen Eingabe, welcher zu Beginn von Kapitel 3.1 vorgestellt wurde.

Wie am Ende von Kapitel 2.2.1 gemäß Rogers (2001) dargestellt, haben durch eine höhere Ordnung k mehr Kontrollpunkte Einfluss auf eine betrachtete Stelle t der B-Splinekurve, wodurch der Verlauf weicher wird. Umgekehrt wirkt sich ein Kontrollpunkt weniger stark auf eine Stelle der Kurve aus. Dadurch wird hier die B-Splinekurve als Filter eingesetzt, welcher aus den Eingabepunkten „zittern“ und „welliges zeichnen“ kompensiert.

Als Ausschnitt einer längeren Punktfolge zeigt Abbildung 3.1 exemplarisch die Wirkung einer B-Splinekurve der Ordnung 20. Die Eingabepunkte sind mit Sternen gekennzeichnet und durch eine Strichlinie verbunden, die Auswertung der B-Splinekurve ist als Volllinie dargestellt. Dabei wird deutlich, wie die Abweichungen der Eingabepunkte ausgeglichen werden und sich somit eine kontinuierliche Biegung der Approximation ergibt. Ebenso ist die Eigenschaft der B-Splinekurve erkennbar, dass die Auswertung in Biegungen etwas in Richtung Zentrum versetzt – im Fall es konkreten Beispiels nach unten. Dieser Effekt wird jedoch schwächer, je höher die Dichte der Eingabepunkt ist, respektive je kleiner der Abstand zwischen ihnen ist.

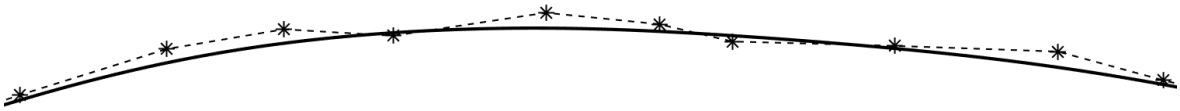

Abbildung 3.1: Ausschnitt einer B-Splinekurve (Volllinie) als Filter der Eingabepunkte (Sterne, mit Strichlinie verbunden).

Von den in Kapitel 2.2.1 vorgestellten Varianten des Knotenvektors wird für diese Anwendung der offene gleichmäßige Typ gewählt. Im Gegensatz zum periodischen Knotenvektor sind bei diesem auch der erste und letzte Punkt der grafischen Eingabe Bestandteil der ausgewerteten B-Splinekurve. Dies ist eine notwendige Bedingung, um die Benutzereingabe auf ihrer vollen Länge untersuchen zu können.

3.3 Identifikation der Trassierungselemente

Entsprechend der Darstellungen in Kapitel 2.1.1 können die Trassierungselemente anhand ihrer verschiedenen Radien respektive Krümmungen unterschieden werden. So besitzt eine Gerade den Krümmungswert 0, ein Kreisbogen einen konstanten Wert $\neq 0$ und bei der Klothoide entwickelt sich die Krümmung linear. Aufgrund der Approximation in Kapitel 3.2 wird die Punkteingabe bereits durch eine mathematische Kurve beschrieben. Wegen der analytisch definierten Ableitungen eignet sich diese besonders, um eine Auswertung der Krümmung zu berechnen. In Anlehnung an das von Gikas & Stratakos (2012) vorgestellte Verfahren wird auch in dieser Arbeit die Krümmung verwendet, um die Einzelnen Trassierungselemente zu identifizieren.

Aus eigenen empirischen Beobachtungen wurde festgestellt, dass die Krümmungswerte selbst mit der in Kapitel 3.2 erfolgten Approximation und Filterung zum Teil noch Spitzen und Unregelmäßigkeiten aufweisen. Um dies zu kompensieren, erfolgt eine erneute Glättung mit Hilfe des gleitenden Mittelwerts. Nach dem von Gikas & Stratakos (2012) beschriebenen Vorgehen wird die Fenstergröße des Mittelwerts mit dem in Kapitel 2.3 vorgestellten Semivariogramm bestimmt.

Um den Verlauf der Krümmung in seine charakteristischen Elemente (0, linear, konstant $\neq 0$) einteilen zu können, wird vorbereitend die Krümmungsänderung durch numerisches Ableiten ermittelt. Mit dieser können charakteristische Punkte bestimmt werden, welche als Abgrenzung zwischen jedem einzelnen Element vorzufinden sind. Bei der Krümmungsänderung erscheinen Geraden als auch Kreisbögen mit dem Wert 0, Klothoiden hingegen besitzen einen konstanten Wert $\neq 0$ (Gikas & Stratakos, 2012). Demnach entspricht jeder Wechseln der Krümmungsänderung zwischen 0 und $\neq 0$ einem charakteristischen Punkt und kann somit als Start- bzw. Endpunkt eines Trassierungselements definiert werden.

Nachdem somit die Trassierungselemente in ihre Position beschrieben sind, erfolgt für alle Elemente mit einer Krümmungsänderung von 0 durch Abschätzen der Krümmung die Bestimmung von Geraden und Kreisbögen einschließlich ihrer Eigenschaften. Um an den Übergängen von und zu Klothoiden einen glatten und ruckfreien Achsverlauf entsprechend Kapitel 2.1.1 zu gewährleisten, werden die Parameter der Klothoiden erst nach der Festlegung von Geraden und Kreisbögen berechnet; dadurch ist eine geometrisch perfekte Einpassung möglich.

Kapitel 4

Programmentwurf

Im Rahmen des Kapitels Programmentwurf erfolgt die detaillierte Entwicklung und Beschreibung des vorgeschlagenen Algorithmus. Abschnitt 4.1 stellt zunächst Forderungen an Qualität und Informationsumfang der Eingabedaten. Ebenso formuliert er eine kleine Nebenbedingung, welche die Möglichkeit zum Speichern und Laden der Eingabedaten beinhaltet. Weiter setzt Teilkapitel 4.2 die Parameter der Punktapproximation mittels der B-Splinekurve. Kapitel 4.3 bildet den Hauptteil des Programmentwurfs; hierzu teilt es sich in den Abschnitt 4.3.1, welcher das Vorgehen zum berechnen der Krümmung und Krümmungsänderung beschreibt, sowie in das Unterkapitel 4.3.2, welches die Schritte und Kriterien zum Identifizieren der Trassierungselemente einschließlich ihrer Parameter erläutert.

4.1 Eingabe der Punktfolge

4.1.1 Grafische Eingabe

Entsprechend der Ausführungen in Kapitel 3.1 liegt die Punktfolge der grafischen Eingabe entweder bezüglich der Erfassung in festen Zeitintervallen oder auf Basis gleichmäßiger (Mindest-) Abstände vor. Als Grundlage für den nachfolgend entwickelten Algorithmus wird die Eigenschaft des gleichmäßigen Abstands gefordert, insbesondere mit Blick auf die hier konkret erfolgte Implementierung des Semivariogramms. Dennoch kann eine gewisse Unregelmäßigkeit der Punktabstände an einzelnen Stellen verkraftet werden.

Bezüglich der Datenstruktur liegen die Eingabepunkte als Vektor vor, wobei jeder Punkt mindestens eine x- und y-Koordinate aufweist. Wie bereits in Kapitel 3.2 erwähnt, gehen die Eingabepunkte im nächsten Schritt des Verfahrens über zu Kontrollpunkte der B-Splinekurve; entsprechend sind nachfolgend beide Bezeichnungen gleichbedeutend.

4.1.2 Speichern und Laden der Eingabe

Ohne weitere Maßnahmen würde der Vektor mit den Kontrollpunkten nach Beendigung der Berechnungen bzw. nach dem Schließen des Programms verloren gehen. Um dem entgegenzuwirken, sieht der Programmentwurf vor, diese in einer Datei zu speichern. Ebenso soll es auch eine Möglichkeit geben, die zu einem früheren Zeitpunkt gespeicherten Informationen wieder zu laden. Zum einen dient dies der Reproduzierbarkeit vergangener Eingaben; zum anderen können so auch von externen Programmen generierte Testdaten in die Anwendung importiert werden. Darüber hinaus stellen die gespeicherten Punktfolgen beim Debugging eine Hilfestellung dar.

Als Datenformat wird eine gewöhnliche Textdatei gewählt, welche jeder Texteditor öffnen und bearbeiten kann. Im Text werden die Informationen in der Syntax einer MATLAB-Matrix abgelegt, da dieser in seiner Struktur recht einfach ist. Darüber hinaus wird dadurch im Rahmen eines erweiterten Debuggings ermöglicht, die Punktfolge in der Datei zu kopieren und in MATLAB im Command Window oder einem Skript respektive einer Funktion direkt einzufügen. Dadurch können die mathematischen Operationen als auch die Visualisierung von MATLAB bei der Problemanalyse unterstützend eingesetzt werden.

4.2 Approximation der Punktfolge

Die aus Kapitel 4.1 gewonnen Eingabepunkte werden nun entsprechend den Ausführungen in Kapitel 3.2 mit einer B-Splinekurve hoher Ordnung k approximiert. Der Wert wird an dieser Stelle relativ frei festgelegt als $k = \mathit{Anzahl\ Eingabepunkte}\ /\ 2$. Aufgrund der Abhängigkeit zur $\mathit{AnzahlderEingabepunkte}$ hat die Dichte der Eingabewerte respektive der konkrete Wert des mittleren Punktabstands einen geringen Einfluss auf die Filterwirkung der B-Splinekurve. Dies kommt insbesondere dann zum Tragen, wenn aus externen Quellen der gewählte Mindestabstand der Erfassungspunkte zu dem hier gewählten Abstand variiert.

Die Anzahl der konkreten Auswertestellen t der B-Splinekurve wird als $10 \cdot \mathit{Anzahl\ der\ Eingabepunkte}$ gewählt, um eine ausreichend feine Auflösung der Kurve zu garantieren. Zugleich trägt diese relativ hohe Auswertedichte dazu bei, die größeren Lücken bei ggf. unregelmäßigen Abständen der Eingabepunkte zu kompensieren.

Zur Berechnung der Auswertepunkte der B-Splinekurve nach Gleichung 2.7 sind die rekursiv definierten Basisfunktionen nach Gleichung 2.8 erforderlich. Aufgrund der hier gewählten rekursiven Konstruktion ergibt sich für eine betrachtete Basisfunktion $N_{i,k}$ das in Tabelle 4.1 gezeigte pyramidenähnliche Abhängigkeitsschema, welches entsprechend der Darstellung in Rogers (2001) übernommen wurde.

$N_{i,k}$					
$N_{i,k-1}$	$N_{i+1,k-1}$				
$N_{i,k-2}$	$N_{i+1,k-2}$	$N_{i+2,k-2}$			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
$N_{i,1}$	$N_{i+1,1}$	$N_{i+2,1}$	$N_{i+3,1}$	$\dots$	$N_{i+k-1,1}$

Tabelle 4.1: Abhängigkeit der B-Spline-Basisfunktionen; entsprechend der Darstellung in Rogers (2001) übernommen.

Um die Anzahl der rekursiven Aufrufe zu verringern, können die Basisfunktionen auch beginnend mit der Ordnung $k = 1$ konstruiert werden. Hierzu wird immer ein Vektor mit allen Basisfunktionen der Ordnung $k = \gamma$ an eine Funktion übergeben, um daraus einen Vektor mit den Basisfunktionen $k = \gamma + 1$ zu generieren; eine entsprechende Implementierung für die Ermittlung der Basisfunktionen der B-Splinekurve war in der Software TUM Open Infra Platform (OIP) bereits vorhanden.

4.3 Interpretation der B-Splinekurve

Die Interpretation der B-Slinekurve spaltet sich in zwei Hauptprozesse: Zunächst werden ausgehend von der oben definierten B-Splinekurve die Krümmung und die Krümmungsänderung als Datengrundlage im Teilkapitel 4.3.1 ermittelt. Die Identifikation der Trassierungselemente erfolgt anschließend im Teilabschnitt 4.3.2; dort bilden die Ergebnisse das Endresultat der erkannten Trassierung.

4.3.1 Ermittlung der Krümmung und Krümmungsänderung

Krümmung der B-Splinekurve

Um die in Kapitel 3.3 vorgestellte Methodik zur Identifikation der Trassierungselemente anwenden zu können, ist zunächst die Krümmung der bereits berechneten B-Splinekurve erforderlich. Entsprechend Gleichung 2.16 kann diese analytisch mit Hilfe der erste und zweite Ableitung der B-Splinekurve berechnet werden; für die Ableitungen werden dabei die Gleichungen 2.13 und 2.15 herangezogen. Das Ergebnis der Krümmung bildet bei einen Vektor, wobei jeder Eintrag eine Auswertestelle t repräsentiert.

Glätten der Krümmung

Im folgenden Teilschritt erfolgt die Glättung der Krümmungswerte durch einen gleitenden Mittelwert. Die Fenstergröße wird mit dem ebenfalls in Kapitel 2.3 erläuterten Semivariogramm ermittelt. Der dort beschriebene Versatz h respektive die Stelle $s_i + h$ beziehen sich dabei üblicherweise auf eine metrische Positionierung – im konkreten Anwendungsfall auf die Kurvenlänge. Da für diesen Algorithmus in Kapitel 4.1.1 weitgehend gleichmäßige Abstände der Datenpunkte gefordert wurden, erfolgt an dieser Stelle die Vereinfachung auf die Indexstellen im Vektor der Krümmung. Entsprechend wird h zu einer ganzzahligen Variable im Intervall $[1; n-1]$, wobei n die Anzahl der Krümmungswerte ist.

Darüber hinaus wird bei der Semivariogrammvariable $\gamma(h)$ nach Gleichung 2.18 abweichend der Vorfaktor geändert zu $1/(2 \cdot N)$, dies entspricht auch der Gleichung 3.1 in Oliver & Webster (2015). Auf den konstanten Anteil $1/\sigma_a$ des Vorfaktors wird verzichtet, da im weiteren Verlauf der Wert der Semivariogrammvariable respektive die charakteristische Größe Sill keine Verwendung findet; einzig die Stelle Range fließt in den Algorithmus ein.

Zur Bestimmung dieser Stelle wird die in Kapitel 2.3 beschriebene Eigenschaft verwendet, dass die Semivariogrammvariable ab dort konstant verläuft, folglich ist dort die Bedingung $\gamma(h_i) < \gamma(h_{i+1})$ nicht mehr erfüllt. Da bei der Berechnung die Werte wie oben beschrieben bezüglich ihrer Indexstelle verwendet wurden, entspricht h bereits der Anzahl Punkte w, welche die Größe des Fensters im gleitenden Mittelwert beschreiben.

Für die Berechnung des gleitenden Mittelwerts nach Gleichung 2.17 wird die halbe Anzahl K der Punkte, welche im Fenster liegen, festgelegt als

$$K = \frac{w-1}{2} \tag{4.1}$$

Von der Grundlagenerläuterung in Kapitel 2.3 abweichend wird hier für die Randwerte an den Stellen 1 bis K und $n-K$ bis n jedoch das Fenster so verkleinert, dass es immer noch symmetrisch zur betrachteten Stelle i ist. Dadurch wird eine sonst sehr starke Abweichung der Anfangswerte verhindert, falls dort bereits ein näherungsweise nichtkonstanter Verlauf vorliegt

Krümmungsänderung

Die analytische Definition der Ableitung einer B-Splinekurve kann hier nicht mehr angewendet werden, da die Werte der Krümmung durch den gleitenden Mittelwert manipuliert wurden. Um die Krümmungsänderung a' durch numerisches Ableiten zu ermitteln, gibt Gikas

& Stratakos (2012) die Formel

$$a'_i = \frac{a_{i+1} - a_i}{s_{i+1} - s_i} \tag{4.2}$$

an, wobei die Krümmung a_i an der Stelle s_i respektive a_{i+1} an der Stelle s_{i+1} eingehen. Aufgrund dieses Berechnungsvorgehens wird der Ergebnisvektor der Krümmungsänderungen im Vergleich zum Vektor der Krümmungen um ein Element kürzer.

4.3.2 Identifikation der Trassierungselemente

Finden der Elementendpunkte

Aufgrund der in Kapitel 3.3 vorgestellten Charakteristik des Verlaufs der Krümmungsänderung können die Endpunkte der Trassierungselemente an den Positionen gefunden werden, wo die Werte der Krümmungsänderung von 0 auf $\neq 0$ oder andersherum wechseln. Da es sich hier jedoch um Daten einer Handzeichnung handelt, welche immer noch von der Störung der Informationen sowie der Störung des Eingabegeräts (siehe Kapitel 3.1) beeinflusst sind, ist das Finden von Werten $= 0$ nicht möglich. Vielmehr ist hier eine Toleranzgrenze notwendig, um zumindest die Werte nach ≈ 0 klassifizieren zu können.

Hierfür wird die Bedingung

$$|a'| \leq a'_0 = \max\{|0{,}25 \cdot a'_{min}|;\ |0{,}25 \cdot a'_{max}|\} \tag{4.3}$$

definiert. Der Ansatz von 25% basiert auf empirischen Beobachtungen bei Testdaten. Wie in Kapitel 2.1.2 vorgestellt, besitzen die unterschiedlichen Verkehrswegetypen sowie deren Entwurfsklassen verschiedene Spannweiten der Radien respektive Krümmung; mit der prozentualen Festlegung des Toleranzwerts a'_0 wird dies berücksichtigt. Je Wert von a' wird das Prüfergebnis als Indikator in einem Vektor gespeichert, wobei 0 für $|a'| \leq a'_0$ und 1 für $|a'| > a'_0$ stehen.

Im Rahmen der Grundlagen wurden im Kapitel 2.1.2 die Mindestlängen der verschiedenen Trassierungselement und der verschiedenen Verkehrswegetypen vorgestellt. Hierauf basierend folgt nun ein kompaktes Extrakt:
Bei Geraden im Straßenwesen ist die kürzeste Mindestlängen 400 m zwischen gleichsinnig gekrümmten Kurven; bei Eisenbahnstrecken stellt 48 m die kleinste Mindestlänge gemäß Regelwert dar. Bei Kreisbögen der Straße erstreckt sich die Mindestlänge von 40 bis 75 m. Klothoiden der Schienenwege weisen eine kleinste Länge von 163 m auf.
Angesichts der wesentlichen hier zusammengefassten Werte und derer in Kapitel 2.1.2 erläuterten Spannen wird eine Mindestelementlänge von 50 m als geeigneter Kompromiss erachtet. Insbesondere die höheren Mindestlängen wie beispielsweise 163 und 400 m werden

an dieser Stelle weniger beachtet, da die Bestimmung der Elementtypen im Rahmen des hier vorgeschlagenen Algorithmus erst nachfolgend geschieht.

Für jede zusammenhängende Folge von 0 oder 1 im Indikatorvektor der Krümmungsänderung wird die dazugehörige Länge der B-Splinekurve ermittelt und in einem neuen temporären Vektor als Elementlänge abgelegt. In diesem Längenvektor werden diejenigen Elemente identifiziert, welche die Mindestlänge nicht einhalten; sie werden im Rahmen dieses Algorithmus als Störung in den Eingabedaten angesehen. Ist an einer Stelle nur ein Element zu kurz, so wird dort der Indikator invertiert; sind mehrere zusammenhänge Elemente zu kurz, wird für den Mittelwert dieser zusammenhängenden Elemente nochmals die Bedingung der Gleichung 4.3 geprüft und das Ergebnis für diese zusammenhängenden Elemente gemeinsam angewendet. Parallel zum Längenvektor erfolgt auch das Festhalten der dazugehörigen Positionen im Vektor der Krümmungsänderung, sie repräsentieren den Beginn und das Ende der jeweiligen Trassierungselemente.

Klassifizieren der Elementtypen

Entsprechen den Darstellungen in Kapitel 3.3 sind Elemente mit einer Krümmungsänderung von ≈ 0 Geraden oder Kreisbögen, welche anhand ihrer Krümmung zu unterscheiden sind. Für den Bereich zwischen dem zuvor bestimmten Elementstart und -ende erfolgt die Abschätzung der durchschnittlichen Krümmung durch einen arithmetischen Mittelwert. Zur Unterscheidung, ob die Krümmung ≈ 0 oder deutlich $\neq 0$ ist, wird wiederum die Bedingung der Gleichung 4.3 herangezogen, wobei jedoch anstelle der Krümmungsänderung a' die Krümmung a eingeht.

Gerade Im Fall der Krümmung ≈ 0 liegt beim betrachteten Element eine Gerade vor. Bei dieser geht lediglich der bereits markierte Startpunkt in das Endergebnis über; basierend auf dem Vektor, welcher vom Start- zum Endpunkt zeigt, erfolgt zudem die Berechnung der Geradenrichtung bezüglich der positiven x-Achse sowie deren Länge.

Kreisbogen Im entgegengesetzten Fall der Krümmung $\neq 0$ liegt ein Kreisbogen vor. Zur Ermittlung dessen Parameter werden die beiden Endpunkte sowie ein weiterer Punkt auf dem Kreisbogen herangezogen; diese sind entsprechend ihrer Reihenfolge auf dem Kreisbogen im Weiteren als Punkte A, B & C bezeichnet. Der Radius als wesentlicher Kreisparameter wird durch eine geometrische Betrachtet über Zuhilfenahme des Kreismittelpunkts errechnet. Hierzu werden zunächst die Richtungsvektoren

$$\vec{AB} = \vec{B} - \vec{A} \tag{4.4a}$$

$$\vec{BC} = \vec{C} - \vec{B} \tag{4.4b}$$

und die Mittelpunkte

$$\vec{P_1} = \vec{A} + 0{,}5 \cdot \vec{AB} \tag{4.5a}$$
$$\vec{P_2} = \vec{B} + 0{,}5 \cdot \vec{BC} \tag{4.5b}$$

zwischen den Punkten A, B & C berechnet. Die Richtungsvektoren werden mit der Vorschrift $x_{neu} = y$; $y_{neu} = -x$ um 90 Grad gedreht, wobei die neuen Vektoren wie folgt bezeichnet sind: $AB \rightarrow v$, $BC \rightarrow w$

Die berechneten Mittelpunkte und gedrehten Richtungsvektoren bilden die Geradengleichungen 4.6a und 4.6b, welche durch das gesuchte Kreiszentrum verlaufen. Als Schnittpunkt der beiden Geraden kann dessen Position mit Hilfe eines linearen Gleichungssystems (LGS) gefunden werden:

$$\vec{g_1}(n_1) = \vec{P_1} + n_1 \cdot \vec{v} \tag{4.6a}$$
$$\vec{g_2}(n_2) = \vec{P_2} + n_2 \cdot \vec{w} \tag{4.6b}$$

Hierzu werden die beiden Geradengleichungen gleichgesetzt zu

$$P_1 + n_1 \cdot v = P_2 + n_2 \cdot w$$

Das LGS ergibt sich daraus als

$$n_1 \cdot v_x - n_2 \cdot w_x = P_{2,x} - P_{1,x} \tag{4.7a}$$
$$n_1 \cdot v_y - n_2 \cdot w_y = P_{2,y} - P_{1,y} \tag{4.7b}$$

Durch das Auflösen der Zeile 4.7a des LGS ergibt sich n_1 zu

$$n_1 = \frac{(P_{2,x} - P_{1,x}) + n_2 \cdot w_x}{v_x} \tag{4.8}$$

Wird n_1 in die Gleichung 4.7b eingesetzt und diese nach n_2 aufgelöst, ergibt sich eine finale Berechnungsformel für n_2:

$$n_2 = \frac{v_x \cdot (P_{2,y} - P_{1,y}) - v_y \cdot (P_{2,x} - P_{1,x})}{w_x \cdot v_y - w_y \cdot v_x} \tag{4.9}$$

Schließlich ergeben sich die Koordinaten des Kreismittelpunkts M durch das Einsetzen des Werts von n_2 in Gleichung 4.6b.

Die Parameter des Kreisbogens lassen sich jetzt berechnen als

$$\text{Radius } R = |\vec{A} - \vec{M}| \tag{4.10}$$

$$\text{Orientierung} = \begin{cases} 1 & \text{für linksgekrümmt wenn } A > 0 \\ 0 & \text{für rechtsgekrümmt wenn } A < 0 \end{cases} \tag{4.11}$$

$$\text{mit der Fläche } A = x_{\vec{MA}} \cdot y_{\vec{MB}} - x_{\vec{MB}} \cdot y_{\vec{MA}} \text{ (Bronstein } et\ al.\text{, 2016)} \tag{4.12}$$

Klothoide Eine solche liegt vor, wenn der Indikator der Krümmungsänderung den Wert 1 aufweist. Aufgrund der in Kapitel 2.1.1 vorgestellten Anforderungen einer Trassierung, sowie wegen der Vorgehensweise beim Finden der Elementendpunkte, ist bereits gewährleistet, dass sich unmittelbar vor und nach einer Klothoide definitiv eine Grade oder ein Kreisbogen befindet. Deren Parameter sind wie oben beschrieben bereits ermittelt; entsprechend können die Tangentenvektoren $\vec{t}_{Start}$ am Beginn der Klothoide und $\vec{t}_{Ende}$ am Ende der Klothoide mit deren Hilfe berechnet werden.

Der Winkel τ zwischen diesen beiden Vektoren ergibt sich nach Bronstein *et al.* (2016) als

$$\tau = \text{arc cos} \frac{\langle \vec{t}_{Start}, \vec{t}_{Ende} \rangle}{|\vec{t}_{Start}| \cdot |\vec{t}_{Ende}|} \tag{4.13}$$

und beschreibt entsprechend der Abbildung 2.3 die Richtungsänderung, welche die Klothoide aufweist. Durch Umstellen der Gleichung 2.5 ergibt sich die Kurvenlänge L der Klothoide, der Radius R stammt dabei vom angrenzenden Kreisbogen. Weiter kann durch Umstellen der Gleichung 2.2 der Klothoidenparameter A berechnet werden.

$$L = 2 \cdot R \cdot \tau \tag{4.14}$$

$$A = \sqrt{R \cdot L} \tag{4.15}$$

Die weiteren Eigenschaften Orientierung der Klothoide kann ebenso wie der Radius vom angrenzenden Kreisbogen übernommen werden.

Kapitel 5

Implementierung

Dieses Kapitel stellt in Abschnitt 5.1 zunächst die open source Anwendung TUM Open Infra Platform (OIP) kurz vor und beschreibt, wie mit dieser im Rahmen der Implementierung gearbeitet wurde. In den folgenden Teilkapitel 5.2 bis 5.6 werden jeweils die verschiedenen Funktionsgruppen aus Kapitel 4 als Pseudocode aufgezeigt und wesentliche Teilschritte erläutert.

Neben der Darstellung als Pseudocode wurde der vorgestellte Algorithmus auch in OIP mit der Programmiersprache C++ implementiert. Der entsprechende Quellcode kann unter folgendem Link öffentlich eingesehen werden:

https://github.com/tumcms/Open-Infra-Platform/pull/202

Dort wird der für diese Arbeit relevante Stand durch einen Kommentar am 11. Oktober 2020 gekennzeichnet.

5.1 TUM Open Infra Platform (OIP)

OIP ist eine open source Entwicklung des Lehrstuhls für Computergestützte Modellierung und Simulation der Technischen Universität München (TUM). Die Software dient zum Öffnen und Betrachteten von Industry Foundation Classes (IFC)-Dateien und ist in der Programmiersprache C++ geschrieben (OIP, 2020b).

Die Implementierung des in Kapitel 4 vorgestellten Algorithmus erfolgte in OIP. Dort war bereits eine Klasse `SplineConverterT` mit Funktionen zum Auswerten einer B-Splinekurve vorhanden, diese waren jedoch auskommentiert und aufgrund eines Updates nicht mehr lauffähig. Im Rahmen eines Refactorings wurden diese Funktionen überarbeitet; in diesem Zuge erfolgte auch eine Neustrukturierung in kleinere Teilfunktionen gemäß einer objektorientierten Programmstruktur. Manche dieser Funktionen, also auch ein Teil der neu entwickelten Funktionen, wurden in eine neue Klasse `SplineUtilities` ausgelagert.

Die Überarbeiteten Funktionen zur B-Splineberechnung sind in der konkreten Implementierung des interaktiven Achsenentwurfs integriert. Zudem erfolge eine Einbettung in die Benutzeroberfläche, um die Hauptfunktion des Achsenentwurfs aufrufen zu können. Die Hauptfunktion `convertSketchToAlignment` ist mit ihren Nebenfunktionen in einer neuen Klasse `SplineInterpretation` angelegt; zudem wurde eine Klasse `SplineInterpretationElement` erstellt, dessen Objekte die Informationen der identifizierten Trassierungselemente aufnehmen.

Eine weitere Einbeziehung von OIP erfolgte hier nicht. Eine grafische Eingabe, als auch das Speichern einer IFC-Datei sind derzeit leider nicht möglich (OIP, 2020a). Diese Funktionalitäten instand zusetzen war nicht Teil des Arbeitsumfangs dieser Bachelorarbeit.

5.2 Hauptfunktion

Der gesamte im Kapitel 4 vorgestellte Algorithmus wird durch eine zentrale Hauptfunktion zusammengefasst, welche für die verschiedenen Aufgaben entsprechende Unterfunktionen aufruft. Die Darstellung Algorithmus 1 zeigt dessen Aufbau.

Die in Kapitel 3.1 vorgestellten Teilprozesse der grafischen Eingabe einschließlich Verarbeitung sind gut erkennbar, analog dazu ist auch die Struktur der in Kapitel 4 angelegten Abschnitte deutlich: In Zeile 2 erfolgt die Eingabe der Kontrollpunkte; weiter werden in den Zeilen 3 bis 7 zunächst die Eigenschaften der B-Splinekurve definiert sowie die Auswertung der Kurve als Approximation berechnet. Die Zeilen 8 bis 11 enthalten mit der Berechnung der Krümmung und Krümmungsänderung die Interpretation der B-Splinekurve; um abschließend in den Zeilen 12 und 13 abschließend die Identifikation der Trassierungselement vorzunehmen.

5.3 Eingabe der Punktfolge

In Kapitel 4.1.2 wurde die eigene Anforderung aufgestellt, eine Möglichkeit zum Speichern und Laden der Eingabepunkte zu schaffen. Ebenfalls erfolgte dort bereits die Festlegung auf eine gewöhnliche Textdatei, welche die Daten im Syntax einer MATLAB-Matrix beinhaltet. Ein Ausschnitt aus einer solchen Datei ist in Syntax 5.1 dargestellt. Die Anzahl der Punkte, welche am Beginn der ersten Zeile enthalten ist, dient lediglich informativen Zwecken und um ggf. im Rahmen einer konkreten Implementierung die Vektorgröße vorab definieren zu können. Die für MATLAB notwendige Matrixdefinition erfolgt mit den eckigen Klammern in Zeile zwei und acht. Ab Zeile drei repräsentiert jede Zeile einen Punkt mit durch Leerzeichen getrennte x-, y-, und z-Koordinaten; letztere ist aufgrund der gegebenen Datenstrukturen in OIP bereits mitgeführt, für die Interpretation der horizontalen Trassierung findet sie jedoch keine Anwendung.

Algorithmus 1 Hauptfunktion

Return
elements Eigenschaften der Interpretierten Elemente

1: **function** mainFunction
2: *controlPoints* ← obtainControlPoints() ▷ Kontrollpunkte zeichnen oder laden
3: *nControlPoints* ← *controlPoints*.size ▷ Anzahl der Kontrollpunkte
4: *nCurvePoints* ← $10 \cdot nControlPoints$ ▷ Anzahl der Auswertestellen
5: *order* ← ceil($nControlPoints/2$) ▷ Ordnung der B-Splinekurve
6: *knotArray* ← obtainKnotArrayOpenUniform(*nControlPoints*, *order*) ▷ offener gleichmäßiger Knotenvektor
7: *bsplinePoints* ← computeBSplineCurveWithKnots(*order*, *knotArray*, *controlPoints*, *nCurvePoints*) ▷ Kurvenpunkte der Approximation berechnen
8: [*curveLength*, *curvature*] ← computeCurvatureOfBSplineCurveWithKnots(*order*, *controlPoints*, *knotArray*) ▷ Berechnung Krümmung mit Kurvenlänge
9: *curvature* ← movingAverageVariableWindow(*curvature*) ▷ Glätten der Krümmung
10: *curvatureChange* ← numericDerivative(*curveLength*, *curvature*) ▷ Krümmungsänderung
11: *curvatureChange* ← movingAverageVariableWindow(*curvatureChange*) ▷ Glätten der Krümmungsänderung
12: [*elementLengthApprox*, *elementIndices*, *elementIndicator*] ← identifyElementEndpoints(*curveLength*, *curvatureChange*) ▷ Endpunkt der Elemente finden
13: *elements* ← identifyElementTypes(*elementLengthApprox*, *elementIndices*, *elementIndicator*, *bsplinePoints*, *curvature*) ▷ Parameter der Element festlegen
14: **return** *elements*
15: **end function**

Syntax 5.1: Ausschnitt einer gespeicherten Punktfolge

```
168 = Number of vertices; the following lines can be pasted
   into MATLAB:
[
353.3690 462.9178 0 ;
363.5100 463.0969 0 ;
373.6432 463.5332 0 ;
(...)
1559.7153 1275.8854 0 ;
1567.0812 1268.9897 0 ];
```

5.4 Approximation der Punktfolge

Die Funktion `obtainKnotArrayOpenUniform` erzeugt einen zur Anzahl der Eingabepunkte passenden Knotenvektor. Ihr Aufbau ist in Algorithmus 11 im Anhang dargestellt.

Ebenfalls im Anhang als Algorithmus 12 ist die Funktion `computeBSplineCurveWithKnots`, welche für alle Auswertestellen t die Koordinaten der B-Splinekurve berechnet. Die hiervon aufgerufene Funktion `computePointOfBSpline` (Algorithmus 13, siehe Anhang) übernimmt die Auswertung für einen konkreten Wert von t.

Der hingegen nachfolgend gezeigte Algorithmus 2 der Funktion `computeBSplineBasisFunctions` ermittelt alle B-Spline Basisfunktionen für eine konkrete Stelle t; durch die zuvor erwähnte Funktion `computePointOfBSpline` wird diese aufgerufen. Innerhalb der Funktion ist in Zeile 5 erkennbar, dass hier zunächst die Basisfunktionen der Ordnung $k = 1$ berechnet werden, bevor mittels einer Schleife die jeweils nächst höhere Ordnung in Zeile 7 konstruiert wird; dies entspricht dem in Kapitel 4.2 erläuterten Vorgehen. Die hier im Weiteren aufgerufenen Funktionen `obtainBasisFunctionFirstOrder` und `obtainBasisFunctionNextOrder` können im Anhang als Algorithmus 14 und 15 vorgefunden werden.

Algorithmus 2 Berechnung der B-Splinebasisfunktionen

Input

order	Ordnung
t	Auswertestelle
nControlPoints	Anzahl der Kontrollpunkte
knotArray	Knotenvektor

Return

basisFuncs	B-Splinebasisfunktionen der Stelle t

1: **function** computeBSplineBasisFunctions(*order*, t, *nControlPoints*, *knotArray*)
2: *degree* $\leftarrow$ *order* $-$ 1 ▷ Grad der Basisfunktionen
3: *nBasisFuncs* $\leftarrow$ *degree* $+$ *nControlPoints* ▷ Anzahl der Basisfunktionen der Ordnung $k = 1$
4: *nKnots* $\leftarrow$ *knotArray*.size ▷ Anzahl der Knoten
5: *basisFuncs* $\leftarrow$ obtainBasisFunctionFirstOrder(t, *nBasisFuncs*, *knotArray*) ▷ Basisfunktionen der Ordnung $k = 1$
6: **for** $k \leftarrow 2$ **:** *degree* **do**
7: *basisFuncs* $\leftarrow$ obtainBasisFunctionNextOrder(k, t, *knotArray*, *tempBasisFuncs*) ▷ Ordnung der Basisfunktionen erhöhen
8: **end for**
9: *basisFuncs* $\leftarrow$ *basisFuncs*(1 **:** *nControlPoints*) ▷ Kopieren der relevanten Formfunktionen
10: **return** *basisFuncs*
11: **end function**

5.5 Ermittlung der Krümmung und Krümmungsänderung

Die Hauptfunktion `computeCurvatureOfBSplineCurveWithKnots` zur Berechnung der Krümmung wird von Algorithmus 3 gezeigt. Die von ihr aufgerufenen Unterfunktionen `computePointOfDerivOne` und `computePointOfDerivTwo` arbeiten prinzipiell analog zur bereits genannten Funktion `computePointOfBSpline`, daher sind diese einschließlich ihrer Teilfunktionen im Anhang als Algorithmen 16, 17, 18, 19, 20 und 21 enthalten.

Vorbereitend zur später durchzuführenden numerischen Ableitung sowie der ersten Abschätzung der Elementlängen wird hier in den Zeilen 11 bis 16 bereits die Kurvenlänge der B-Splinekurve berechnet.

Algorithmus 3 Berechnung der Krümmung der B-Splinekurve

Input

order	Ordnung
controlPoints	Kontrollpunkte
knotArray	Knotenvektor
nCurvePoints	Anzahl der Auswertepunkte

Return

curveLength	Kurvenlänge der B-Splinekurve
curvature	Krümmung der B-Splinekurve

1: **function** computeCurvatureOfBSplineCurveWithKnots(*order*, *controlPoints*, *knotArray*, *nCurvePoints*)
2: $nControlPoints \leftarrow controlPoints$.size ▷ Anzahl der Kontrollpunkte
3: $knotStart \leftarrow knotArray(order)$
4: $knotEnd \leftarrow knotArray(controlPoints.\text{size} + 1)$
5: $step \leftarrow (knotEnd - knotStart)/controlPoints.\text{size} - 1$ ▷ Intervallgröße der Auswertestellen t
6: $t \leftarrow knotStart$
7: **for** $i \leftarrow 1$ **:** $nCurvePoints$ **do**
8: $curvePoint =$ computePointOfBSpline($order$, t, $controlPoints$, $knotArray$)
9: $derivOne =$ computePointOfDerivOne($order$, t, $controlPoints$, $knotArray$)
10: $derivTwo =$ computePointOfDerivTwo($order$, t, $controlPoints$, $knotArray$) ▷ Auswertepunkt, erste & zweite Ableitung der B-Splinekurve
11: **if** $i = 1$ **then**
12: $tempLength \leftarrow 0$
13: **else**
14: $tempLength \leftarrow curveLength(i-1) + (prevCurvePoint - curvePoint)$.length
15: **end if**
16: $curveLength(i) \leftarrow tempLength$ ▷ Kurvenlänge
17: $curvature(i) \leftarrow (derivOne.\text{x} \cdot derivTwo.\text{y} - derivTow.\text{x} \cdot derivOne.\text{y}$ $/(derivOne.\text{x}^2 + derivOne.\text{y}^2)^{3/2}$ ▷ Krümmung
18: $prevCurvePoint \leftarrow curvePoint$ ▷ Speichern des aktuellen Kurvenpunkts
19: $t \leftarrow t + 1$ ▷ Vorrücken zur nächsten Auswertestelle
20: **end for**
21: **return** [*curveLength*, *curvature*]
22: **end function**

Das Vorgehen zur Glättung der Daten mit dem gleitenden Mittelwert ist mit der Funktion `movingAverageVariableWindow` in Algorithmus 4 aufgezeigt. Die als `range` bezeichnete Fenstergröße wird mittels des Semivariogramms in der Funktion `variogrammGetRange` bestimmt, der dazugehörige Pseudocode ist als Algorithmus 22 im Anhang beigefügt.

Die variable Fenstergröße für Werte am Rand des Datenvektors wird in den Zeilen 6 bis 15 festgelegt. Hierbei erfolgt zunächst den den Zeilen 6 und 7 die Berechnung der Start- und Endindices unabhängig davon, ob sich die aktuelle Position i am Rand befindet. Erst bei den Abfragen in Zeile 8 und 12 wird geprüft, ob das Fenster anschließend in den Zeilen 9 und 10

respektive 13 und 14 verkleinert werden muss. Exemplarisch für den linken Rand wird bei der Korrektur des Fensterendes $jEnd$ ausgehend von der aktuellen Position i die Anzahl der Werte $i - jStart$, welche sich zwischen Start und i befinden, hinzuaddiert. Dadurch wird das in Kapitel 4.3.1 als symmetrisch gesetzt Fenster realisiert.

Algorithmus 4 Gleitender Mittelwert mit variabler Fenstergröße

Input
data Vektor mit den zu glättenden Datenwerte
Return
dataSmooth Vektor mit geglätteten Datenwerte

1: **function** movingAverageVariableWindow(*data*)
2: $range \leftarrow$ variogrammGetRange(*data*) ▷ Fenstergröße
3: $K \leftarrow$ ceil($range/2$) ▷ halbe Fenstergröße
4: $n \leftarrow$ *data*.size ▷ Anzahl der Datenwerte
5: **for** $i \leftarrow 1 : n$ **do**
6: $jStart \leftarrow i - K$ ▷ Startindex des Fensters
7: $jEnd \leftarrow i + K$ ▷ Endindex des Fensters
8: **if** $jStart < 1$ **then** ▷ Fenster verkleinern, da am linken Rand
9: $jStart \leftarrow 1$
10: $jEnd \leftarrow i + (i - jStart)$
11: **end if**
12: **if** $jEnd > n$ **then** ▷ Fenster verkleinern, da am rechten Rand
13: $jEnd \leftarrow n$
14: $jStart \leftarrow i - (n - i)$
15: **end if** ▷ Position des Fensters wurde festgelegt
16: $dataSmooth(i) \leftarrow 0$
17: **for** $j \leftarrow jStart : jEnd$ **do**
18: $dataSmooth(i) \leftarrow dataSmooth(i) + data(j)$ ▷ Summe der Datenwerte
19: **end for**
20: $dataSmooth(i) \leftarrow dataSmooth(i)/(jEnd - jStart + 1)$ ▷ Mittelwert der Daten
21: **end for**
22: **return** *dataSmooth*
23: **end function**

Algorithmus 23 im Anhang zeigt die Umsetzung der numerischen Ableitung zur Berechnung der Krümmungsänderung. Die hiervon erhaltenen Werte werden durch die Hauptfunktion (Alg. 1) nochmals an die Funktion `movingAverageVariableWindow` übergeben. Nach dieser erneuten Glättung der Krümmungsänderung erfolgt der Übergang zur Identifikation der Trassierungselemente.

5.6 Identifikation der Trassierungselemente

Wie bereits in Kapitel 4.3.2 angedeutet, ist die Identifikation in zwei wesentliche Teilprozesse aufgeteilt.

Im ersten Teil übernimmt die Funktion `identifyElementEndpoints` (Alg. 5) das Finden der Elementendpunkte. Hierzu beurteilt die Funktion `indicateCurvatureChange` (Alg. 24, siehe Anhang) für jeden Wert der Krümmungsänderung, ob dieser innerhalb deiner Toleranzgrenze liegt. `obtainElementsFromIndicator` (Alg. 26, im Anhang) in Zeile 3 legt für alle Folgen identischer Indikatoren die Indices der Element fest und berechnet zudem eine erste Abschätzung der Elementlänge auf Grundlage der B-Spline-Kurvenlänge.

Algorithmus 5 Endpunkte der Elemente bestimmen

Input

curveLength	Vektor der Kurvenlänge der B-Splinekurve
curvatureChange	Vektor der Krümmungsänderung der B-Splinekurve

Return

elementLength	Vektor mit Elementlängen
elementIndices	Vektor mit Start- & Endindizes
elementIndicator	Vektor mit Indikator der Krümmungsänderung

1: **function** identifyElementEndpoints(*curveLength*, *curvatureChange*)
2: [*indicator*, *curvatureZero*] ← indicateCurvatureChange(*curvatureChange*)
▷ Beurteilung der Krümmungsänderung; Schwellenwert der Krümmungsänderung
3: [*elementLength*, *elementIndices*, *elementIndicator*] ←
obtainElementsFromIndicator(*indicator*, *curveLength*)
▷ Je Element Länge und Start- & Endindizes
4: *elementIndicator* ← correctShortElements(*elementLength*, *elementIndices*, *elementIndicator*, *curvatureChange*, *curvatureZero*)
▷ Indikatoren zu kurzer Elemente anpassen
5: [*elementLength*, *elementIndices*, *elementIndicator*] ←
mergeShortElements(*elementLength*, *elementIndices*, *elementIndicator*)
▷ Elemente mit identischem Indikator verschmelzen
6: **return** [*elementLength*, *elementIndices*, *elementIndicator*]
7: **end function**

Im weiteren Verlauf wird durch `identifyElementEndpoints` in Zeile 4 die Funktion `correctShortElements` (Algorithmus 6) aufgerufen, welche zu kurze Element bezüglich der in Kapitel 4.3.2 definierten Mindestlänge findet und mit Nachbarelementen vereinigt.

Hierzu wird zunächst in Zeile 5 fortlaufend für jedes Element die Mindestlänge geprüft. Wird ein zu kurzes Element gefunden, sucht die Schleife in Zeile 7 nach dem letzten unmittelbar zusammenhängenden zu kurzen Element. Abhängig von der Anzahl zu kurzer Elemente (Prüfung in Zeile 14) wird entweder lediglich der Indikator der Krümmungsänderung für das eine Elemente modifiziert (Zeile 15 bis 19) oder für die mehreren Elemente ein Mittelwert der Krümmungsänderung gebildet (Zeile 21 bis 24). Bei letzterem wird der neue Indikator für alle dort zusammenhängenden kurzen Elemente einheitlich übernommen (Zeile 24).
Da j im Vektor der Elemente weiter nach vorne gerückt ist und dabei die zu kurzen Elemente bereits bearbeitet hat, kann auch der Zähler i in Zeile 26 entsprechend nach vorne springen.

Der letzte Prozessschritt von `identifyElementEndpoints` ist der Aufruf von `mergeShortElements` (Alg. 7). Innerhalb dieser Unterfunktion werden die zuvor bearbeiteten zu kurzen Elemente mit ihren Nachbarn zusammengefügt. Hierfür wird in Zeile 4 nach aufeinanderfolgenden Elementen mit identischem Indikator der Krümmungsänderung gesucht. Diese Suche der Schleife erfolgt vom Vektorende beginnend nach vorne bis zum ersten Element, da sich im Folgenden der Vektor verkürzt. Ist eine Übereinstimmung der Indikatoren gefunden, werden alle Attribute des Elements $i + 1$ in das vorherige Element i übertragen; die Länge addiert sich dabei und bei den Indices muss lediglich der Endindex des Elements $i + 1$ als neuer Endindex des Elements i übernommen werden. Abschließend werden die drei Eigenschaften Länge, Indices und Indikator des $(i + 1)$-ten Elements gelöscht.

Algorithmus 6 Indikatoren von zu kurzen Elementen korrigieren

Input

elementLength	Vektor mit Elementlängen
elementIndices	Vektor mit Start- & Endindizes
elementIndicator	Vektor mit Indikator der Krümmungsänderung
curvatureChange	Vektor der Krümmungsänderung der B-Splinekurve
curvatureZero	Schwellenwert der Krümmungsänderung

Return

elementIndicator	Vektor mit Indikator der Krümmungsänderung

```
1: function correctShortElements(elementLength, elementIndices, elementIndicator,
   curvatureChange, curvatureZero)
2:     minLength ← 50                                          ▷ Mindestelementlänge
3:     n ← elementLength.size
4:     for i ← 1 : n do
5:         if elementLength(i) < minLength then
6:             firstElement ← i
7:             for j ← i : n do
8:                 if elementLength(j) ≥ minLength or j = n − 1 then
9:                     if j = n − 1 then                       ▷ letztes Element erreicht
10:                        lastElement ← j
11:                    else                                    ▷ nicht am letzten Element
12:                        lastElement ← j − 1
13:                    end if
14:                    if firstElement = lastElement then      ▷ nur ein Element zu kurz
15:                        if elementIndicator(firstElement) = 0 then
16:                            elementIndicator(firstElement) = 1
17:                        else
18:                            elementIndicator(firstElement) = 0
19:                        end if
20:                    else                ▷ Mehrere zusammenhängende Elemente zu kurz
21:                        idStart ← elementIndices(firstElement).elStart
22:                        idEnd ← elementIndices(lastElement).elEnd
23:                        curvatureChangePart ← curvatureChange(idStart : idEnd)
24:                        elementIndicator(firstElement : lastElement) ←
                               abs(indicateDataByAverage(
                               curvatureChangePart, curvatureZero))
25:                    end if
26:                    i ← j        ▷ Auf erstes Element nach bearbeitetem Bereich springen
27:                    break for
28:                end if
29:            end for
30:        end if
31:    end for
32:    return elementIndicator
33: end function
```

Algorithmus 7 Zusammenfassen aufeinanderfolgender Elemente mit identischem Indikator

Input & Return

$elementLength$	Vektor mit Elementlängen
$elementIndices$	Vektor mit Start- & Endindizes
$elementIndicator$	Vektor mit Indikator der Krümmungsänderung

```
1:  function mergeShortElements(elementLength, elementIndices, elementIndicator)
2:      n ← elementLength.size                                  ▷ Anzahl der Elemente
3:      for i ← n − 1 : 1 do
4:          if elementIndicator(i) = elementIndicator(i + 1) then
5:              elementLength(i) ← elementLength(i) + elementLength(i + 1)
                                                                ▷ Längen zusammenfassen
6:              elementLength(i + 1) ← [ ]                      ▷ Eintrag löschen
7:              elementIndices(i).elEnd ← elementIndices(i + 1).elEnd
                                                                ▷ Index des Endpunkts übertragen
8:              elementIndices(i + 1) ← [ ]                     ▷ Eintrag löschen
9:              elementIndicator(i + 1) ← [ ]                   ▷ Eintrag löschen
10:         end if
11:     end for
12:     return [elementLength, elementIndices, elementIndicator]
13: end function
```

Der zweite Teil der Identifikation erfolgt durch `identifyElementTypes` (Alg. 8, über zwei Seiten). Hierfür werden zunächst in der Schleife Zeile 4 für alle Elemente mit einer Krümmungsänderung von ≈ 0 die Mittelwerte der Krümmung bestimmt und bezüglich einem Toleranzwert auf ≈ 0 geprüft. Weisen sie keine respektive eine geringe Krümmung auf, legt `obtainStraigth` (Alg. 28, siehe Anhang) in Zeile 10 bis 12 die Eigenschaften einer Geraden als Trassierungselement fest. Weist das Element jedoch eine Krümmung auf, so berechnet `obtainArc` (Alg. 9) die Parameter eines Kreisbogens in Zeile 14 bis 17, nähere Erläuterungen zu dieser Teilfunktion siehe Seite 41.

Nachdem alle Geraden und Kreise bestimmt sind, werden ab Zeile 21 die Klothoiden mit Parametern versehen. Zunächst müssen jedoch in den Zeilen 23 bis 30 einige Sonderfälle ausgeschlossen werden, welche im Rahmen dieser Arbeit nicht implementiert wurden. In der Zeile 34 bis 39 wird das vorherige Element bestimmt (Zeile 39) oder eine virtuelle Gerade als Vorgänger konstruiert, welche für Klothoiden als erstes Element mit einer Anfangskrümmung von 0 zur Berechnung benötigt wird. Analoges geschieht in den Zeilen 41 bis 46 für den Nachfolger. Final werden in den Zeilen 48 bis 50 die Parameter der Klothoide bestimmt, wobei die Funktion `obtainClothoid` (Alg. 10) aufgerufen wird – Erläuterungen hierzu siehe Seite 43.

Weitere Hilfsfunktionen werden ab Seite 43 erläutert.

Algorithmus 8 Elementtypen identifizieren und Parameter festlegen

Input

$elementLengthApprox$	Vektor mit Elementlängen der Approximation
$elementIndices$	Vektor mit Start- & Endindizes
$elementIndicator$	Vektor mit Indikator der Krümmungsänderung
$bsplinePoints$	Kurvenpunkte der Approximation
$curvature$	Krümmung der B-Splinekurve

Return

$element$	alle Vektoren mit Eigenschaften der Elemente

1: **function** identifyElementTypes($elementLengthApprox$, $elementIndices$, $elementIndicator$, $bsplinePoints$, $curvature$)
2: $curvatureThreshold \leftarrow$ obtainThreshold($curvature$) ▷ Schwellenwert der Krümmung
3: $n \leftarrow elementIndicator$.size ▷ Anzahl der Elemente
4: **for** $i \leftarrow 1 : n$ **do**
5: **if** $elementIndicator(i) = 0$ **then** ▷ Krümmungsänderung ≈ 0
6: $idStart \leftarrow elementIndices$.elStart
7: $idEnd \leftarrow elementIndices$.elEnd ▷ Start- & Endindex des Elements bezüglich Krümmungsvektor
8: $curvatureIndicator \leftarrow$ indicateDataByAverage($curvature(idStart : idEnd)$, $curvatureThreshold$) ▷ Beurteilung, ob Krümmung ≈ 0
9: **if** $curvatureIndicator = 0$ **then** ▷ Gerade
10: $elementType(i) \leftarrow$ 'straight'
11: $elementStart(i) \leftarrow bsplinePoints(idStart)$ ▷ Startpunkt der Gerade
12: $[elementDirection(i), elementLength(i)] \leftarrow$ obtainStraigth($bsplinePoints(idStart)$, $bsplinePoints(idEnd)$)
13: **else**
14: $elementType(i) \leftarrow$ 'arc' ▷ Kreisbogen
15: $elementStart(i) \leftarrow bsplinePoints(idStart)$
16: $idMid \leftarrow$ ceil$((idStart + idEnd)/2)$ ▷ Index in Elementmitte
17: $[elementDirection(i), elementLength(i), elementRadius(i), elementIsCCW(i), elementCenter(i), elementAngle(i)] \leftarrow$ obtainArc($bsplinePoints(idStart)$, $bsplinePoints(idMid)$, $bsplinePoints(idEnd)$, $curvatureIndicator$)
18: **end if**
19: **end if**
20: **end for** ▷ Fortsetzung siehe nächste Seite

▷ Nach Bestimmung von Geraden & Kreisbögen nun Klothoiden berechnen
21: **for** $i \leftarrow 1 : n$ **do**
22: **if** $elementIndicator(i) = 1$ **then** ▷ Krümmungsänderung ≈ 0
▷ Einige nicht unterstützte Typen ausschließen:
23: **if** $i = 1$ **and** $elementType(2)$ = 'straight' **then**
24: $elementType(i) \leftarrow$ 'clothoid with radius at sketch start not supported'
25: **else if** $i = n$ **and** $elementType(n-1)$ = 'straight' **then**
26: $elementType(i) \leftarrow$ 'clothoid with radius at sketch end not supported'
27: **else if** $elementType(i-1)$ = 'arc' **and** $elementType(i+1)$ = 'arc' **then**
28: $elementType(i) \leftarrow$ 'reversible clothoid not supported'
29: **else if** $elem.Type(i-1)$ = 'straight' **and** $elem.Type(i+1)$ = 'straight' **then**
30: $elementType(i) \leftarrow$ 'clothoid straight-radius-straight (without arc) not supported'
31: **else** ▷ Parameter der Klothoide berechnen
32: $idStart \leftarrow elementIndices$.elStart
33: $idEnd \leftarrow elementIndices$.elEnd
▷ Start- & Endindex des Elements bezüglich Krümmungsvektor
34: **if** $i = 0$ **then** ▷ vorheriges Element festlegen
35: $previousType \leftarrow$ 'straight'
36: $vector \leftarrow bsplinePoints(idStart + 1) - bsplinePoints(idStart)$
37: $previousDirection \leftarrow \text{sign}(vector.\text{y}) \cdot \text{angleOfVectors}([1; 0], vector)$
▷ virtuelles Geradenelement anlegen
38: **else**
39: $previous \leftarrow element(i-1)$
▷ alle Eigenschaften des vorherigen Elements übernehmen
40: **end if**
41: **if** $i = n$ **then** ▷ nachfolgendes Element festlegen
42: $nextType \leftarrow$ 'straight'
43: $vector \leftarrow bsplinePoints(idEnd) - bsplinePoints(idEnd - 1)$
44: $nextDirection \leftarrow \text{sign}(vector.\text{y}) \cdot \text{angleOfVectors}([1; 0], vector)$
▷ virtuelles Geradenelement anlegen
45: **else**
46: $next \leftarrow element(i+1)$
▷ alle Eigenschaften des nachfolgenden Elements übernehmen
47: **end if**
48: $elementType(i) \leftarrow$ 'clothoid'
49: $elementStart(i) \leftarrow bsplinePoints(idStart)$
50: $element(i) \leftarrow \text{obtainClothoid}(previous, next, bsplinePoints(idStart))$
▷ Berechnung der Klothoideneigenschaften; Auflistung siehe Alg. 10 obtainClothoid
51: **end if**
52: **end if**
53: **end for**
54: **return** *element*
55: **end function**

Die Parameter eines Kreisbogens werden durch die Funktion `obtainArc` (Alg. 9 berechnet. Das in Kapitel 4.3.2 vorgestellte vektorbasierte Berechnungsverfahren zur Ermittlung des Kreismittelpunkts erfolgt dabei in den Zeilen 2 bis 9; Zeile 8 repräsentiert dabei direkt die Lösung des vorgestellten linearen Gleichungssystems lineares Gleichungssystem (LGS). Die Orientierung der Kurve in der Eigenschaft $elementIsCCW$ wird unmittelbar vom Indikator der Krümmung interpretiert; für diesen Zweck kann dieser -1 als auch $+1$ darstellen – der Fall 0 entfällt an dieser Stelle, da dieser aufgrund der Tatsache Kreisbogen ausgeschlossen ist.

Algorithmus 9 Eigenschaften eines Kreisbogens festlegen

Input

startPoint	Startpunkt des Elements
midPoint	Punkt auf Kreisbogen
endPoint	Endpunkt des Elements
curvatureIndicator	Beurteilung der Krümmung

Return

elementDirection	Richtung der Tangente am Elementstart
elementLength	Kurvenlänge des Elements
elementRadius	Radius des Kreisbogens
elementIsCCW	Orientierung des Kreisbogens
elementCenter	Mittelpunkt des Kreisbogens
elementAngle	Öffnungswinkel des Kreisbogens

1: **function** obtainArc(*startPoint*, *midPoint*, *endPoint*)
2: $AB \leftarrow midPoint - startPoint$ ▷ Richtungsvektor Start zu Mitte
3: $BC \leftarrow endPoint - midPoint$ ▷ Richtungsvektor Mitte zu Ende
4: $P_1 \leftarrow startPoint + 0.5 \cdot AB$ ▷ Mittelpunkt Start zu Mitte
5: $P_2 \leftarrow midPoint + 0.5 \cdot BC$ ▷ Mittelpunkt Mitte zu Ende
6: $v \leftarrow [AB.\text{y}; \text{-}AB.\text{x}]$ ▷ Richtungsvektor Gerade 1
7: $w \leftarrow [BC.\text{y}; \text{-}BC.\text{x}]$ ▷ Richtungsvektor Gerade 2
8: $n_2 \leftarrow (v.x \cdot (P_2.y - P_1.y) - v.y \cdot (P_2.x - P_1.x))/(w.x \cdot v.y - w.y \cdot v.x)$
▷ Parameter der Geradengleichung
9: $elementCenter \leftarrow P_2 + n_2 \cdot w$ ▷ Koordinaten des Kreismittelpunkts
10: $elementRadius \leftarrow (elementCenter - startPoint)$.length
11: **if** $curvatureIndicator = 1$ **then**
12: $elementIsCCW \leftarrow 1$ ▷ Linkskurve
13: **else**
14: $elementIsCCW \leftarrow 0$ ▷ Rechtskurve
15: **end if**
16: $elementDirection \leftarrow$ tangentDirection(
$elementCenter$, $startPoint$, $elementIsCCW$) ▷ Richtung am Elementstart
17: $elementAngle \leftarrow$ abs(angleOfVectors(
$startPoint - elementCenter$, $midPoint - elementCenter$)
+ angleOfVectors($midPoint - elementCenter$, $endPoint - elementCenter$))
▷ Öffnungswinkel des Kreisbogens in Grad
18: $elementLength \leftarrow elementRadius \cdot elementAngle \cdot pi/180$
▷ Bogenlänge mit Umrechnung von Grad in Radianten
19: **return** [*elementDirection*, *elementLength*, *elementRadius*, *elementIsCCW*, *elementCenter*, *elementAngle*]
20: **end function**

In der Funktion `obtainClothoid` (Alg. 10) müssen zunächst in den Zeilen 2 bis 12 und 13 bis 21 die Randbedingungen der zu konstruierenden Klothoide festgelegt werden. Dies sind insbesondere die Anfangs- bzw. Endrichtung sowie der zu erreichende Radius. In den Zeilen 22 bis 24 erfolgt schließlich die Berechnung der Parameter Richtungsänderung (temporäre Variable), Kurvenlänge und Klothoidenparameter nach den in Kapitel 4.3.2 hergeleiteten Gleichungen 4.13 bis 4.15.

Die folgenden Hilfsfunktionen können dem Anhang entnommen werden:

Funktion	Algorithmus
`angleOfVectors`	29
`tangentDirection`	30
`tangentVector`	31
`tangentVectorFromDirection`	32

Algorithmus 10 Eigenschaften einer Klothoide festlegen

Input

previous	Eigenschaften des vorherigen Elements; beinhaltet *previousType*, *previousDirection*, *previousCenter*, *previousIsCCW*, *previousRadius*
next	Eigenschaften des nachfolgenden Elements, beinhaltet *nextType*, *nextDirection*, *nextRadius*, *nextIsCCW*
startPoint	Startpunkt des Elements

Return

element	Eigenschaften des betrachteten Elements; beinhaltet *elementDirection*, *elementRadiusClothoidStart*, *elementRadiusClothoidEnd*, *elementIsCCW*, *elementLength*, *elementClothoidparameterA*

1: **function** obtainClothoid(*previous*, *next*, *startPoint*)
2: **if** *previous.Type* = 'straight' **then** ▷ vorheriges Element ist Gerade
3: *tangentStart* ← tangentVectorFromDirection(*previous.Direction*) ▷ Tangentenvektor am Start
4: *element.Direction* ← *previous.Direction*
5: *element.RadiusClothoidStart* ← *nan* ▷ kein Radius, da Gerade
6: **else** ▷ vorheriges Element ist Kreis
7: *tangentStart* ← tangentVector(*previous.Center*, *startPoint*, *previous.IsCCW*)
8: *element.Direction* ← sign(*tangentStart*.y) · angleOfVectors([1; 0], *tangentStart*)
9: *element.RadiusClothoidStart* ← *previous.Radius*
10: *radius* ← *previous.Radius* ▷ für Berechnung, ohne Zuordnung 'Start'
11: *element.IsCCW* ← *previous.IsCCW*
12: **end if**
13: **if** *next.Type* = 'straight' **then** ▷ nächstes Element ist Gerade
14: *tangentEnd* ← tangentVectorFromDirection(*next.Direction*) ▷ Tangentenvektor am Ende
15: *element.RadiusClothoidEnd* ← *nan* ▷ kein Radius, da Gerade
16: **else** ▷ nächstes Element ist Kreis
17: *tangentStart* ← tangentVectorFromDirection(*next.Direction*)
18: *element.RadiusClothoidEnd* ← *next.Radius*
19: *radius* ← *next.Radius* ▷ für Berechnung, ohne Zuordnung 'Ende'
20: *element.IsCCW* ← *next.IsCCW*
21: **end if**
22: *tau* ← angleOfVectors(*tangentStart*, *tangentEnd*) ▷ Richtungsänderung der Klothoide in Grad
23: *element.Length* ← $2 \cdot radius \cdot (tau * pi/180)$ ▷ Kurvenlänge der Klothoide
24: *element.ClothoidparameterA* ← sqrt($radius \cdot element.length$)
25: **return** *element*
26: **end function**

Kapitel 6

Evaluierung

Die Erhebung von Evaluierungsdaten erfolgt in zwei Schritten: Zunächst werden in Kapitel 6.1 Testdaten in perfekter geometrischer Lage als Eingabewerte in das Programm gegeben. Im zweiten Abschnitt 6.2 erfolgt die Verarbeitung gezeichneter Eingabepunkte. Im Teilkapitel 6.3 werden die Ergebnisse hinsichtlich ihrer Qualität beurteilt.

6.1 Berechnete Eingabepunkte

Zunächst erfolgt eine Überprüfung mit extern generierten Testdatensätzen, ob eine technisch korrekte Trassierung als Ergebnis zurückgegeben wird. Die Eingabepunkte wurden in AutoCAD gezeichnet und exportiert. Geraden und Kreisbögen konnten dort direkt gezeichnet werden; für Klothoiden hingegen war die Zuhilfenahme einer Excel-Tabelle erforderlich, um dort Auswertepunkte entlang einer Klothoide zu berechnen. Tabelle 6.1 zeigt im linken Teil die gewählten Trassierungsparameter, in der rechten Hälfte sind die Ergebnisse der Identifikation des implementierten Algorithmus dargestellt.

Berechnete Testdaten			Interpretation		
Element	Länge	R bzw. A	Element	Länge	R bzw. A
Kreisbogen	355	400	Kreisbogen	312,47	400,96
Klothoide	127	225	Klothoide	190,64	276,47
Gerade	500	-	Gerade	454,87	-
Klothoide	145	225	Klothoide	175,69	248,10
Kreisbogen	555	350	Kreisbogen	538,83	350,35

Tabelle 6.1: Vergleich der Interpretation bei berechneten Eingabepunkten.

Abbildung 6.1 zeigt den aus der B-Spline berechneten Krümmungsverlauf als Punktlinie, als auch der Ergebnis nach dem Glätten durch gleitenden Mittelwert als Volllinie. Abbildung 6.2 zeigt den Verlauf der Krümmungsänderung, ebenfalls vor dem glätten durch den gleitenden Mittelwert (Punktlinie) und nach dem Glätten (Volllinie).

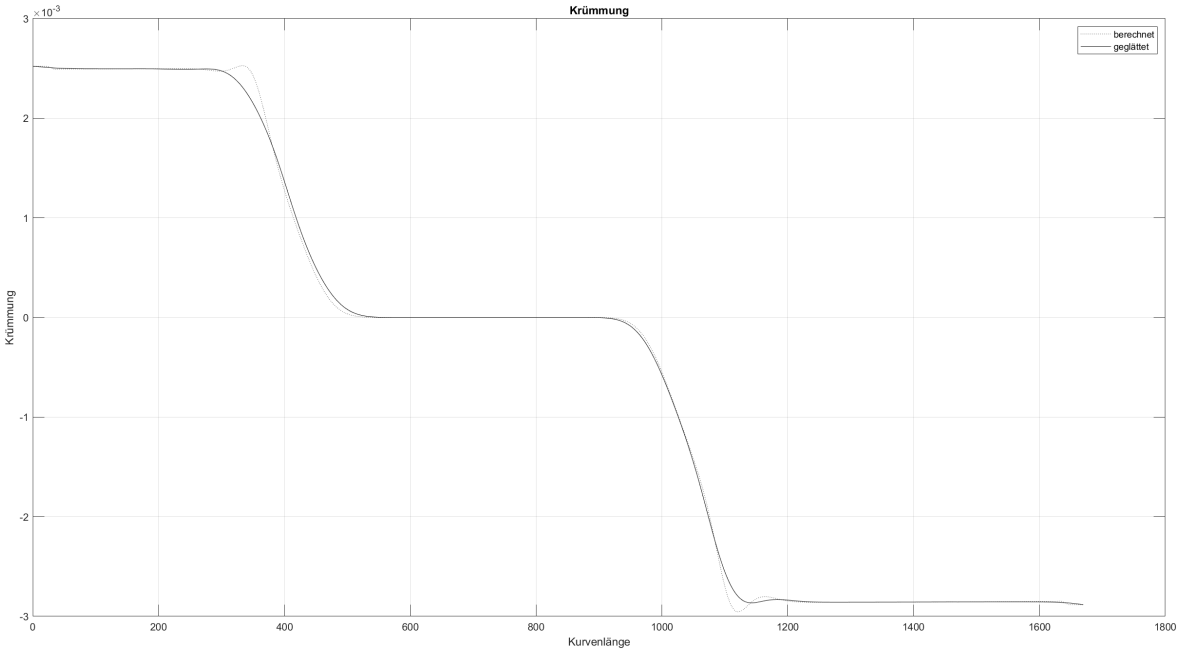

Abbildung 6.1: Krümmungsverlauf vor und nach Glätten der Eingabepunkte in perfekter Lage.

Eine grafische Darstellung des Ergebnisses zeigt Abbildung 6.3. Hierbei ist in weiß kaum sichtbar die Trassierung in perfekter Lage und das Resultat der Interpretation in orange dargestellt. Die Endpunkte der interpretierten Elemente sind mit kleinen Kreisen gekennzeichnet.

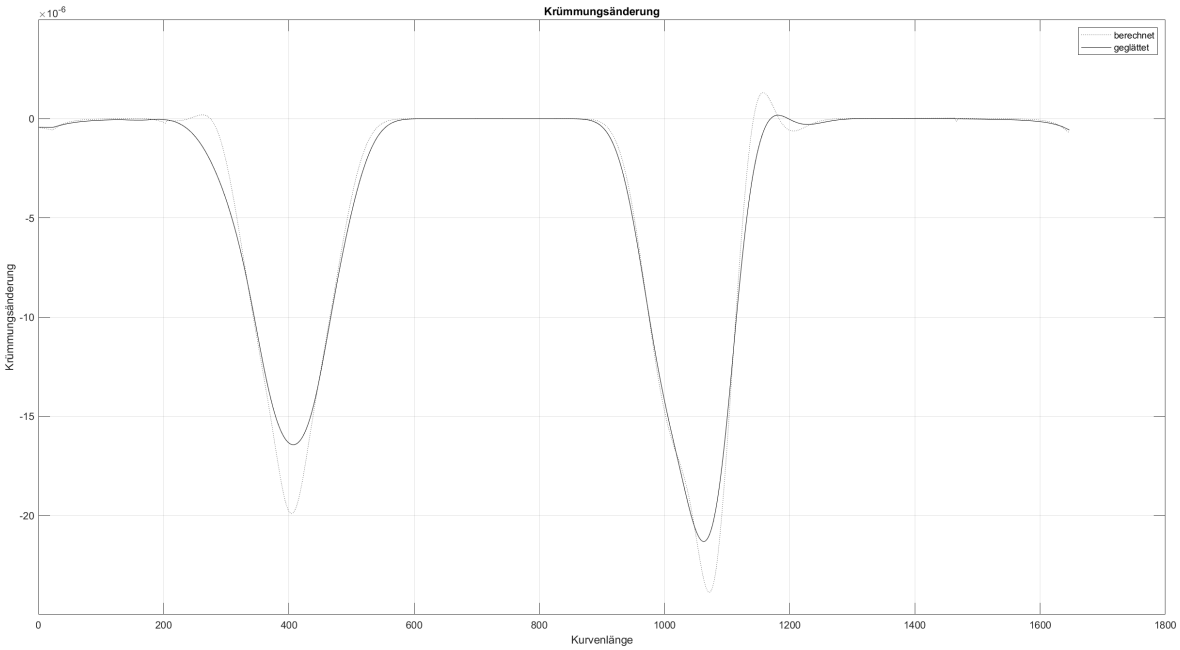

Abbildung 6.2: Verlauf der Krümmungsänderung vor und nach Glätten der Eingabepunkte in perfekter Lage.

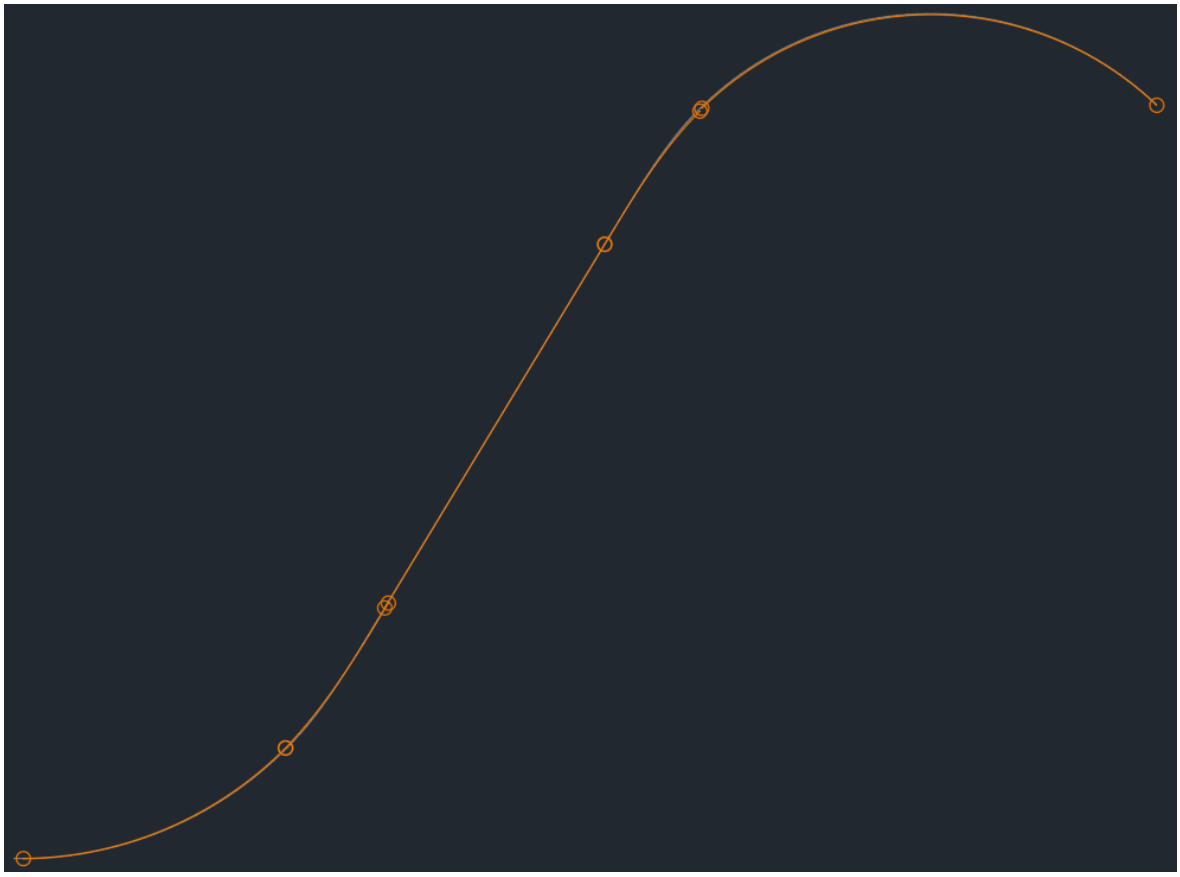

Abbildung 6.3: Grafische Darstellung der berechneten perfekten Testdaten (weiß) und der interpretierten Elemente (orange).

6.2 Gezeichnete Eingabepunkte

Da die Zeichenfunktion in OIP aufgrund der Ausführungen in Kapitel 5.1 derzeit nicht verfügbar war, wurde als Ersatz in AutoCAD der Befehl „Skizze“ herangezogen. Bei diesem ist es Möglich, eine Freihandlinie zu zeichnen. Diese wird in Liniensegmenten erzeugt, wobei eine Linie in etwas so lang ist, wie der als „Inkrement“ eingestellte Wert.

Zwecks Vergleichbarkeit wurde die oben in Kapitel 6.1 bereits beschriebene Trassierung mit dem Inkrementwert 10 nachgezeichnet. Abbildung 6.4 zeigt die ausgewählte Skizze in oranger Linienfarbe, die korrespondierende perfekte Trassierung ist in weiß mit dargestellt. Das Ergebnis des erstellten Programms ist in Abbildung 6.5 dargestellt.

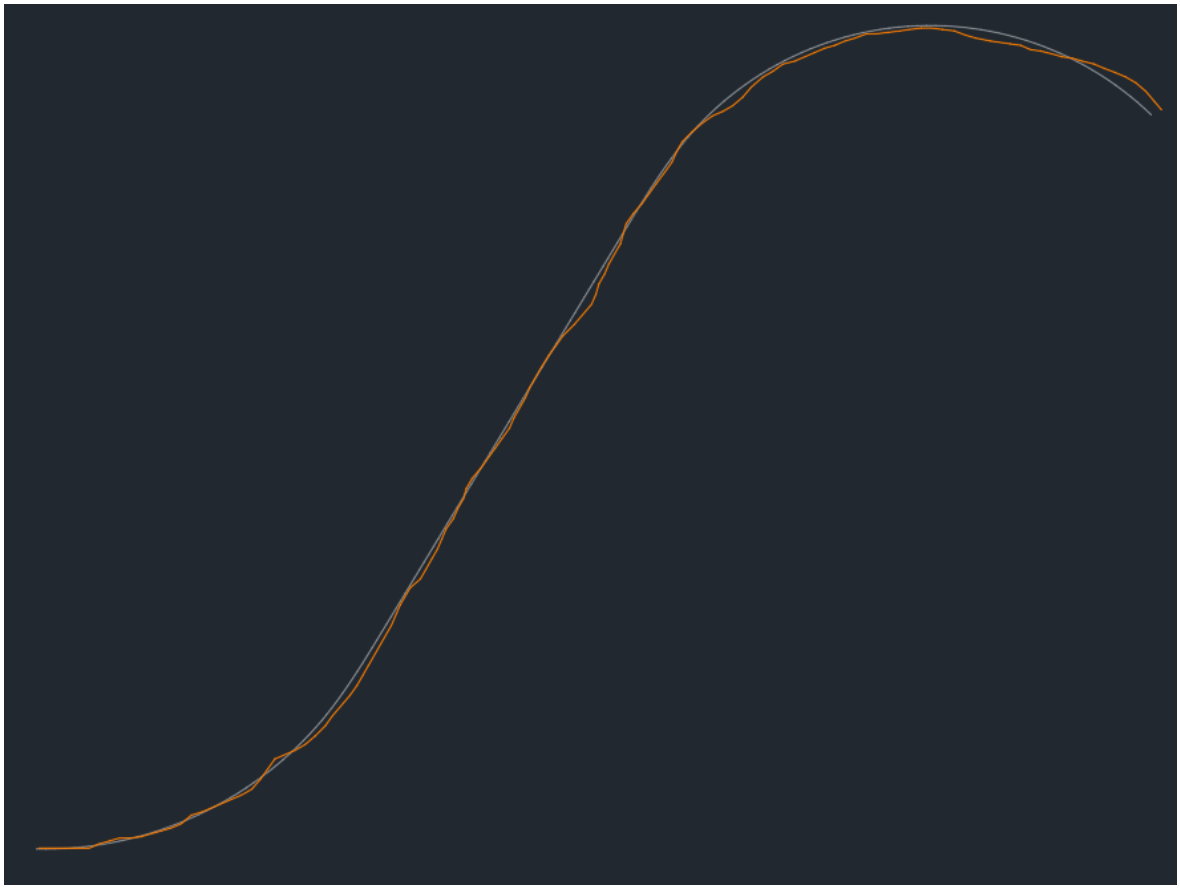

Abbildung 6.4: Darstellung der als Skizze gezeichneten Trassierung in orange.

Analog den Darstellungen in Kapitel 6.1 zeigen die Abbildungen 6.6 und 6.7 den Krümmungsverlauf und den den Verlauf der Krümmungsänderung.

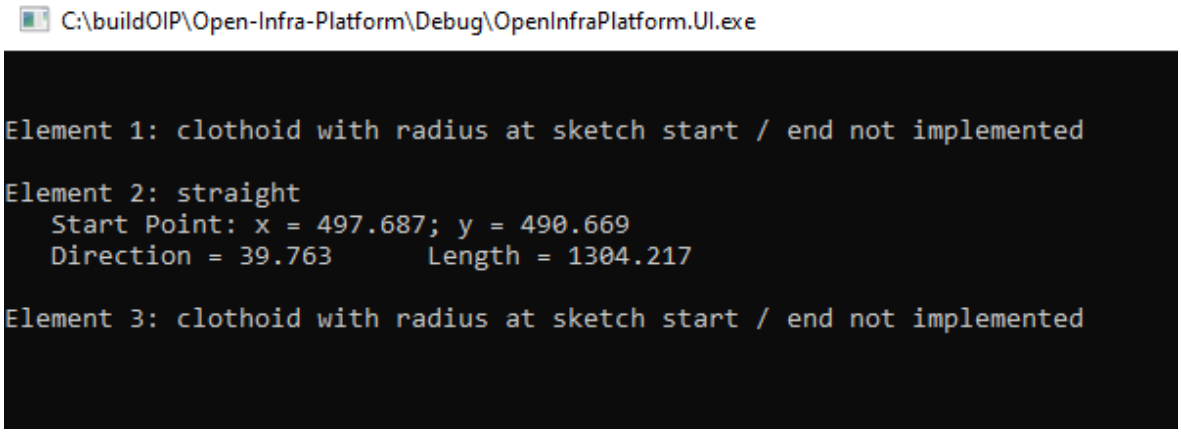

Abbildung 6.5: Konsolenausgabe des Ergebnisses der gezeichneten Testdaten.

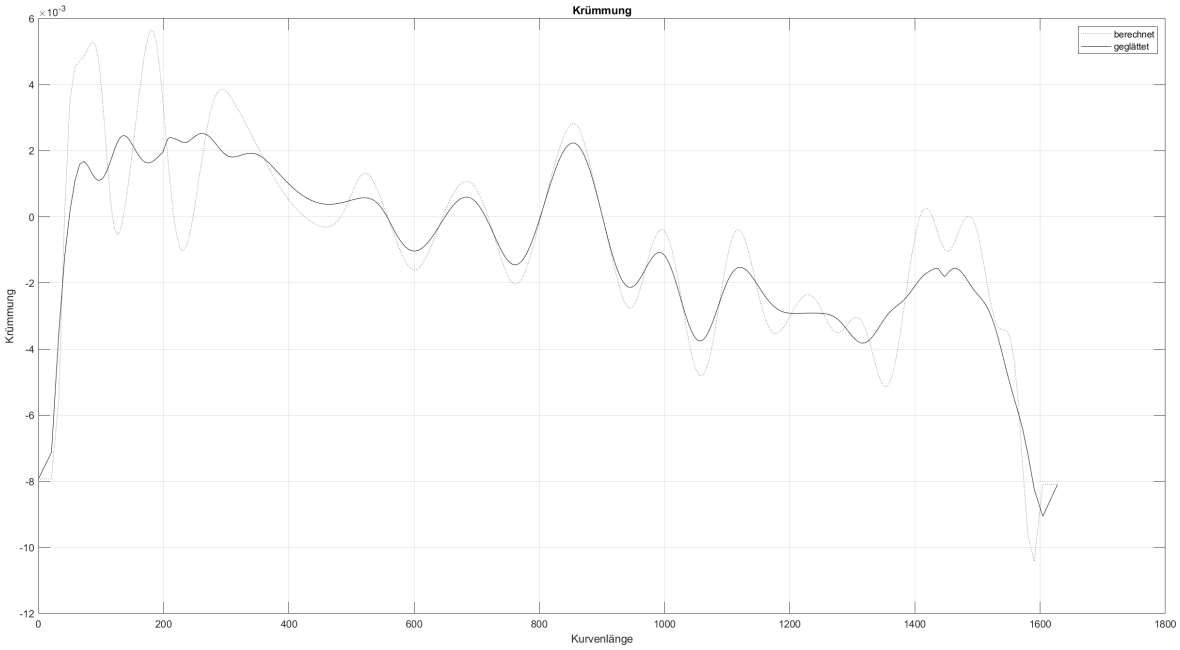

Abbildung 6.6: Krümmungsverlauf vor und nach Glätten der per Skizze gezeichneten Eingabepunkte.

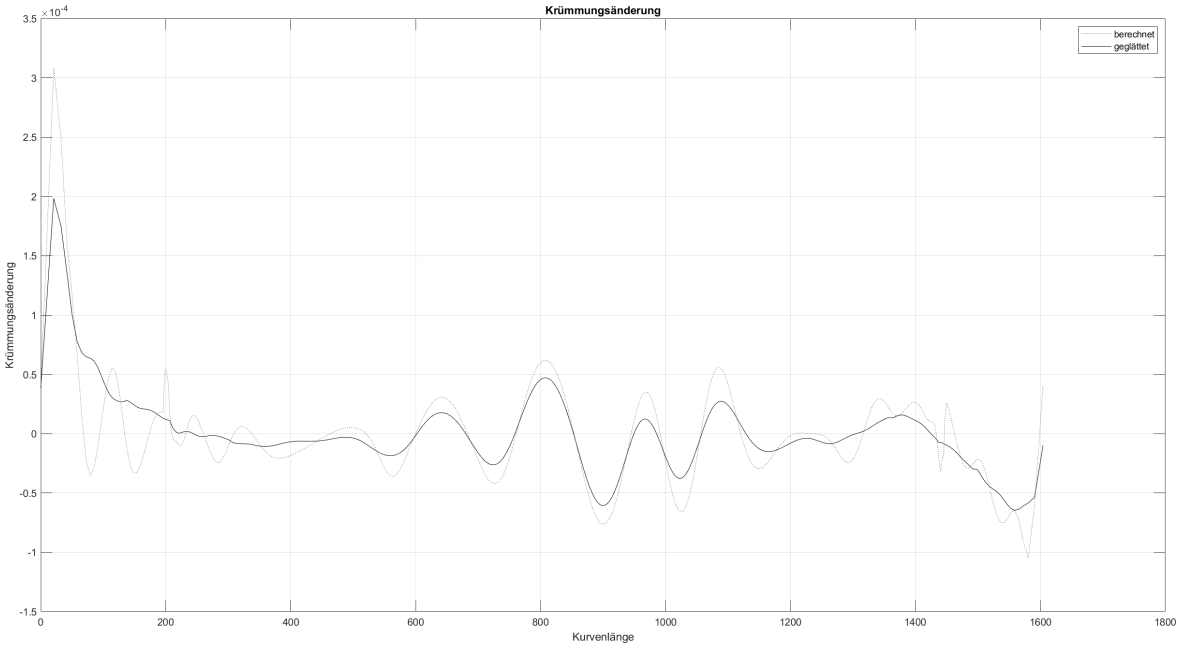

Abbildung 6.7: Verlauf der Krümmungsänderung vor und nach Glätten der per Skizze gezeichneten Eingabepunkte.

6.3 Beurteilung

Zusammenfassend kann festgestellt werden, dass die Glättung mittels dem gleitenden Mittelwert insbesondere Störungen auf kurzer Länge gut kompensieren kann, beispielsweise erkennbar in Abbildung 6.6 im Abschnitt der Kurvenlänge von ca. 50 bis 400 m. Ebenso ist festzuhalten, dass charakteristische Punkte der Verläufe teilweise etwas verschwimmen, beispielsweise in Abbildung 6.1 an der Kurvenlänge von ca. 350 m.

Entsprechend der Angaben in Tabelle 6.1 konnten bei den Eingabepunkten in perfekter Lage die Radien der Kreisbögen sehr präzise ermittelt werden. Die Abweichung der Elementlängen ist auf die unscharfen charakteristischen Punkte infolge des gleitenden Mittelwerts, als auch den Betrag der in Kapitel 4.3.2 definierten Toleranzgrenze zurückzuführen. Dies wird deutlich, da die Längen der Klothoiden größer sind. Abbildung 6.3 zeigt, dass die interpretierten Elemente dennoch sehr deckungsgleich mit ursprünglich perfekten Lage positioniert wurde.

Bei der Auswertung der per Hand gezeichneten Trassierung konnten die Elemente nicht korrekt erkannt werden. Bereits im Verlauf der Krümmung sind weder die Gerade in der Mitte, noch die Kreisbögen am Beginn und Ende deutlich ausgeprägt. Die deutlichen Störungen im Bereich von 600 bis 1200 m konnten lediglich aufgrund der definierten Mindestelementlänge kompensiert werden.

Kapitel 7

Zusammenfassung

7.1 Diskussion

Zusammenfassend kann entsprechend der Resultate in Kapitel 6.1 mit Erläuterungen in Kapitel 6.3 festgestellt werden, dass der vorgestellte Algorithmus im Fall von perfekten Eingabedaten sehr gute Ergebnisse liefert. Entsprechend scheint die vorgestellte Methodik grundsätzlich zielführend zu sein. Entsprechend Abbildung 6.3 wurde die Hauptanforderung der kontinuierlichen Trassierung mit stetiger Krümmung erfüllt werden.

Noch nicht erwähnt wurde die geringe Abweichung der Endpunkte in Abbildung 6.3. Dies betrifft den Übergang von der linken Klothoide zur Geraden, sowie von der rechten Klothoide zum Kreisbogen. Bezüglich der Zeichenrichtung, welche hier von links nach rechts ausgerichtet ist, tritt die Abweichung immer am Ende der Klothoidensegmente auf. Analog zur Konstruktion der perfekten Trasse wurden diese auch hier beim Nachzeichnen der interpretierten Elemente mit Hilfe einer Berechnung von Stützpunkten in Excel festgelegt. Folglich wird die Ungenauigkeit an dieser Stelle der Reihenentwicklung (vgl. Gl. 2.4) zugeordnet.

Die nur ungenügenden Ergebnisse der gezeichneten Eingabedaten aus Kapitel 6.2 zeigen jedoch, dass die Glättung der Daten noch optimiert werden muss. Unter anderem kann dies auf den gleitenden Mittelwert mit Fenstergröße aus dem Semivariogramm zurückgeführt werden. Nach der Beschreibung in Kapitel 2.3 ist dieser von Gikas & Stratakos (2012) vorgestellte und verwendete Filter ursprünglich für fraktales Rauschen von Messdaten mit Gaußscher Charakteristik vorgesehen. Diese Art des Rauschens tritt jedoch im konkreten Testfall nicht auf. Trotz des nicht idealen Einsatzes dieses Filters verschlechtert er zumindest die Verläufe von Krümmung und Krümmungsänderung nicht signifikant.

Ebenfalls konnte beobachtet werden, dass die Qualität der Ergebnisses stark von der Qualität der Eingabe abhängt. Folglich kann ein interaktive Trassierungserkennung lediglich so gut sein, wie angefertigte Zeichnung. Nehmen die Störungen der Information – also die zeich-

nerische Abweichung vom gewünschten respektive idealen Verlauf – zu sehr überhand, so erschwert sich die korrekte Erkennung der Elemente zunehmend.

Die in Kapitel 2.1.2 vorgestellten Grenzwerte der Trassierung konnten bedingt eingehalten werden. Die dort erläutertet Mindestelementlänge konnte im Rahmen des Entwurfs integriert werden. Der größtmögliche theoretische Radius von 25.000 m konnte jedoch als Abgrenzung zwischen Gerade und Kreisbogen nicht angewendet werden. Der korrespondierende Krümmungswert von $4 \cdot 10^{-5}$ als Toleranzgrenze ist bezüglich der in Abbildung 6.6 dargestellten Krümmung bedeutend zu nieder. Ggf. würden sich hier jedoch bei einem andern Zeichenmaßstab mit einer größeren Inkrementweite bessere Bedingungen einstellen.

Eine weitere Einschränkung der Anwendbarkeit stellt die Implementierung der Klothoide dar, welche aktuell lediglich die Form der einfachen Klothoide umfasst. Derzeit werden beispielsweise keine Wendeklothoiden unterstützte, welche jedoch insbesondere bei der Trassierung von Straßen eine entscheidende Rolle spielen.

7.2 Weiterführende Arbeiten

Ein wesentlicher Bestandteil für hierauf aufbauende Arbeiten kann die Weiterentwicklung und Erprobung der B-Splinekurve als Filtertechnik für grafische Eingabepunkte darstellen. In diesem Zusammenhang kann auch der hier eingesetzte Filter des gleitenden Mittelwerts verbessert oder ersetzt werden.

In Algorithmus 8 auf Seite 40 wurden einige Sonderfälle der Klothoide ausgeschlossen. Insbesondere um die Anwendbarkeit in der Straßenplanung zu erhöhen, könnten diese noch ergänzend Implementiert werden.

Auch die Anpassung zur Erkennung der vertikalen Trassierung ist ein möglicher Ansatzpunkt, um die Funktionalität zu erweitern. Analog dazu können neben der vorhanden Klothoide auch andere Übergangskurven implementiert werden.

Ebenso können die erkannten Elemente hinsichtlich der noch nicht einbezogenen Trassierungsgrenzwerte geprüft werden. Hierbei bietet sich zunächst an, die Elemente, bei denen die Mindest- und Maximalwerte nicht eingehalten sind, beispielsweise farblich zu markieren. Darauf aufbauend könnten diese Elemente dann ggf. automatisch so bearbeitete werden, dass die Einhaltung der Grenzwerte gewährleistet wird.

Eine Erweiterung in OIP ist ebenso denkbar. Insbesondere die direkte grafische Eingabe oder das Erzeugen und Speichern von IFC-Dateien stellen eine große Bereicherung für das bereits entwickelte Teilprogramm dar.

Anhang A

Pseudocode zum Kapitel 5 Implementierung

A.1 Approximation der Punktfolge

`obtainKnotArrayOpenUniform` in Algorithmus 11 erzeugt den zur Anzahl der Eingabepunkte passenden offenen gleichmäßigen Knotenvektor.

Algorithmus 11 Erzeugung des offenen gleichmäßigen Knotenvektors

Input

$nControlPoints$	Anzahl der Kontrollpunkte der B-Splinekurve
$order$	Ordnung der B-Splinekurve

Return

$knotArray$	offener gleichmäßiger Knotenvektor

```
1:  function obtainKnotArrayOpenUniform(nControlPoints, order)
2:      for i ← 1 : order do
3:          knotArray(i) ← 0
4:      end for
5:      for i ← order + 1 : nControlPoints do
6:          knotArray(i) ← i − order
7:      end for
8:      for i ← nControlPoints + 1 : nControlPoints + order do
9:          knotArray(i) ← nControlPoints − order + 1
10:     end for
11:     return knotArray
12: end function
```

Für alle Auswertestellen t berechnet der in Algorithmus 12 dargestellte Code die Koordinaten der B-Splinekurve.

Algorithmus 12 Berechnung der vollständigen B-Splinekurve

Input

order	Ordnung
knotArray	Knotenvektor
controlPoints	Kontrollpunkte
nCurvePoints	Anzahl der Auswertepunkte

Return

bsplinePoints	Auswertepunkte der B-Splinekurve

```
1:  function computeBSplineCurveWithKnots(order, knotArray, controlPoints,
    nCurvePoints)
2:      knotStart ← knotArray(order)
3:      knotEnd ← knotArray(controlPoints.size + 1)
4:      step ← (knotEnd − knotStart)/controlPoints.size − 1
                                          ▷ Intervallgröße der Auswertestellen t
5:      t ← knotStart
6:      for i ← 1 : nCurvePoints do
7:          bsplinePoints(i) ←computePointOfBSpline(order, t, controlPoints, knotArray)
                                ▷ Berechnung des i-ten Auswertepunkts der B-Splinekurve
8:          t ← t + 1                          ▷ Vorrücken zur nächsten Auswertestelle
9:      end for
10:     return bsplinePoints
11: end function
```

Algorithmus 13 berechnet die Koordinaten der B-Splinekurve lediglich für einen konkreten Wert von t.

Algorithmus 13 Berechnung eines Punkts der B-Splinekurve

Input

order	Ordnung
t	Auswertestelle
controlPoints	Kontrollpunkte
knotArray	Knotenvektor

Return

point	ein Auswertepunkt der B-Splinekurve

```
1: function computePointOfBSpline(order, t, controlPoints, knotArray)
2:     nControlPoints ← controlPoints.size                ▷ Anzahl der Kontrollpunkte
3:     basisFuncs ← computeBSplineBasisFunctions(
           order, t, nControlPoints, knotArray)    ▷ B-Splinebasisfunktionen der Stelle t
4:     for i ← 1 : nControlPoints do
5:         point ← point + basisFuncs(i) · controlPoints(i)
6:     end for                                    ▷ Berechnung des Kurvenpunkts
7:     return point
8: end function
```

Funktion `obtainBasisFunctionFirstOrder` (Algorithmus 14) zum Berechnen der B-Spline Basisfunktionen.

Algorithmus 14 Initialisieren der Basisfunktionen für Ordnung $k = 1$

Input

t	Auswertestelle
$nBasisFuncs$	Anzahl der Basisfunktionen
$knots$	Knotenvektor

Return

$basisFuncs$	Basisfunktionen der Ordnung $k = 1$

```
1:  function obtainBasisFunctionFirstOrder(t, nBasisFuncs, knotArray)
2:      for i ← 1 : nBasisFuncs do
3:          if t ≥ knots(i) and t < knots(i + 1) and knots(i) < knots(i+) then
4:              basisFuncs(i) ← 1
5:          else
6:              basisFuncs(i) ← 0
7:          end if
8:      end for
9:      return basisFuncs
10: end function
```

Funktion `obtainBasisFunctionNextOrder` (Algorithmus 15) zum Berechnen der B-Spline Basisfunktionen.

Algorithmus 15 Berechnung der Basisfunktionen der nächsthöheren Ordnung

Input

$order$	Ordnung
t	Auswertestelle
$knots$	Knotenvektor
$basisFuncs$	Basisfunktionen der Ordnung k

Return

$basisFuncs$	Basisfunktionen der Ordnung $k+1$

1: **function** obtainBasisFunctionNextOrder($order$, t, $knots$, $basisFuncs$)
2: **for** $i \leftarrow 1$ **:** $basisFuncs$.size $- order$ **do**
3: **if** $basisFuncs(i) = 0$ **or** $knots(i + order) = knots(i)$ **then**
4: $firstSum \leftarrow 0$
5: **else**
6: $firstSum \leftarrow (t - knots(i))/(knots(i + order) - knots(i))$
7: **end if** ▷ erster Summand der Basisfunktion
8: **if** $basisFuncs(i + 1) = 0$ **or** $knots(i + order + 1) = knots(i + 1)$ **then**
9: $secondSum \leftarrow 0$
10: **else**
11: $secondSum \leftarrow (knots(i + order + 1) - t)$
$/(knots(i + order + 1) - knots(i + 1))$
12: **end if** ▷ zweiter Summand der Basisfunktion
13: $basisFuncs(i) = firstSum \cdot basisFuncs(i) + secondSum \cdot basisFuncs(i + 1)$
14: **end for**
15: **return** $basisFuncs$
16: **end function**

A.2 Ermittlung der Krümmung und Krümmungsänderung

Algorithmus 16 Berechnung eines Punkts der ersten Ableitung

Input

order	Ordnung
t	Auswertestelle
controlPoints	Kontrollpunkte
knotArray	Knotenvektor

Return

point	ein Auswertepunkt der ersten Ableitung

```
1: function computePointOfDerivOne(order, t, controlPoints, knotArray)
2:     nControlPoints.size()                          ▷ Anzahl der Kontrollpunkte
3:     basisFuncs ← computeDerivOneBasisFunctions(
           order, t, nControlPoints, knotArray)       ▷ Basisfunktionen für erste Ableitung
4:     for i ← 1 : nControlPoints do
5:         point ← point + basisFuncs(i) · controlPoints(i)
6:     end for                                        ▷ Berechnung des Auswertepunkts
7:     return point
8: end function
```

Algorithmus 17 Berechnung eines Punkts der zweiten Ableitung

Input

$order$	Ordnung
t	Auswertestelle
$controlPoints$	Kontrollpunkte
$knotArray$	Knotenvektor

Return

$point$	ein Auswertepunkt der ersten Ableitung

```
1: function computePointOfDerivTwo(order, t, controlPoints, knotArray)
2:     nControlPoints.size()                          ▷ Anzahl der Kontrollpunkte
3:     basisFuncs ← computeDerivTwoBasisFunctions(
           order, t, nControlPoints, knotArray)       ▷ Basisfunktionen für zweite Ableitung
4:     for i ← 1 : nControlPoints do
5:         point ← point + basisFuncs(i) · controlPoints(i)
6:     end for                                        ▷ Berechnung des Auswertepunkts
7:     return point
8: end function
```

Algorithmus 18 Berechnung der Basisfunktionen der ersten Ableitung

Input

$order$	Ordnung
t	Auswertestelle
$nControlPoints$	Kontrollpunkte
$knotArray$	Knotenvektor

Return

$basisFuncsDerivOne$	Basisfunktionen der ersten Ableitung

1: **function** computeDerivOneBasisFunctions($order$, t, $nControlPoints$, $knotArray$)
2: $nBasisFuncs \leftarrow order - 1 + nControlPoints$ ▷ Anzahl der Basisfunktionen
3: $basisFuncsBSpline \leftarrow$ obtainBasisFunctionFirstOrder(t, $nBasisFuncs$, $knotArray$)
▷ Basisfunktionen des B-Splines der Ordnung $k = 1$
4: $basisFuncsDerivOne \leftarrow$ zerosVector($nBasisFuncs$)
▷ Basisfunktionen der ersten Ableitung der Ordnung $k = 1$
5: **for** $k \leftarrow 2 : order$ **do**
6: $basisFuncsDerivOne \leftarrow$ obtainBasisFunctionDerivOneNextOrder(
$order$, t, $knotArray$, $basisFuncsBSpline$, $basisFuncsDerivOne$)
7: $basisFuncsBSpline \leftarrow$ obtainBasisFunctionNextOrder(
$order$, t ,$knotArray$, $basisFuncsBSpline$)
8: **end for** ▷ Ordnung der Basisfunktionen erhöhen
9: $basisFuncsDerivOne \leftarrow basisFuncsDerivOne(1 : nControlPoints)$
▷ Kopieren der relevanten Basisfunktionen
10: **return** $basisFuncsDerivOne$
11: **end function**

Algorithmus 19 Berechnung der Basisfunktionen der ersten Ableitung in der nächsthöheren Ordnung

Input

$order$	Ordnung
t	Auswertestelle
$knots$	Knotenvektor
$basisFuncsBSpline$	Basisfunktionen der Ordnung k
$basisFuncsDerivOne$	Basisfunktionen der ersten Ableitung der Ordnung k

Return

$basisFuncsDerivOne$	Basisfunktionen der ersten Ableitung der Ordnung $k+1$

1: **function** obtainBasisFunctionDerivOneNextOrder($order$, t, $knots$, $basisFuncsBSpline$, $basisFuncsDerivOne$)
2: **for** $i \leftarrow 1$ **:** $basisFuncsDerivOne$.size $- order$ **do**
3: **if** $knots(i + order) = knots(i)$ **then**
4: $firstSum \leftarrow 0$
5: **else**
6: $firstSum \leftarrow$ $(basisFuncsBSpline(i) + (t - knots(i)) \cdot basisFuncsDerivOne(i))$ $/(knots(i + order) - knots(i))$ ▷ erster Summand der Basisfunktion
7: **end if**
8: **if** $knots(i + order + 1) = knots(i + 1)$ **then**
9: $secondSum \leftarrow 0$
10: **else**
11: $secondSum \leftarrow ((knots(i + order + 1) - t) \cdot basisFuncsDerivOne(i + 1)$ $-basisFuncsBSpline(i + 1))/(knots(i + order + 1) - knots(i + 1))$
12: **end if** ▷ zweiter Summand der Basisfunktion
13: $basisFuncsDerivOne(i) = firstSum + secondSum$
14: **end for**
15: **return** $basisFuncsDerivOne$
16: **end function**

Algorithmus 20 Berechnung der Basisfunktionen der zweiten Ableitung

Input

$order$	Ordnung
t	Auswertestelle
$nControlPoints$	Kontrollpunkte
$knotArray$	Knotenvektor

Return

$basisFuncsDerivTwo$	Basisfunktionen der zweiten Ableitung

1: **function** computeDerivTwoBasisFunctions($order$, t, $nControlPoints$, $knotArray$)
2: $nBasisFuncs \leftarrow order - 1 + nControlPoints$ ▷ Anzahl der Basisfunktionen
3: $basisFuncsBSpline \leftarrow$ obtainBasisFunctionFirstOrder(t, $nBasisFuncs$, $knotArray$)
▷ Basisfunktionen des B-Splines der Ordnung $k = 1$
4: $basisFuncsDerivOne \leftarrow$ zerosVector($nBasisFuncs$)
5: $basisFuncsDerivTwo \leftarrow$ zerosVector($nBasisFuncs$)
▷ Basisfunktionen der ersten & zweiten Ableitung der Ordnung $k = 1$
6: **for** $k \leftarrow 2 : order$ **do**
7: $basisFuncsDerivTwo \leftarrow$ obtainBasisFunctionDerivTwoNextOrder(
$order$, t, $knotArray$, $basisFuncsDerivOne$, $basisFuncsDerivTwo$)
8: $basisFuncsDerivOne \leftarrow$ obtainBasisFunctionDerivOneNextOrder(
$order$, t ,$knotArray$, $basisFuncsBSpline$, $basisFuncsDerivOne$)
9: $basisFuncsBSpline \leftarrow$ obtainBasisFunctionNextOrder(
$order$, t ,$knotArray$, $basisFuncsBSpline$)
10: **end for** ▷ Ordnung der Basisfunktionen erhöhen
11: $basisFuncsDerivTwo \leftarrow basisFuncsDerivTwo(1 : nControlPoints)$
▷ Kopieren der relevanten Basisfunktionen
12: **return** $basisFuncsDerivTwo$
13: **end function**

Algorithmus 21 Berechnung der Basisfunktionen der zweiten Ableitung in der nächsthöheren Ordnung

Input

$order$	Ordnung
t	Auswertestelle
$knots$	Knotenvektor
$basisFuncsDerivOne$	Basisfunktionen der ersten Ableitung der Ordnung k
$basisFuncsDerivTwo$	Basisfunktionen der zweiten Ableitung der Ordnung k

Return

$basisFuncsDerivTwo$	Basisfunktionen der zweiten Ableitung der Ordnung $k+1$

1: **function** obtainBasisFunctionDerivTwoNextOrder($order$, t, $knots$, $basisFuncsBSpline$, $basisFuncsDerivOne$, $basisFuncsDerivTwo$)
2: **if** $order = 1$ **then**
3: **for** $i \leftarrow 1$ **:** $basisFuncsDerivOne$.size $- order$ **do**
4: $basisFuncsDerivTwo(i) \leftarrow 0$
5: **end for**
6: **else**
7: **for** $i \leftarrow 1$ **:** $basisFuncsDerivOne$.size $- order$ **do**
8: **if** $knots(i + order) = knots(i)$ **then**
9: $firstSum \leftarrow 0$
10: **else**
11: $firstSum \leftarrow$ $(2 \cdot basisFuncsDerivOne(i) + (t - knots(i)) \cdot basisFuncsDerivTwo(i))$ $/(knots(i + order) - knots(i))$ ▷ erster Summand der Basisfunktion
12: **end if**
13: **if** $knots(i + order + 1) = knots(i + 1)$ **then**
14: $secondSum \leftarrow 0$
15: **else**
16: $secondSum \leftarrow ((knots(i + order + 1) - t) \cdot basisFuncsDerivTwo(i + 1)$ $-2 \cdot basisFuncsDerivOne(i + 1))/(knots(i + order + 1) - knots(i + 1))$
17: **end if** ▷ zweiter Summand der Basisfunktion
18: $basisFuncsDerivTwo(i) = firstSum + secondSum$
19: **end for**
20: **end if**
21: **return** $basisFuncsDerivTwo$
22: **end function**

Algorithmus 22 zeigt den Pseudocode zur Bestimmung der Fenstergröße, welche beim gleitenden Mittelwert Anwendung findet.

Algorithmus 22 Bestimmung der Fenstergröße des gleitenden Mittelwerts

Input
data Vektor mit Datenwerte
Return
range Range des Semivariogramms

```
1:  function variogrammGetRange(data)
2:      for h ← 1 : data.size−1 do
3:          gamma(h) ← 0
4:          N ← data.size−h                          ▷ Anzahl der Datenpaare
5:          for i ← 1 : N do
6:              gamma(h) ← gamma(h) + (data(i) − data(i + h))^2
7:          end for
8:          gamma(h) ← gamma(h)/(2 · N)              ▷ Semivariogrammvariable
9:      end for
10:     for h ← 1 : gamma.size −1 do
11:         if gamma(h) ≥ gamma(h + 1) then
12:             range = h                            ▷ range bei Hochpunkt des Verlaufs
13:             break for
14:         end if
15:         range = h + 1                            ▷ range, falls kein Hochpunkt gefunden
16:     end for
17:     return range
18: end function
```

Algorithmus 23 zeigt die Umsetzung der numerischen Ableitung.

Algorithmus 23 Nummerische Ableitung

Input
x x-Werte
y y-Werte
Return
dy y-Werte der Ableitung

```
1: function numericDerivative(x, y)
2:     for i ← 1 : x.size −1 do
3:         dy(i) ← (y(i + 1) − y(i))/(x(i + 1) − x(i))
4:     end for
5:     return dy
6: end function
```

A.3 Identifikation der Trassierungselemente

Die Funktion `indicateCurvatureChange` (Alg. 24) prüft an jeder Auswertestelle der Krümmungsänderung den Wert gegen den durch `obtainThreshold` (Alg. 25) definierten Schwellenwert. Rückgabewert ist 0, wenn der Wert im Rahmen des Schwellenwerts nahe null liegt, und 1, wenn der Wert außerhalb des Toleranzfensters liegt.

Algorithmus 24 Betrag der Krümmungsänderung prüfen

Input
curvatureChange Vektor der Krümmungsänderung der B-Splinekurve
Return
indicator Vektor des Prüfergebnisses der Krümmungsänderung
curvatureZero Schwellenwert der Krümmungsänderung

```
1: function indicateCurvatureChange(curvatureChange)
2:     curvatureZero ← obtainThreshold(curvatureChange)
                                          ▷ Schwellenwert der Krümmungsänderung
3:     for i ← 1 : curvatureChange.size do
4:         if abs(curvatureChange(i)) ≤ curvatureZero then
5:             indicator(i) ← 0                          ▷ Krümmungsänderung ≈ 0
6:         else
7:             indicator(i) ← 1                          ▷ Krümmungsänderung ≠ 0
8:         end if
9:     end for
10:    return [indicator, curvatureZero]
11: end function
```

Algorithmus 25 Schwellenwert als Betrag von Min.- & Max.-Wert bestimmen

Input
data Vektor mit Datenwerte
Return
threshold Schwellenwert

1: **function** obtainThreshold(*data*)
2: $dataMin \leftarrow \min(data)$
3: $dataMax \leftarrow \max(data)$
4: $threshold \leftarrow \max(\text{abs}(0.25 \cdot dataMin), \text{abs}(0.25 \cdot dataMax))$
5: **return** *threshold*
6: **end function**

Algorithmus 26 Bestimmung der Elementlängen und der Start- & Endindizes

Input

indicator	Indikator der punktweisen Krümmungsänderung
curveLength	Kurvenlänge der B-Splinekurve

Return

elementLength	Vektor mit Elementlängen
elementIndices	Vektor mit Start- & Endindizes
elementIndicator	Vektor mit Indikator der Krümmungsänderung

```
1: function obtainElementsFromIndicator(indicator, curveLength)
2:     type ← indicator(1)
3:     elStart ← 1
4:     n ← 1
5:     for i ← 2 : indicator.size do
6:         if indicator(i) ≠ type then
7:             elEnd ← i
8:             elementLength(n) ← curveLength(elEnd) − curveLength(elStart)
9:             elementIndices(n) ← [elStart, elEnd]
10:            elementIndicator(n) ← type
11:            n ← n + 1
12:            type ← indicator(i)
13:            elStart ← i
14:        end if
15:    end for
16:    elEnd ← curveLength.size
17:    elementLength(n) ← curveLength(elEnd) − curveLength(elStart)
18:    elementIndices(n) ← [elStart, elEnd]
19:    elementIndicator(n) ← type                    ▷ Letztes Element abschließen
20:    return [elementLength, elementIndices, elementIndicator]
21: end function
```

Die in `indicateDataByAverage` (Alg. 27) als Eingabe erhaltenen Daten werden zunächst als arithmetischer Mittelwert zusammengefasst, um sie anschließend bezüglich des Schwellenwerts zu kategorisieren.

Algorithmus 27 Krümmungsänderung mehrerer Elemente durch Mittelwert abschätzen

Input
data Vektor mit Daten
threshold Schwellenwert der Beurteilung
Return
indicator Prüfergebnis

1: **function** indicateDataByAverage(*data*, *threshold*)
2: $a \leftarrow$ average(*data*)
3: **if** $a < -threshold$ **then**
4: $indicator \leftarrow -1$
5: **else if** $a > threshold$ **then**
6: $indicator \leftarrow 1$
7: **else**
8: $indicator \leftarrow 0$
9: **end if**
10: **return** *indicator*
11: **end function**

Parameter der Gerade werden in Alg. 28 festgelegt.

Algorithmus 28 Eigenschaften einer Geraden festlegen

Input
startPoint Startpunkt des Elements
endPoint Endpunkt des Elements
Return
elementDirection Richtung der Tangente am Elementstart
elementLength Kurvenlänge des Elements

1: **function** obtainStraigth(*startPoint*, *endPoint*)
2: $dx \leftarrow endPoint.\text{x} - startPoint.\text{x}$
3: $dy \leftarrow endPoint.\text{y} - startPoint.\text{y}$
4: $elementDirection \leftarrow \text{sign}(dy) \cdot \text{angleOfVectors}([1; 0], [dx; dy])$
▷ Winkel zur x-Achse mit Vorzeichen von dy
5: $elementLength \leftarrow (endPoint - startPoint).\text{length}$
6: **return** [*elementDirection*, *elementLength*]
7: **end function**

A.4 Weitere Hilfsfunktionen

Algorithmus 29 Berechnung des Winkels zwischen zwei Vektoren

Input

a	Vektor a
b	Vektor b

Return

angle	Winkel in Grad

1: **function** angleOfVectors(a, b)
2: $numerator \leftarrow a.\mathrm{x} \cdot b.\mathrm{x} + a.\mathrm{y} \cdot b.\mathrm{y}$ ▷ Zähler als Skalarprodukt
3: $denominator \leftarrow \mathrm{sqrt}(a.\mathrm{x}^2 + a.\mathrm{y}^2) \cdot \mathrm{sqrt}(b.\mathrm{x}^2 + b.\mathrm{y}^2)$ ▷ Nenner als Produkt der Beträge
4: $angle \leftarrow \arccos(numerator/denominator) \cdot 180/\mathrm{pi}$ ▷ Winkelberechnung mit Umrechnung von Radianten zu Grad
5: **return** *angle*
6: **end function**

Algorithmus 30 Berechnung des Winkels einer Tangente an einem Kreispunkt

Input

$centerPoint$	Kreismittelpunkt
$tangentPoint$	Punkt der Tangente, liegt auf Kreisbogen
$elementIsCCW$	Orientierung des Kreises

Return

$tangentAngle$	Winkel der Tangente bezüglich positiver x-Achse

1: **function** tangentDirection($centerPoint$, $tangentPoint$, $elementIsCCW$)
2: $tV \leftarrow$ tangentVector($centerPoint$, $tangentPoint$, $elementIsCCW$) ▷ Tangentialer Vektor in Richtung der Kilometrierung
3: $tangentAngle \leftarrow$ angleOfVectors([1; 0], tV)
4: **return** $tangentAngle$
5: **end function**

Algorithmus 31 Berechnung des Vektors einer Tangente an einem Kreispunkt

Input

$centerPoint$	Kreismittelpunkt
$tangentPoint$	Punkt der Tangente, liegt auf Kreisbogen
$elementIsCCW$	Orientierung des Kreises

Return

tV	Tangentenvektor in Richtung der Kilometrierung

1: **function** tangentVector($centerPoint$, $tangentPoint$, $elementIsCCW$)
2: $radialVector \leftarrow tangentPoint - centerPoint$
▷ Vektor Kreismittelpunkt zu Tangentenpunkt
3: **if** $elementIsCCW = 1$ **then**
4: $tV \leftarrow [-radialVector.\text{y}; radialVector.\text{x}]$ ▷ Drehen $+90°$
5: **else**
6: $tV \leftarrow [radialVector.\text{y}; -radialVector.\text{x}]$ ▷ Drehen $-90°$
7: **end if**
8: **return** tV
9: **end function**

Algorithmus 32 Berechnung des Vektors einer Richtung durch Winkel

Input

$direction$	Richtung in Grad

Return

$vector$	Richtungsvektor

1: **function** tangentVectorFromDirection($direction$)
2: $vector \leftarrow [\cos(direction \cdot pi/180); \sin(direction \cdot pi/180)]$
3: **return** $vector$
4: **end function**

Literaturverzeichnis

Albert, A. (2016). *Schneider–Bautabellen für Ingenieure: mit Berechnungshinweisen und Beispielen, 22. Aufl.* Bundesanzeiger Verlag.

Amann, J., Flurl, M., Jubierre, J. & Borrmann, A. (2014). An Approach to Describe Arbitrary Transition Curves in an IFC-Based Alignment Product Data Model. In: *Computing in Civil and Building Engineering (2014)*, S. 933–941.

Bronstein, I., Semendjajew, K., Musiol, G. & Mühlig, H. (2016). *Taschenbuch der Mathematik, 10. überarbeitete Auflage.* Europa-Lehrmittel, Haan.

buildingSmart International (2020a). IfcCircularArcSegment2D. https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC1/HTML/schema/ifcgeometryresource/lexical/ifccirculararcsegment2d.htm. Aufgerufen: 06.10.2020.

buildingSmart International (2020b). IfcCurveSegment2D. https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC1/HTML/schema/ifcgeometryresource/lexical/ifccurvesegment2d.htm. Aufgerufen: 06.10.2020.

buildingSmart International (2020c). IfcLineSegment2D. https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC1/HTML/schema/ifcgeometryresource/lexical/ifclinesegment2d.htm. Aufgerufen: 06.10.2020.

buildingSmart International (2020d). IfcTransitionCurveSegment2D. https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC1/HTML/schema/ifcgeometryresource/lexical/ifctransitioncurvesegment2d.htm. Aufgerufen: 06.10.2020.

Coetzee, C. A. (2019). *Sketch-Based Alignment Design.* Bachelorarbeit, Technische Universität München.

Cruz, L. M. V. & Velho, L. (2010, Aug). A Sketch on Sketch-Based Interfaces and Modeling. In: *2010 23RD SIBGRAPI - Conference on Graphics, Patterns and Images Tutorials*, S. 22–33.

DIN EN 13803 (2017). Bahnanwendungen – Oberbau – Trassierungsparameter – Spurweiten 1435 mm und größer. Auslegestelle-Beuth-Bayerisches Hochschul-Konsortium HS München.

Freudenstein, S. (2018). *Verkehrswegebau Grundmodul*. Lehrstuhl und Prüfamt für Verkehrswegebau, Technische Universität München.

Gikas, V. & Stratakos, J. (2012). A Novel Geodetic Engineering Method for Accurate and Automated Road/Railway Centerline Geometry Extraction Based on the Bearing Diagram and Fractal Behavior. *IEEE Transactions on Intelligent Transportation Systems* 13(1), S. 115–126.

Han, L., De Amicis, R. & Conti, G. (2005). Freehand 3D Curve Recognition and Oversketching. In: *Theory and Practice of Computer Graphics*, S. 187–193.

Kobryń, A. (2017). *Transition Curves for Highway Geometric Design*. Springer.

Meek, D. & Walton, D. (1992). Clothoid Spline Transition Spirals. *Mathematics of Computation* 59(199), S. 117–133.

OIP (2020a). TUM Open Infra Platform, Issue 227. https://github.com/tumcms/Open-Infra-Platform/issues/227. Aufgerufen: 10.10.2020.

OIP (2020b). TUM Open Infra Platform, Readme.md. https://github.com/tumcms/Open-Infra-Platform. Aufgerufen: 10.10.2020.

Oliver, M. A. & Webster, R. (2015). *Basic Steps in Geostatistics: The Variogram and Kriging*. Springer International Publishing.

Prautzsch, H., Boehm, W. & Paluszny, M. (2002). *Bézier and B-spline techniques*. Springer Science & Business Media. Stand: 2020-06-09.

Rogers, D. F. (2001). *An introduction to NURBS: with historical perspective*. Morgan Kaufmann Publishers Inc. Stand: 2020-06-09.

Schlenger, J. (2018). *Entwicklung eines Tools für interaktiven Achsenentwurf*. Bachelorarbeit, Technische Universität München.

Vamos, C. & Craciun, M. (2012). *Automatic Trend Estimation*. Springer Netherlands.

Wijnholts, L. (2016). *Automated geometry checking for infrastructure projects*. Masterarbeit, Technische Universiteit Eindhoven.

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

Rottenburg an der Laaber, 11. Oktober 2020

Christoph Kaiser

Christoph Kaiser
[redacted]