Freie Universität Berlin

Fachbereich Mathematik

Approximation von Windkomponenten in der Luftfahrt durch lineare Interpolation

Bachelorarbeit, eingereicht von

Leo Adrian Vornberger

(Matrikel-Nummer 5016126, leo.vornberger@fu-berlin.de)

zur Erlangung des akademischen Grades

Bachelor Of Science

Erstgutachter:	Betreuer:	Zweitgutachter:
Prof. Dr. Ralf Borndörfer	M.Sc. Adam Schienle	Prof. Dr. Thorsten Koch

Berlin, den 17. August 2018

Abstract

Das Wind-Interpolation-Problem (WIP) ist ein bisher selten diskutiertes Problem der Flugplanungsoptimierung, bei dem es darum geht, Wind-Komponenten auf einer Luftstraße zu approximieren. Anhand von Winddaten, die vektoriell an den Gitterpunkten eines den Globus umspannenden Gitters vorliegen, soll bestimmt werden, wie viel Wind entlang der Luftstraße und quer zu ihr weht. Thema dieser Arbeit ist ein Spezialfall des WIP, nämlich das statische WIP auf einer Planfläche (SWIPP). Dazu wird zuerst ein Algorithmus besprochen, der das SWIPP zwar löst, aber einem Ansatz zugrunde liegt, der bei genauerem Hinsehen nicht sinnvoll erscheint: hier wird Wind zwischen vier Punkten interpoliert, wozu es keine triviale Methode gibt. Ähnlich zu diesem Algorithmus, der heute als State-of-the-Art gilt, wird als Ergebnis dieser Arbeit ein neuer Algorithmus vorgestellt, der das SWIPP akkurater und schneller löst. Hier wird deutlich seltener auf die Interpolation zwischen vier Punkten zurückgegriffen – stattdessen wird fast immer linear zwischen zwei Punkten interpoliert. Die Algorithmen zum Lösen des SWIPP werden auf ihre Genauigkeit, asymptotische Laufzeit und Geschwindigkeit untersucht und verglichen. Als Testareal dienen zum einen echte Wetterdaten sowie das Luftstraßennetz, das die Erde umspannt, und zum anderen ein eigens generiertes Windfeld und fiktive Luftstraßen. Es wird gezeigt, dass der hier vorgestellte Algorithmus die State-of-the-Art-Variante in allen genannten Aspekten übertrifft.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich alle Inhalte dieser Arbeit selbstständig erstellt habe. Alle externen Informationen sind als solche gekennzeichnet und mit Quellenangaben hinterlegt.
Diese Arbeit ist speziell zu meiner Erlangung des akademischen Grades *Bachelor of Science* angefertigt worden und war vorher nie, auch nicht zu Teilen, Gegenstand eines anderen Prüfungsverfahrens.

Leo Adrian Vornberger, Berlin den 17. August 2018

Danksagungen

Mein Dank gilt dem Team der Flight Trajectory Optimization am Zuse Institut Berlin, ins.bes. Prof. Dr. Ralf Borndörfer, Adam Schienle, Marco Blanco und Pedro Maristany de las Casas, für ihre durchgehend hilfreiche und konstruktive Unterstützung während der gesamten Zeit.
Außerdem möchte ich dem Team von Lufthansa Systems für ihre Unterstützung und insbesondere Swen Schlobach danken, auf dessen Anregung diese Arbeit zurückzuführen ist.
Ich danke meiner Familie und meinen Freunden, die mich während dieser Arbeit unterstützt und mit ihren Ideen bereichert haben.

Inhaltsverzeichnis

1 Einführung und Präzisierung des Problems

1.1 Einführung

Das Ziel einer jeden Airline ist es, möglichst kostengünstig und umweltschonend zu fliegen. Neben Personalplanung und Wartungsarbeiten hat die Routenplanung einen besonders hohen Einfluss auf die Betriebskosten, da diese kritisch dafür ist, wie viel Treibstoff auf einem Flug verbraucht wird. Mit einem durchschnittlichen Jet-Fuel-Price von 688,1 $ pro Tonne am 20. Juli 2018 [Int18] und einem jährlichen Verbrauch der Deutschen Lufthansa von über 9 600 000 Tonnen im Jahr 2016 [Deu18] ist leicht erkennbar, wie bereits geringe Einsparungen des Treibstoffs zu finanziellen Einsparungen von mehrerenen Millionen Dollar führen können: mit einer Sprit-Einsparung von nur 1,6 % könnte die Deutsche Lufthansa z.B. von diesem finanziellen Überschuss bereits ihre Flotte jährlich um einen Airbus A320 erweitern [Air18]. Aufgrund der ständig wachsenden Passagierzahlen [Int17] und Treibstoffpreisen [Int18] ist zu erwarten, dass die finanziellen Effekte durch spritsparenderes Fliegen in Zukunft noch extremer ausfallen werden.

Es wird nun ein kurzer Überblick darüber gegeben, wie Airlines ihre Routen planen. Über den Globus ist ein Netz aus Waypoints gespannt, deren feste Koordinaten den Airlines bekannt sind. Zwischen diesen Waypoints existieren Verbindungen, die auch Luftstraßen (in dieser Arbeit auch Segmente) genannt werden. Einem Flugzeug ist es nur erlaubt, sich auf diesen Segmenten zu bewegen. Das eben beschriebene Modell lässt sich auf natürlicher Weise als ein gerichteter Graph auffassen, in dem die Waypoints die Knoten und die Segmente die Kanten definieren.

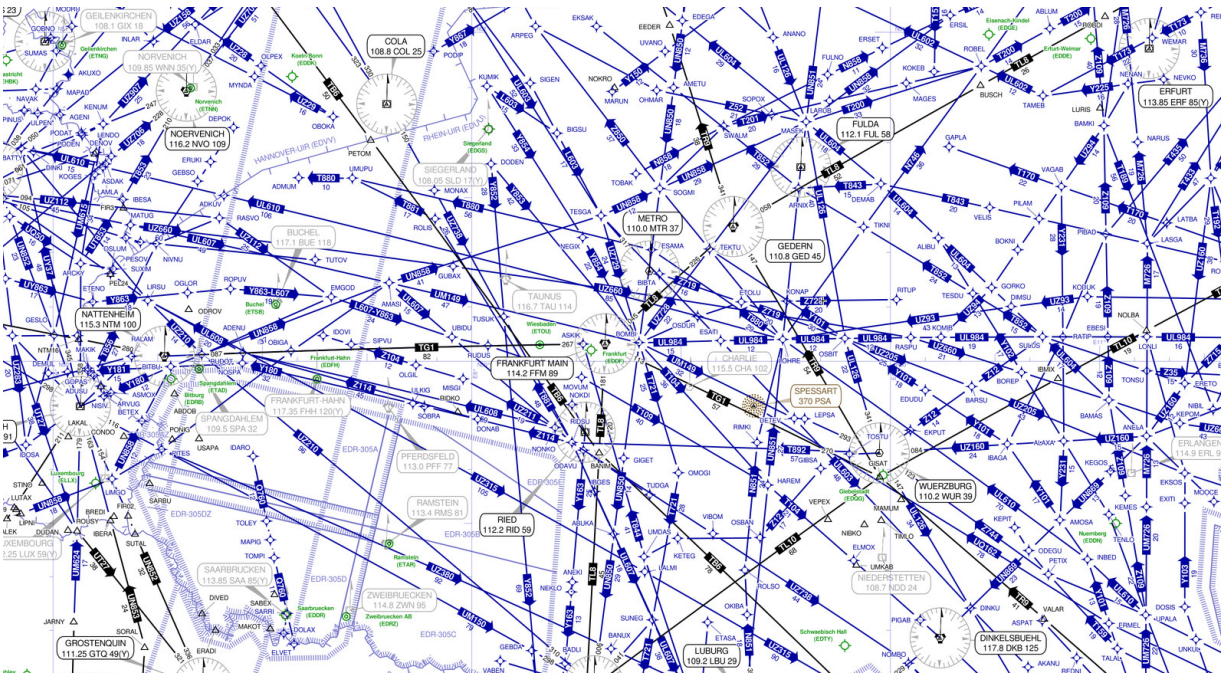

Abbildung 1.1: Navigationskarte des Luftraums über Frankfurt (zu erkennen in der Mitte). Bildquelle: `skyvector.com` (aufgerufen am 19.07.2018)

Eine Route zwischen zwei gegebenen Flughäfen besteht also aus einer Folge von Waypoints und den Höhen, in denen diese überflogen werden. Welche Route für einen Flug am besten geeignet ist, wird kurz vor dem Flug mithilfe eines Flugplanungssystems bestimmt, das eine Version des Dijkstra-Algorithmus (vgl. [Dij59]) verwendet: Für jedes untersuchte Segment wird der Wert einer Kostenfunktion ermittelt, sodass am Ende die Route vom System vorgeschlagen wird, auf der die Summe der Kosten minimal ist. Der Wert dieser Kostenfunktion hängt für jedes Segment unter anderem von dem verbrauchten Treibstoff ab. Flugzeuge fliegen im Reiseflug mit einer näherungsweise konstanten Geschwindigkeit relativ zur umgebenden Luft (auch Air Speed genannt). Der Treibstoffverbrauch hängt also davon ab, wie viel Zeit ein Flugzeug auf einem Segment benötigt. Diese Zeit korelliert mit der Entfernung, die das Flugzeug durch die Luft zurücklegt (Air Distance). Diese Luft hat in der Regel eine eigene Bewegung, die auch Wind genannt wird.

Selbstverständlich wird versucht, im Reiseflug optimalen Wind auszunutzen, da so bei gleichem aerodynamischen Widerstand eine höhere Geschwindigkeit über Grund erzielt werden kann, was in einer kürzeren Estimated Time En-route (ETE) und damit in einem geringeren Spritverbrauch resultiert [HS79]. Da insbesondere für einen Langstreckenflug die Menge der vom Algorithmus untersuchten Segmente sehr große Ausmaße annimmt (die Größe des Suchraums ist auf `skyvector.com` zu erkennen), wird diese Windberechnung vor jedem Flug sehr oft aufgerufen und muss somit schnell funktionieren. Gleichzeitig muss sie akkurate Ergebnisse zurückgeben, da einerseits jeder zusätzliche Liter Kerosin das Gesamtgewicht des Flugzeugs und damit auch seinen Spritverbrauch erhöht [OO89]. Andererseits stellt fehlender Treibstoff ein Sicherheitsrisiko dar, was zu einem Notfall führen kann [Hra12]. In dieser Arbeit wird ein neuer Algorithmus, der diese Windberechnung übernimmt, vorgestellt und mit einem bereits existierenden Algorithmus verglichen.

1.2 Das Wind-Interpolation-Problem

In diesem Abschnitt wird das Wind-Interpolation-Problem (WIP) in seiner allgemeinen Form dargestellt. Ein Punkt $P = (\lambda, \mu, h)$ in der Atmosphäre ist gegeben durch seine geographische Länge λ, seine geographische Breite μ und seine Höhe h über dem Meeresspiegel. Beim WIP ist ein Windvektorfeld w gegeben, welches jedem Punkt in der Atmosphäre zu jeder Zeit t einen Windvektor zuordnet:

$$w : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^2, \quad ((\lambda, \mu, h), t) \mapsto w((\lambda, \mu, h), t) = (w_u, w_v), \tag{1.1}$$

Dabei ist w_u die Windgeschwindigkeit nach Osten und w_v die Geschwindigkeit nach Norden.

Weiterhin seien zwei Punkte P_{en} und P_{ex} gegeben. Die kürzeste Verbindung auf der Erdoberfläche zwischen ihnen definiert ein Segment $S \subset \mathbb{R}^3$. S kann durch eine Kurve $\gamma : [0, 1] \to \mathbb{R}^3$ mit $\gamma(0) = P_{en}$, $\gamma(1) = P_{ex}$ und $\gamma([0, 1]) = S$ beschrieben werden.

Für einen Punkt $P \in S$ und eine Zeit $t \in \mathbb{R}$ wird der Trackwind $w_t(P, t) \in \mathbb{R}^2$ als der Anteil des Vektors $w(P, t)$ in Richtung des Tracks definiert. Der Track $\alpha(P) \in \mathbb{R}$ am Punkt P ist die Flugrichtung über Grund. Dabei steht $0\,^\circ$ für Norden, $90\,^\circ$ für Osten, usw.

$$w_t(P, t) = \langle w(P, t), e_t(P) \rangle \quad \text{mit } e_t(P) := \begin{pmatrix} \sin \alpha(P) \\ \cos \alpha(P) \end{pmatrix} \tag{1.2a}$$

Dabei ist $e_t(P) \in \mathbb{R}^2$ der Einheitsvektor, der am Punkt P in Trackrichtung $\alpha(P)$ zeigt. $\langle ., . \rangle : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ ist das Standard-Skalarprodukt auf dem $\mathbb{R}^2$. Analog wird der Crosswind $w_c(P,t) \in \mathbb{R}^2$ als Anteil des Windvektors $w(P,T)$ in Steuerbord-Richtung (90 ° im Uhrzeigersinn zur Flugrichtung) definiert:

$$w_c(P,t) = \Big\langle w(P,t), e_t^{\perp}(P) \Big\rangle \quad \text{mit } e_t^{\perp}(P) := \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot e_t(P) = \begin{pmatrix} \cos \alpha(P) \\ -\sin \alpha(P) \end{pmatrix} \tag{1.2b}$$

Um den Trackwind $w_t(S)$ auf dem kompletten Segment S zu erhalten, wird nun über γ der Anteil des Windes in Trackrichtung integriert:

$$w_t(S) = \int_{s=0}^{1} \Big\langle w\Big(\gamma(s), t(s)\Big), e_t(\gamma(s)) \Big\rangle \, ds, \tag{1.3a}$$

Dabei gibt $t(s)$ an, zu welcher Zeit das Flugzeug am Punkt $\gamma(s)$ ist. Da γ hier so parametrisiert ist, dass $\gamma([0,1])$ das komplette Segment abdeckt, ist das Ergebnis direkt der durchschnittliche Trackwind auf dem gesamten Segment.

Der Crosswind $w_c(S)$ auf dem Segment S ergibt sich analog zu Formel (1.3a):

$$w_c(S) = \int_{s=0}^{1} \Big\langle w\Big(\gamma(s), t(s)\Big), e_t^{\perp}(\gamma(s)) \Big\rangle \, ds. \tag{1.3b}$$

Abgesehen davon, dass dies rechentechnisch nicht effizient wäre, ist der Wind nicht in der ganzen Atmosphäre bekannt. Gegeben ist ein Gitter mit einem Abstand von 1,25 ° (lateral und longitudinal) und mehreren Höhen, an dessen Gitterpunkten Winddaten existieren. Der Wind ist wie in Formel (1.1) angegeben. Diese Wetterdaten werden alle sechs Stunden veröffentlicht und geben eine Prognose für das Wetter in den nächsten sechs Stunden. Anhand dieser Informationen muss nun zwischen den Gitterpunkten horizontal, vertikal und zeitlich ein Wert interpoliert werden, der den durchschnittlichen Wind in Track- und Crosswind (vgl. Gleichungen (1.3)) approximiert.

1.3 Das statische WIP auf einer Planfläche

Das sehr allgemeine WIP wird in diesem Abschnitt in einen engeren Rahmen eingegrenzt.

Angefangen wird mit der Annahme, dass sich die Segmente nicht auf einer kugelförmigen Oberfläche befinden, sondern auf einer Ebene. Intuitiverweise ist der Unterschied des Verlaufs von langen Segmente drastischer als von kurzen. In Abbildung 1.2 ist dargestellt, wie sich die Erdkrümmung auf ein fiktives Segment über dem Nordatlantik auswirkt. Mit 1000 km Länge ist dieses fiktive Segment allerdings bereits länger als die meisten in dieser Region. Wie in der Abbildung zu sehen ist, unterscheiden sich beide Trajektorien nur minimal voneinander, was die Näherung durch die ungekrümmte Ebene nahe legt.

Als nächstes wird davon ausgegangen, dass sich der Wind zeitlich konstant verhält, wodurch die zeitliche Interpolation entfällt. Dadurch erhält das neue Problem den Zusatz *statisch*.

Eine weitere Annahme ist, dass P_{en} und P_{ex} und mit ihnen das komplette Segment auf der selben Höhe liegen. Außerdem wird angenommen, dass auf dieser Höhe exakte Wetterdaten vorliegen, wodurch die Höheninterpolation komplett entfällt. Diese später zu realisieren ist nicht

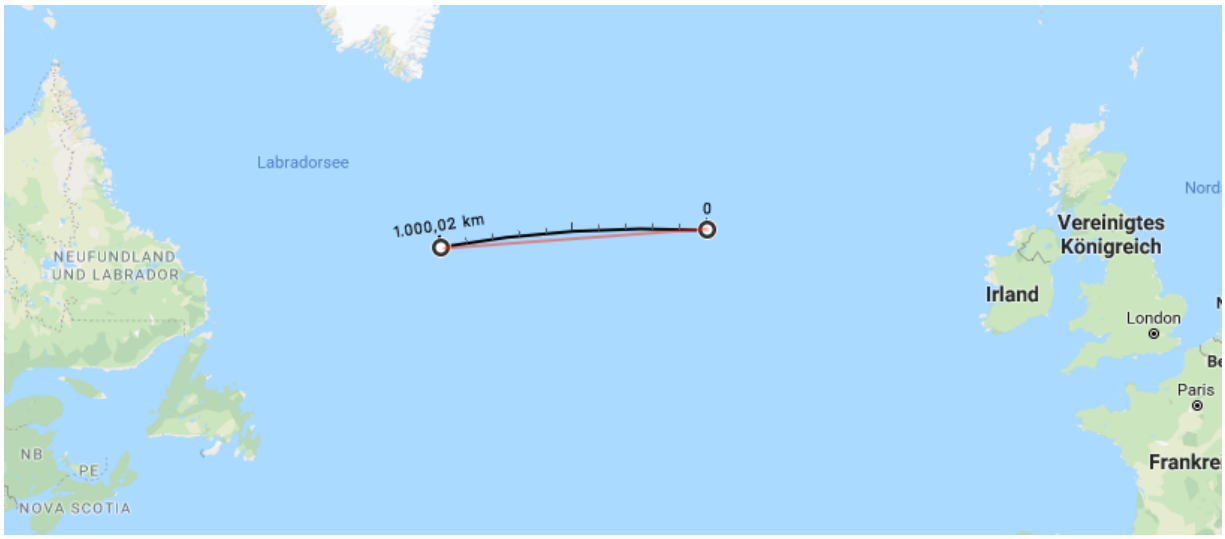

Abbildung 1.2: Vergleich der kürzesten Verbindungen eines fiktiven Segments der Länge von ca. 1000 km. Schwarz: Verlauf des Segments als kürzeste Verbindung von Anfangs- und Endpunkt auf der kugelförmigen Erdoberfläche, rot: Verlauf des Segments als kürzeste Verbindung auf der Ebene. In dieser Abbildung wird die Mercator-Projektion benutzt. Bildquelle: `maps.google.com`, aufgerufen am 19.07.2018

kompliziert: Seien h_L und h_U die Flugflächen unter, bzw. über einem Punkt P auf der Höhe h_P, auf denen die Wetterdaten bekannt sind, und seien P_L und P_U die Punkte auf diesen Höhen unter, bzw. über P, dann beträgt der Wind am Punkt P

$$w(P) = \frac{h_U - h_P}{h_U - h_L} w(P_L) + \frac{h_P - h_L}{h_U - h_L} w(P_U) \tag{1.4}$$

Es folgt nun eine kurze Zusammenfassung des SWIPP.

Problem: Statisches Wind-Interpolation-Problem auf einer Planfläche (SWIPP)
Input: Gitter mit Windinformationen $w : \mathcal{X} \to \mathbb{R}^2$; Segment S, gegeben durch P_{en} und P_{ex}.
Output: Windvektor $(w_t(S), w_c(S))$ mit dem durchschnittlichen Trackwind $w_t(S)$ und Crosswind $w_c(S)$ auf dem Segment S.
Annahmen: Der Wind verhält sich zwischen ausgewählten Punkten linear und ist unabhängig von der Zeit. Das Segment und das Gitter befinden sich auf einer und der selben Ebene.

In dieser Arbeit wird der Wind ausschließlich linear interpoliert. In einigen Situationen ist es sinnvoll, nach der Redewendung „Der Wind dreht“ stattdessen polar zu interpolieren. Die Unterschiede beider Methoden sind in Abbildung 1.3 dargestellt. Polare Interpoliation ist allerdings problematisch, wenn die Windvektoren an den zwei äußeren Punkten in ungefähr entgegengesetzte Richtungen zeigen. Dann ist nämlich nicht eindeutig, in welche Richtung der Wind zwischen diesen Punkten dreht. Ggf. wird mit polarer Interpolation genau die falsche Richtung angenommen. Da der tatsächliche Wind innerhalb der Zellen nicht bekannt ist, sind beide Arten

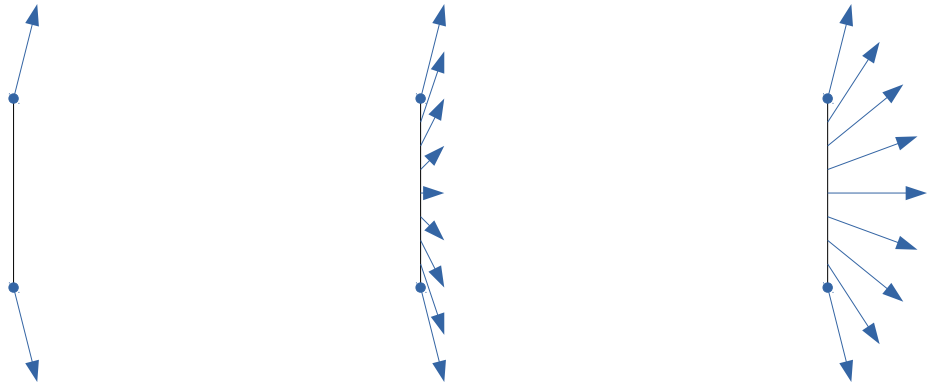

Abbildung 1.3: Lineare vs. polare Interpolation. Links: gegebene Windvektoren an zwei Gitterpunkten und die verbindende Gitterlinie. Mitte: lineare Interpolation auf der Linie (u- und v-Komponente werden linear interpoliert). Rechts: polare Interpolation (Windstärke und Richtung werden linear interpoliert)

der Interpolation à priori gleichzustellen. Aufgrund der eventuellen starken Abweichungen der polaren Interpolation wird in dieser Arbeit also auf die lineare Interpolation zurückgegriffen.

1.4 Notation und Nomenklatur

In diesem Abschnitt werden Notationen und Begriffe eingeführt, die in dieser Arbeit benutzt werden. Alle vorkommenden Größen werden ebenfalls definiert. Es wird vorausgesetzt, dass der Leser dieser Arbeit mit den Grundlagen der linearen Algebra vertraut ist. Als Nachschlagewerk wird [Beu14] empfohlen. Ebenfalls werden Grundkenntnisse in der Analysis verlangt, welche in [Beh15] nachgeschlagen werden können. Weitere vorausgesetzte Grundlage ist das Verständnis der asymptotischen Laufzeit und der $\mathcal{O}$-Notation, welche in [Bar16] erklärt werden.

Für ein gegebenes $n \in \mathbb{N}$ definiere zunächst $[n] := \{0, 1, 2, \ldots, n\}$.

Die geographische Länge wird in dieser Arbeit mit dem Formelzeichen λ angegeben, und es gilt $\lambda \in [-180\,°, 180\,°]$. Dabei stehen positive Werte für Ost, negative Werte für West. Die geographische Breite trägt das Symbol μ, und analog gilt $\mu \in [-90\,°, 90\,°]$. Positive Werte stehen für Nord, negative für Süd.

Für die Berechnung existieren Wetterdaten, die in einem Gitter über die ganze Welt verteilt sind. Das Gitter hat einen Gitterabstand von 1,25 °. Da im Folgenden des öfteren Auf- und Abrundfunktionen benutzt werden müssen, wird dieses Gitter zunächst transformiert, sodass der Gitterabstand eine Längeneinheit beträgt. Das neue Gitter (dargestellt in Abbildung 1.4) hat nun also die Abmessungen $360\,°/1{,}25\,° = 288$ (waagerecht) mal $180\,°/1{,}25\,° = 144$ (senkrecht). Koordinaten werden in dem neuen Gitter als (x, y) angegeben. Der Nullpunkt des Koordinatensystems liegt an dem Äquivalent zu (180 ° West, 90 ° Süd). Das Pendant von (180 ° Ost, 90 ° Nord)

hat die Koordinaten (288,144). Dadurch ergeben sich folgende Umrechnungsgleichungen:

$$\lambda(x) = -180\,^\circ + 1{,}25\,^\circ \cdot x, \quad \mu(y) = -90\,^\circ + 1{,}25\,^\circ \cdot y \quad \text{und} \tag{1.5a}$$

$$x(\lambda) = 144 + \frac{\lambda}{1{,}25\,^\circ}, \quad y(\mu) = 72 + \frac{\mu}{1{,}25\,^\circ}. \tag{1.5b}$$

Diese Transformation ist auch als *Equirectangular Projection* (engl. für rektanguläre Projektion) bekannt [Sny97].

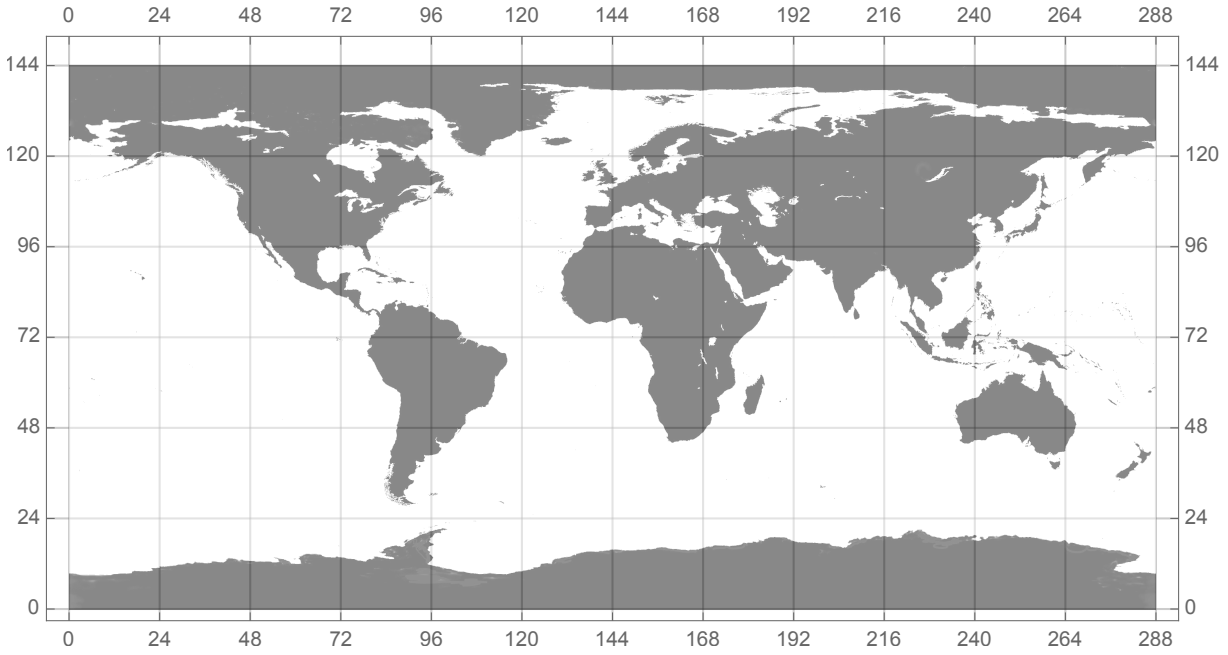

Abbildung 1.4: Transformiertes Gitter mit kartesischen Koordinaten. Für eine bessere Übersichtlichkeit wurden nur alle 24 Höhen- bzw. Breitengrade Linien gezeichnet. Bilddaten: [Wor]

Als eine *Zelle* $C_{i,j}$ dieses Gitters wird die Menge

$$C_{i,j} := \{(x, y) \in \mathbb{R}^2 | i \leq x < i + 1 \wedge j \leq y < j + 1\}, \quad (i, j) \in [288] \times [144] \tag{1.6}$$

definiert.

Die Zellen des neuen Gitters werden jeweils mit den Koordinaten ihrer linken unteren Ecke identifiziert. Ein Punkt mit den Koordinaten (x, y) befindet sich also in der Zelle $(\lfloor x \rfloor, \lfloor y \rfloor)$. Die Menge der Gitterpunkte wird durch $\mathcal{X} = [288] \times [144]$ definiert. Die Gitterpunkte werden mit $X = (i, j) \in \mathcal{X}$ bezeichnet. Oft werden sie in dieser Arbeit mit einem Index versehen, um sie voneinander zu unterscheiden.

Alle Windgrößen werden durch ein w gekennzeichnet (w_t steht für den Trackwind, w_c für den Crosswind; an den Gitterpunkten steht w_u für den Wind nach Osten, w_v für den Wind nach

Norden). Alle Windinformationen, die für die Berechnung zur Verfügung stehen, befinden sich an diesen Gitterpunkten. Es ergibt sich also die Windfunktion

$$w : \mathcal{X} \to \mathbb{R}^2, \quad X \mapsto w(X) = (w_u(X), w_v(X)). \tag{1.7}$$

Segmente werden im Verlauf dieser Arbeit in mehrere *Teilstücke* unterteilt. Die Teilungsstellen werden hier *Schnittstellen* genannt. Diese werden in ihrer Reihenfolge auf dem Segment nummeriert und werden mit $P_i, i \in [n]$ bezeichnet. $n+1$ ist also die Anzahl der Schnittstellen eines Segments. Dabei ist $P_0 = P_{en}$ und $P_n = P_{ex}$. P_1 bis P_{n-1} werden auch als *innere Punkte* bezeichnet, während P_0 und P_n auch *Randpunkte* genannt werden.

$\langle ., . \rangle$ bezeichnet in dieser Arbeit das Standard-Skalarprodukt auf dem $\mathbb{R}^2$:

$$\langle ., . \rangle : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}, \quad (v, w) \mapsto \langle v, w \rangle := v_1 \cdot w_1 + v_2 \cdot w_2 \tag{1.8}$$

Zusätzlich wird die Metrik d verwendet, die über die von $\langle ., . \rangle$ induzierte Standardnorm $||.||$ induziert wird:

$$||.|| : \mathbb{R}^2 \to \mathbb{R}, \quad v \mapsto ||v|| := \sqrt{\langle v, v \rangle} = \sqrt{v_1^2 + v_2^2} \tag{1.9}$$

$$d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2, \quad (v, w) \mapsto d(v, w) := ||v - w|| = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2} \tag{1.10}$$

Zur Bestimmung der Entfernung zweier Punkte auf der Erdkugeloberfläche wird die Funktion

$$d_K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}, (v, w) \mapsto d_K(v, w) \tag{1.11}$$

verwendet. Die Argumente liegen hier in Polarkoordinaten vor. Zurückgegeben wird die Entfernung in Metern.

Längen von Segmenten werden hier mit dem Formelzeichen l gekennzeichnet, sofern sie auf der Ebene gemessen werden. Längen auf der Erdkugeloberfläche werden mit l_K in Metern angegeben.

In dieser Arbeit treten verschiedene Arten von Gewichten auf, die mit σ_u bzw. σ_v, φ, p und τ bezeichnet werden. Sie werden in Abschnitt 2.1 genauer erklärt.

Die Flugrichtung (auch Track genannt) wird mit dem Formelzeichen α versehen. Sie wird als rechtweisender Kurs angegeben, d.h. $0\,^\circ$ steht für Norden, $90\,^\circ$ für Osten, usw.

1.5 Bisherige Ansätze zum Lösen des SWIPP

Auf der Suche nach Literatur wird klar, dass dieses Problem bisher wenig diskutiert wurde – bekannt ist lediglich ein einziger Ansatz. Dieser kommt heutzutage weltweit zum Einsatz und wird deshalb in dieser Arbeit als State Of The Art (SOTA) bezeichnet. Ähnlich zu SOTA wird in Abschnitt 2.2 ein neuer Algorithmus zum SWIPP vorgestellt.

2 Funktionsweise der Algorithmen zum Lösen des SWIPP

Es werden nun zwei Algorithmen vorgestellt, die das SWIPP lösen. Zunächst wird der State-of-the-Art-Algorithmus SOTA im Detail erklärt. Anschließend wird der neue Algorithmus vorgestellt, der ähnlich zu SOTA konzipiert wurde. Da es viele Gemeinsamenkeiten zwischen beiden Algorithmen gibt, werden teilweise nur die Unterschiede erwähnt.

Gegeben sei ein Segment $S \subset \mathbb{R}^2$ auf der Ebene (vollständig definiert durch $P_{en}, P_{ex} \in \mathbb{R}^2$) sowie ein Windgitter $w : \mathcal{X} \to \mathbb{R}^2$. Beide hier besprochenen Algorithmen gliedern sich in zwei Abschnitte. Im Preprocessing wird ermittelt, welche Gitterpunkte $X \in \mathcal{X}$ zu welchen Anteilen das Ergebnis, also den Windvektor $(w_t(S), w_c(S))$ beeinflussen. Ein Gitterpunkt $X \in \mathcal{X}$ beeinflusst eine Schnittstelle P_i ($i \in [n]$) des Segments S, wenn sich der an P_i herrschende Wind aus $w(X)$ bestimmt. Wann dies der Fall ist, ist ein Unterschied zwischen den beiden Algorithmen. Als Ergebnis dieses Preprocessings werden für alle $X \in \mathcal{X}$ die Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ zurückgegeben, die angeben, zu welchem Anteil die u- bzw. die v-Komponente des Windes am Gitterpunkt X das Ergebnis bestimmen. Für diesen Teil ist es unwichtig, wie der Wind weht; die Gewichte hängen nur vom Gitter und vom Segment ab, welche unabhängig vom Wind sind.

Im zweiten Teil, der Query, werden die Winddaten ausgelesen, mit den entsprechenden Gewichten multipliziert und anschließend aufaddiert. Hier unterscheiden sich die beiden Algorithmen nicht in ihrer Arbeitsweise, sondern nur ggf. in der Anzahl der ausgewerteten Gitterpunkte.

2.1 Der State-of-the-Art-Algorithmus SOTA

Bei diesem Algorithmus handelt es sich um eine Lösung des WIP, die bereits sehr schnell und effizient arbeitet. Zunächst wird beschrieben, was im Preprocessing von SOTA passiert. Abhängig von der Länge l_K des Segments S auf der Erdkugel und der geographischen Breite μ_e von P_{en} oder P_{ex} (entscheidend ist jeweils der Punkt von beiden, der näher zum Äquator liegt) wird eine Anzahl von n Teilstücken ermittelt, in die das Segment eingeteilt wird. Diese Anzahl bestimmt sich mit $a = 300\,000$ nach der Formel

$$n = \begin{cases} 1, & l_K < 50 \\ 2, & 50 \leq l_K \leq 30\,000 \\ \min\{2^{\lfloor \frac{l_K}{a} \rfloor + 1 + \lfloor 3{,}5 \cdot \sin |\mu_e| \rfloor}, 32\}, & \text{sonst} \end{cases}. \tag{2.1}$$

Dadurch ergeben sich $n + 1$ Schnittstellen $P_i, i \in [n]$, inkl. $P_0 = P_{en}$ und $P_n = P_{ex}$. Wie im letzten Fall zu erkennen ist, wird diese Anzahl künstlich nach oben durch 33 beschränkt. Diese Heuristik greift jedoch nur bei den wenigsten Segmenten (von den Segmenten des weltweiten Luftstraßen-Netzes sind etwa 6 % betroffen).

O.B.d.A. habe das Segment die Länge 1. Es wird nun angenommen, dass auf dem ganzen Segment ein stückweise konstanter Wind herrscht, wobei jede Schnittstelle P_i den Wind auf einem entsprechenden Bereich repräsentiert. Insbesondere sind also auch Track- und Crosswind stückweise konstant. Der Trackwind auf dem gesamten Segment ist das Integral der entstehenden PWC (engl. **p**iecewise **c**onstant **f**unction, stückweise konstante Funktion). Dieser Ansatz ist für den Trackwind in Abbildung 2.1 dargestellt. Das Verfahren für den Crosswind ist analog hierzu.

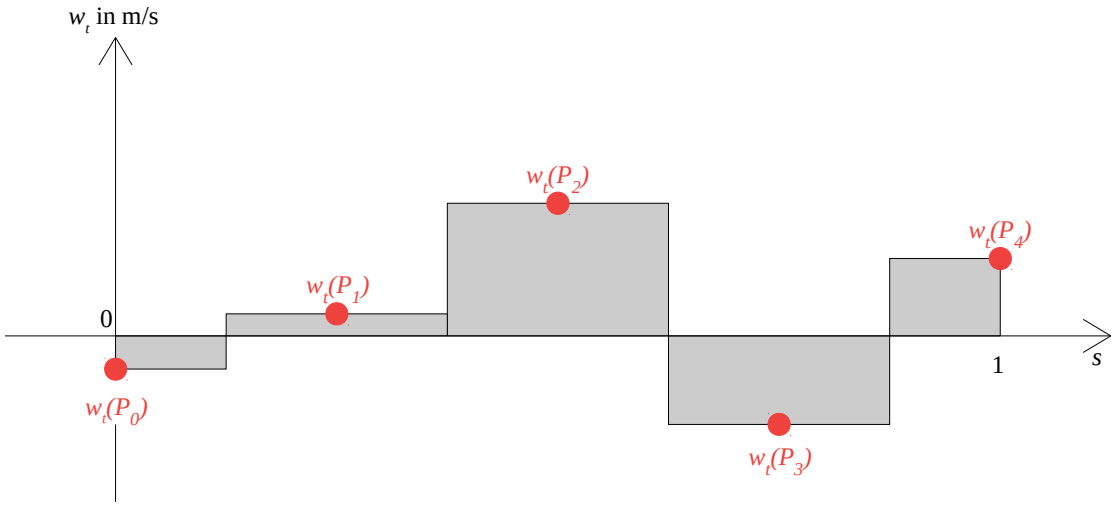

Abbildung 2.1: Beispielhafte resultierende PWC für den Trackwind (hier mit $n = 4$). s gibt die Position auf dem Segment an. Die resultierende Trackwindkomponente ist das Integral der PWC. Das Verfahren für den Crosswind ist analog hierzu.

Um das Integral der PWC in Abbildung 2.1 zu bestimmen, werden jeweils Breite und Höhe der Rechtecke benötigt. Da die Gesamtlänge des Segments auf 1 normiert wurde, geben die Breiten

$$\tau(P_i) = \begin{cases} \frac{1}{2n}, & i \in \{0, n\} \\ \frac{1}{n}, & \text{sonst} \end{cases} \tag{2.2}$$

an, welchen Anteil der an der Schnittstelle P_i herrschende Wind am Gesamtwind ausmacht. Diese Gewichtung gilt sowohl für den Track- als auch für den Crosswind, da die Schnittstellen bei beiden die selben sind.

Der Wind an einer Schnittstelle P_i berechnet sich aus den Daten der vier umliegenden Gitterpunkte A, B, C und D (gegen den Uhrzeigersinn benannt, angefangen beim unteren linken Eckpunkt der Zelle, siehe Abbildung 2.2).

Die Zelle, in der ein Punkt P_i nun liegt, wird mit zwei Schnitten in vier Rechtecke geteilt: ein vertikaler Schnitt bei $x = (P_i)_x$ und ein horizontaler bei $y = (P_i)_y$. Jeder Eckpunkt geht nun mit dem Flächeninhalt des ihm gegenüberliegenden Rechtecks als Gewicht in die Berechnung ein. Konkret bedeutet das für die Gewichte φ_{P_i} der Eckpunkte:

$$\varphi_{P_i}(A) = (C_x - (P_i)_x) \cdot (C_y - (P_i)_y) \tag{2.3a}$$
$$\varphi_{P_i}(B) = ((P_i)_x - D_x) \cdot (D_y - (P_i)_y) \tag{2.3b}$$
$$\varphi_{P_i}(C) = ((P_i)_x - A_x) \cdot ((P_i)_y - A_y) \tag{2.3c}$$
$$\varphi_{P_i}(D) = (B_x - (P_i)_x) \cdot ((P_i)_y - B_y) \tag{2.3d}$$

Diese Gewichte sind in Abbildung 2.2 grafisch dargestellt.

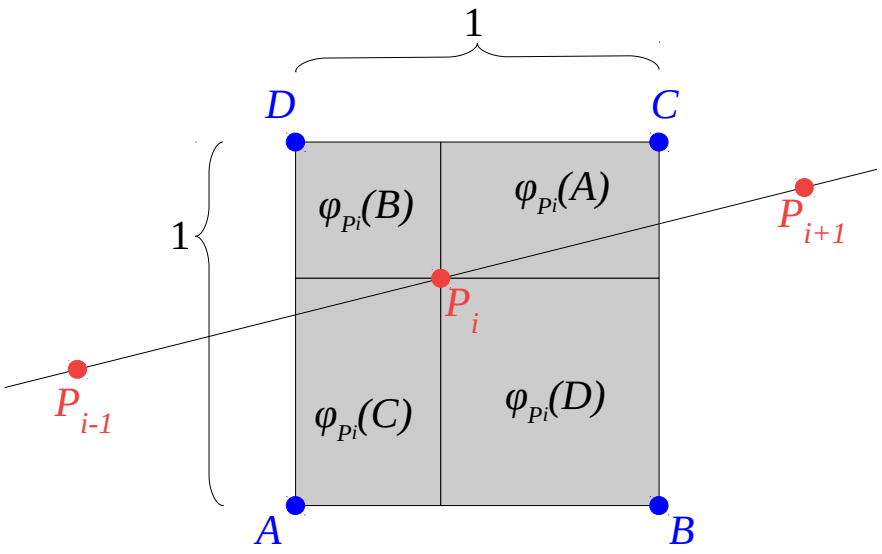

Abbildung 2.2: Grafische Darstellung der Gitterpunktbezeichnungen (blau) und der Gewichtsbestimmung der Eckpunkte in SOTA. Die Flächeninhalte der grauen Rechtecke entsprechen den Gewichten $\varphi_{P_i}(X)$ der Eckpunkte $X \in \{A, B, C, D\}$ bzgl. der Schnittstelle P_i (rot, Mitte). Ebenfalls rot eingezeichnet sind die vorgehende und die nachfolgende Schnittstelle von P_i.

Da alle Seitenlängen der Zelle 1 sind, ist die Summe der Gewichte in Formel (2.3) auch 1. Für alle Gitterpunkte $X \notin \{A, B, C, D\}$ definiere $\varphi_{P_i}(X) = 0$. Es ergibt sich am Punkt P_i ein Wind von

$$\begin{aligned} w(P_i) &= \sum_{X \in \mathcal{X}} \varphi_{P_i}(X) \cdot w(X) \\ &= \varphi_{P_i}(A) \cdot w(A) + \varphi_{P_i}(B) \cdot w(B) + \varphi_{P_i}(C) \cdot w(C) + \varphi_{P_i}(D) \cdot w(D). \end{aligned} \tag{2.4}$$

Man beachte, dass w bisher an P_i nicht definiert ist. Deswegen ist $w(P_i)$ hier eine Fortsetzung dieser Funktion.

Dieser Windvektor wird nun in Track- und Crosswindkomponenten w_t und w_c zerlegt. Diese berechnen sich wie folgt:

$$w_t(P_i) = \langle w(P_i), (e_t)_i \rangle \quad \text{mit} \quad (e_t)_i = \begin{pmatrix} \sin \alpha_i \\ \cos \alpha_i \end{pmatrix}, \tag{2.5a}$$

$$w_c(P_i) = \left\langle w(P_i), (e_t^{\perp})_i \right\rangle \quad \text{mit} \quad (e_t^{\perp})_i = \begin{pmatrix} \cos \alpha_i \\ -\sin \alpha_i \end{pmatrix}, \tag{2.5b}$$

wobei α_i den Track des Segments am Punkt P_i angibt. e_t und $e_t^{\perp}$ sind als Einheitsvektoren zu interpretieren, die in Trackrichtung bzw. nach Steuerbord zeigen.

Jetzt, da an allen Schnittstellen Gewichte und Wind bekannt sind, werden diese miteinander multipliziert und über alle Schnittstellen $P_i, i \in [n]$ summiert:

$$w_t(S) = \sum_{i=0}^{n} \tau(P_i) \cdot w_t(P_i) \tag{2.6a}$$

$$w_c(S) = \sum_{i=0}^{n} \tau(P_i) \cdot w_c(P_i) \tag{2.6b}$$

Diese Darstellung ist zur schnellen Berechnung jedoch ungeeignet, da ein Gitterpunkt mehrere Schnittstellen beeinflussen und somit in mehreren Summanden auftauchen kann. Es ist also notwendig, über alle Gitterpunkte $X \in \mathcal{X}$ zu summieren; gesucht sind Darstellungen der Form

$$w_t(S) = \sum_{X \in \mathcal{X}} \langle \sigma_t(X), w(X) \rangle = \sum_{X \in \mathcal{X}} \Big(\sigma_{t,1}(X) \cdot w_u(X) + \sigma_{t,2}(X) \cdot w_v(X) \Big) \quad \text{und} \tag{2.7a}$$

$$w_c(S) = \sum_{X \in \mathcal{X}} \langle \sigma_c(X), w(X) \rangle = \sum_{X \in \mathcal{X}} \Big(\sigma_{c,1}(X) \cdot w_u(X) + \sigma_{c,2}(X) \cdot w_v(X) \Big), \tag{2.7b}$$

wobei X die Gitterpunkte und $\sigma_{(.)}(X)$ deren Gewichte sind. Diese Darstellung ist insbesondere deswegen wichtig, da $w_u(X)$ und $w_v(X)$ für alle $X \in \mathcal{X}$ bekannt sind. Sind die Koeffizienten $\sigma_{(.)}(X)$ bekannt, sind diese Terme schnell ausgerechnet.

Es wird nun also zurückverfolgt, mit welchem Gewicht ein Gitterpunkt X in die Berechnung einfließt. Durch Ausformulieren von Formel (2.6) und unter Benutzung von Formel (2.4) ergibt sich:

$$w_t(S) = \sum_{i=0}^{n} \tau(P_i) \left\langle \sum_{X \in \mathcal{X}} \varphi_{P_i}(X) \cdot w(X), \begin{pmatrix} \sin \alpha_i \\ \cos \alpha_i \end{pmatrix} \right\rangle$$

Die Bilinearität von $\langle ., . \rangle$ wird nun ausgenutzt, und die Summenzeichen werden vertauscht. Anschließend wird die Symmetrie des Skalarprodukts benutzt:

$$\begin{aligned} w_t(S) &= \sum_{X \in \mathcal{X}} \sum_{i=0}^{n} \left\langle \tau(P_i) \cdot \varphi_{P_i}(X) \cdot w(X), \begin{pmatrix} \sin \alpha_i \\ \cos \alpha_i \end{pmatrix} \right\rangle \\ &= \sum_{X \in \mathcal{X}} \sum_{i=0}^{n} \left\langle \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \begin{pmatrix} \sin \alpha_i \\ \cos \alpha_i \end{pmatrix}, w(X) \right\rangle \end{aligned}$$

Nun wird das Skalarprodukt ausgeschrieben:

$$w_t(S) = \sum_{X \in \mathcal{X}} \left[\left(\sum_{i=0}^{n} \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \sin \alpha_i \right) w_u(X) + \left(\sum_{i=0}^{n} \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \cos \alpha_i \right) w_v(X) \right]$$

Zum Schluss werden für alle $X \in \mathcal{X}$ die Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ eingeführt:

$$\sigma_u(X) = \sum_{i=1}^{n} \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \sin \alpha_i \quad \forall X \in \mathcal{X} \quad \text{und} \tag{2.8a}$$

$$\sigma_v(X) = \sum_{i=1}^{n} \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \cos \alpha_i \quad \forall X \in \mathcal{X} \tag{2.8b}$$

Es ergibt sich:

$$w_l(S) = \sum_{X \in \mathcal{X}} \Big(\sigma_u(X) \cdot w_u(X) + \sigma_v(X) \cdot w_v(X) \Big) \tag{2.9a}$$

Analoge Rechenschritte werden nun für den Crosswind durchgeführt:

$$\begin{aligned} w_c(S) &= \sum_{i=0}^{n} \tau(P_i) \left\langle \sum_{X \in \mathcal{X}} \varphi_{P_i}(X) w(X), \begin{pmatrix} \cos \alpha_i \\ -\sin \alpha_i \end{pmatrix} \right\rangle \\ &= \sum_{X \in \mathcal{X}} \left(\sum_{i=0}^{n} \tau(P_i) \varphi_{P_i}(X) \cos \alpha_i \right) w_u(X) + \left(\sum_{i=0}^{n} \tau(P_i) \varphi_{P_i}(X) \cdot (-\sin \alpha_i) \right) w_v(X) \end{aligned}$$

An dieser Stelle können $\sigma_u(X)$ und $\sigma_v(X)$ aus den Formeln (2.8) eingesetzt werden:

$$w_c(S) = \sum_{X \in \mathcal{X}} \Big(\sigma_v(X) w_u(X) - \sigma_u(X) w_v(X) \Big) \tag{2.9b}$$

In der Darstellung (2.9) wird deutlich, dass, wenn $\sigma_u(X)$ und $\sigma_v(X)$ für alle Gitterpunkte $X \in \mathcal{X}$ bekannt sind, für die Query nur die nötigsten Rechenoperationen übrig bleiben.

Die Funktionsweise von SOTA ist in Algorithmus 1 als Pseudo-Code dargestellt.

Proposition 1. *Die asymptotische Laufzeit von* SOTA *ist für ein Segment der Länge l_K auf der Erdkugeloberfläche durch $\mathcal{O}(2^{\frac{l_K}{300\,000}})$ gegeben.*

Beweis. Gegeben sei ein Segment S mit dem Anfangspunkt $P_{en} = (\lambda_{en}, \mu_{en})$ und dem Endpunkt $P_{ex} = (\lambda_{ex}, \mu_{ex})$. Sei $l_K := d_K(P_{en}, P_{ex})$ die Länge von S auf der Erdoberfläche in Metern und $\mu_e := \min\{|\mu_{en}|, |\mu_{ex}|\}$. Nimm an, dass l_K groß ist, insbesondere $l_K > 30\,000$.

$$\Rightarrow n = 2^{\lfloor \frac{l_K}{300\,000} \rfloor + 1 + \lfloor 3{,}5 \cdot \sin \mu_e \rfloor}$$

Wegen $\lfloor 3{,}5 \cdot \sin \mu_e \rfloor \leq 3$ gilt nun

$$\begin{aligned} n &\leq 2^{\lfloor \frac{l_K}{300\,000} \rfloor} \cdot 2^4 \\ \Rightarrow n(l_K) &\in \mathcal{O}(2^{\frac{l_K}{300\,000}}) \end{aligned} \tag{2.10}$$

Wie in Algorithmus 1 zu erkennen ist, wird die einzige Schleife, die vom Segment abhängt, $n + 1$ mal durchlaufen. Die Anzahl der Rechenoperationen innerhalb eines Schleifendurchlaufs ist konstant und hängt nicht von n ab. Daher ist die asymptotische Laufzeit von SOTA durch $\mathcal{O}(2^{\frac{l_K}{300\,000}})$ gegeben. □

An dieser Stelle sei angemerkt, dass die Koordinaten von P_{en} und P_{ex} ggf. in kartesischer Form als Input zur Verfügung stehen, und nicht polar. Dies beeinflusst allerdings nicht die asymptotische Laufzeit, da die Koordinaten mit den Formeln in (1.5) umgerechnet werden können.

Input: Gitter mit Winddaten $w : \mathcal{X} \to \mathbb{R}^2$, Segment gegeben durch P_{en}, P_{ex}.
Output: Windvektor $\begin{pmatrix} w_t(S) \\ w_c(S) \end{pmatrix}$

begin
  for $X \in \mathcal{X}$ **do**
    $\sigma_u(X) \leftarrow 0$, $\sigma_v(X) \leftarrow 0$
  $\mu_e \leftarrow \min\{|\mu((P_{en})_y)|, |\mu((P_{ex})_y)|\}$
  $l_K \leftarrow d_K(P_{en}, P_{ex})$ `/* l ist hier die Länge des Segments auf der Erdkugeloberfläche, angegeben in Metern. */`
  $$n \leftarrow \begin{cases} 1, & l_K < 50 \\ 2, & 50 \le l_K \le 30\,000 \\ \min\{2^{\lfloor \frac{l_K}{300\,000} \rfloor + 1 + \lfloor 3{,}5 \cdot \sin \mu_e \rfloor}, 32\}, & \text{else} \end{cases}$$
  for $i \in [n]$ **do**
    $P \leftarrow \frac{i}{n} P_{ex} + \frac{n-i}{n} P_{ex}$
    $$\tau \leftarrow \begin{cases} \frac{1}{2n}, & i \in \{0, n\} \\ \frac{1}{n}, & \text{else} \end{cases}$$
    `SOTARaiseWeights(`P, τ`)`
  return $\sum_{X \in \mathcal{X}} \begin{pmatrix} \sigma_u(X) \cdot w_u(X) + \sigma_v(X) \cdot w_v(X) \\ \sigma_v(X) \cdot w_u(X) - \sigma_u(X) \cdot w_v(X) \end{pmatrix}$
Function `SOTARaiseWeights(`P, τ`)`
  $x \leftarrow \lfloor P_x \rfloor$, $y \leftarrow \lfloor P_y \rfloor$
  $\alpha \leftarrow$ Track am Punkt P
  $X_1 \leftarrow (x, y)$
  $X_2 \leftarrow (x + 1, y)$
  $X_3 \leftarrow (x + 1, y + 1)$
  $X_4 \leftarrow (x, y + 1)$
  for $j \in \{1, 2, 3, 4\}$ **do**
    $\sigma_u(X_j)$ += $\Big(1 - |(X_j)_x - P_x|\Big) \cdot \Big(1 - |(X_j)_y - P_y|\Big) \cdot \tau \cdot \sin \alpha$
    $\sigma_v(X_j)$ += $\Big(1 - |(X_j)_x - P_x|\Big) \cdot \Big(1 - |(X_j)_y - P_y|\Big) \cdot \tau \cdot \cos \alpha$

Algorithmus 1: Pseudocode von SOTA

2.2 SWEN, ein neuer Ansatz zum Lösen des SWIPP

Gegeben sei nach wie vor das Segment S und das Windgitter $w : \mathcal{X} \to \mathbb{R}^2$. Auch wenn SOTA prozesssicher funktioniert, beruht er auf einem nicht ganz einleuchtenden Ansatz. Wie viele Schnittstellen für die Berechnung herangezogen werden, entscheidet sich nach einer obskuren Formel (vgl. Formel 2.1). Da die Schnittstellen äquidistant auf dem Segment verteilt sind, liegen sie fast immer im Inneren der Zellen. Dort ist der Wind jedoch nicht trivial zwischen den vier umliegenden Gitterpunkten zu interpolieren. Nach den Gitterpunkten, an denen der echte Wind bekannt ist, kann die Windfunktion am einfachsten auf den Gitterkanten fortgesetzt werden: um hier den Wind zu erhalten muss lediglich zwischen zwei Gitterpunkten linear interpoliert werden. Es ist also sinnvoll, die Schnittstellen nicht willkürlich in den Zellen zu setzen, sondern dort, wo das Segment S das Gitter schneidet. Mit diesem systematischen Ansatz soll der Algorithmus SWEN (**S**ystematic **W**ind **E**stimation on **N**etworks) das SWIPP nicht nur akkurater lösen – durch entfallende Rechenoperationen soll SWEN zudem schneller arbeiten als SOTA.

Durch den Ansatz von SWEN fallen für jede Schnittstelle P_i nur zwei beeinflussende Gitterpunkte (sie heißen hier A und B) an, und damit nur zwei Gewichte $\varphi_{P_i}(A)$ und $\varphi_{P_i}(B)$. In Abbildung 2.3 ist dargestellt, wie diese Gewichte entstehen.

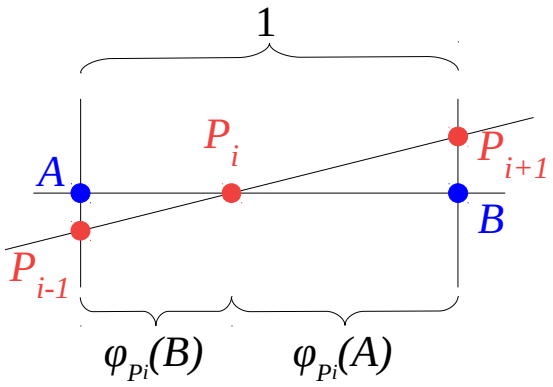

Abbildung 2.3: Grafische Darstellung der Gitterpunktbezeichnungen und der Gewichtsbestimmung dieser Gitterpunkte in SWEN. Abgebildet ist eine Gitterkante und ihre zwei begrenzenden Gitterpunkte A und B (blau). Das Segment schneidet die Gitterkante im Punkt P_i (rot). Ebenfalls in rot eingezeichnet sind die vorgehende und die nachfolgende Schnittstelle von P_i.

An einem solchen Schnittpunkt P_i berechnet sich der Wind $w(P_i)$ als Linearkombination der Winde der begrenzenden Punkte A und B der Gitterkante. p sei definiert als die Länge der Strecke $\overline{AP_i}$. Mit

$$\varphi_{P_i}(X) = \begin{cases} 1-p, & X = A \\ p, & X = B \\ 0, & \text{sonst} \end{cases} \tag{2.11}$$

gilt nun

$$\begin{aligned} w(P_i) &= \sum_{X \in \mathcal{X}} \varphi_{P_i}(X) \cdot w(X) \\ &= (1-p) \cdot w(A) + p \cdot w(B) \end{aligned} \tag{2.12}$$

Hierbei ist $w(A)$ der Wind am Punkt A, bzw. $w(B)$ der Wind am Punkt B. Da die Strecke $\overline{AB}$ die Länge 1 hat, bedarf es hier keinem Korrekturfaktor bzgl. der Gewichte.

Diese Berechnungen geschehen für alle inneren Punkte. Die Randpunkte, P_{en} und P_{ex} liegen jedoch nicht zwangsweise auf einer Gitterlinie, weswegen sie nach einem anderen Schema berechnet werden müssen. Hier wird wieder auf die Methode von SOTA zurückgegriffen, in welcher der Wind bilinear interpoliert wird (siehe Formel (2.4)). Die Aufteilung in Track- und Crosswind erfolgt analog zu SOTA, Formel siehe (2.5).

Zwischen den Schnittstellen liegt jeweils das Innere einer Zelle. Da hier keine Information über den Wind existiert, wird angenommen, dass sich dieser zwischen den Schnittstellen linear verhält. Insbesondere verhalten sich also auch Track- und Crosswind linear, sodass für beide entlang des Segments eine stückweise lineare Funktion (engl. engl: **p**iece**w**ise **l**inear function, kurz PWL) entsteht, die in Abbildung 2.4 für den Trackwind dargestellt ist.

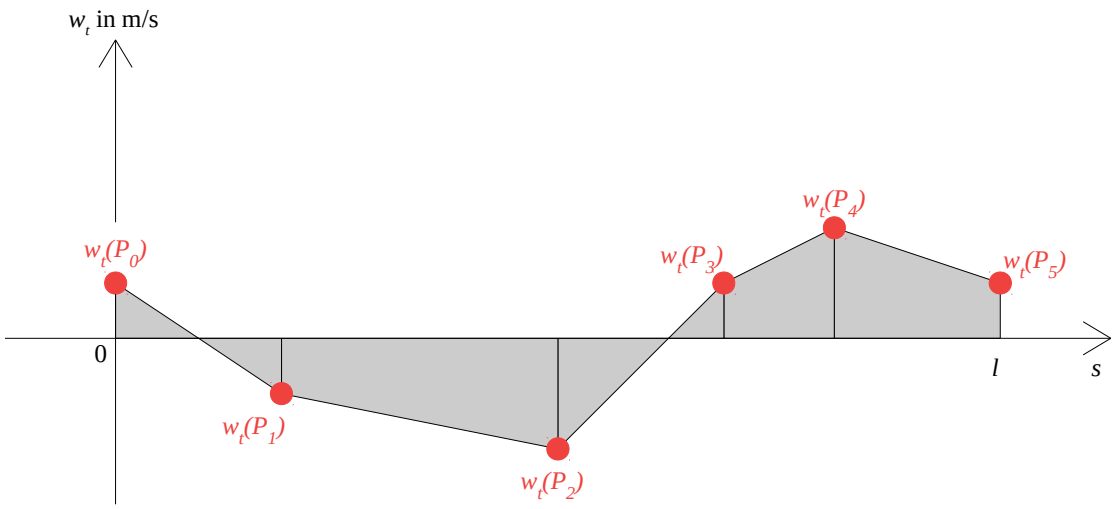

Abbildung 2.4: Beispielhafte resultierende Funktion für den Trackwind in SWEN. s steht für die Position auf dem Segment, läuft diesmal allerdings von 0 bis $l := d(P_{en}, P_{ex})$. Die resultierende Trackwindkomponente ist das Integral der PWL. Das Verfahren für den Crosswind ist analog hierzu.

Das Integral dieser PWL kann als Summe von Trapezen bestimmt werden, was allerdings einige Probleme in der Implementation mit sich bringt. Das Ziel ist wie bei SOTA, Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ für die Winde $w_u(X)$ und $w_v(X)$ an den Gitterpunkten $X \in \mathcal{X}$ zu bestimmen, mit

welchen am Ende das Ergebnis schnell berechnet werden kann. Hier, wie auch im Rest dieses Absatzes, werden nur innere Punkte diskutiert, da diese erstens den Hauptunterschied zu SOTA darstellen und zweitens in einer deutlichen größeren Anzahl existieren. Durch das Prinzip von SWEN ist es möglich, dass ein Gitterpunkt X zwei Schnittstellen beeinflusst, was bedeutet, dass $w(X)$ zur Berechnung von drei Trapezen nötig ist. Umgekehrt wird ein Trapez von bis zu vier Gitterpunkten beeinflusst. Es ist daher sinnvoll, zur Vereinfachung der Implementation die PWL in eine PWC zu transformieren, sodass die Flächenbilanz unter dem Graphen erhalten bleibt, wie in Abbildung 2.5 dargestellt ist. Das Integral ist nun eine Summe von Flächeninhalten von Rechtecken, die jeweils von nur zwei Gitterpunkten beeinflusst werden. So kann der Algorithmus mit einer vergleichsweise einfachen Schleife über alle Schnittstellen implementiert werden. Anderenfalls müsste dies mit einer Schleife über alle Trapeze passieren, sodass zu jedem Gitterpunkt mehr Rechnungen stattfinden würden, was die Geschwindigkeit reduzieren würde.

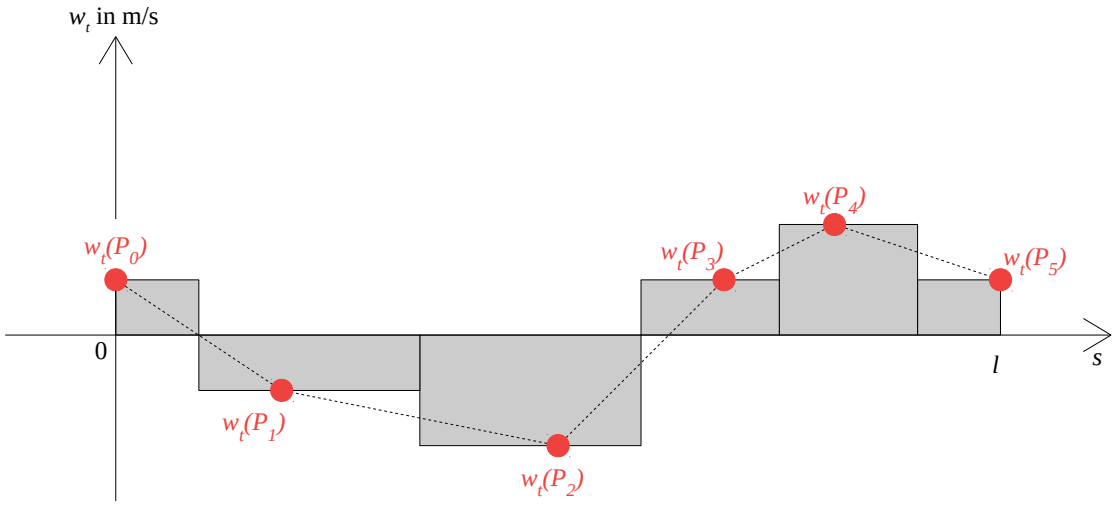

Abbildung 2.5: Gegeneinanderstellung der Rechnung als stückweise lineare Funktion (PWL), bzw. stückweise konstante Funktion (PWC). Die Integrale des gestrichelten Graphen und der PWC sind gleich.

Die Höhe der Rechtecke in Abbildung 2.5 ist gegeben durch den jeweiligen Trackwind $w_t(P_i)$ an jener Schnittstelle P_i, die dem Rechteck zugeordnet ist. Die Breite der Rechtecke ergibt sich durch den Durchschnitt der Entfernungen von P_i zu seinem Vorgänger und von P_i zu seinem Nachfolger. Anders als in SOTA (vgl. Gleichung (2.2)) wird hier also definiert:

$$\tau(P_i) = \begin{cases} \frac{1}{2} d(P_0, P_1), & i = 0 \\ \frac{1}{2} d(P_n, P_{n-1}), & i = n \\ \frac{1}{2} \Big(d(P_i, P_{i-1}) + d(P_i, P_{i+1}) \Big), & \text{sonst} \end{cases} \tag{2.13}$$

An dieser Stelle ist anzumerken, dass sich die Gewichte hier anders als in SOTA nicht zu 1, sondern

zur Gesamtdistanz des Segments aufaddieren. Da dieser Faktor linear in das Endergebnis eingeht, muss dieses als Konsequenz durch die Länge l des Segments geteilt werden.

In der Query werden nun mit

$$\sigma_u(X) = \sum_{i=1}^{n} \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \sin \alpha_i \quad \forall X \in \mathcal{X} \quad \text{und} \tag{2.14a}$$

$$\sigma_v(X) = \sum_{i=1}^{n} \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \cos \alpha_i \quad \forall X \in \mathcal{X} \tag{2.14b}$$

ähnlich zu Formel (2.9) Track- und Crosswind ausgerechnet:

$$w_t(S) = \frac{1}{l} \sum_{X \in \mathcal{X}} \Big(\sigma_u(X) \cdot w_u(X) + \sigma_v(X) \cdot w_v(X) \Big) \tag{2.15a}$$

$$w_c(S) = \frac{1}{l} \sum_{X \in \mathcal{X}} \Big(\sigma_v(X) \cdot w_u(X) - \sigma_u(X) \cdot w_v(X) \Big) \tag{2.15b}$$

Die Funktionsweise von SWEN ist in Algorithmus 2 als Pseudo-Code dargestellt.

Proposition 2. *Die asymptotische Laufzeit von* SWEN *ist für ein Segment der Länge l durch $\mathcal{O}(l)$ gegeben.*

Beweis. Gegeben sei ein Segment S mit dem Anfangspunkt $P_{en} = (x_{en}, y_{en})$ und dem Endpunkt $P_{ex} = (x_{ex}, y_{ex})$. O.B.d.A. gelte $x_{en} \leq x_{ex}$, $y_{en} \leq y_{ex}$ und $P_{en} \in C_{i,j}$. Sei $l := d(P_{en}, P_{ex})$ die Länge von S. Sei P die Menge der Schnittstellen in SWEN. Da für jeden Schnittpunkt von Segment und Gitter eine Schnittstelle anfällt und P_{en} und P_{ex} als Schnittstellen benutzt werden, werden für die Berechnung insgesamt

$$\begin{aligned} |P| &= (\lceil (P_{ex})_x \rceil - i - 1) + (\lceil (P_{ex})_y \rceil - j - 1) + 2 \\ &= (\lceil (P_{ex})_x \rceil - i) + (\lceil (P_{ex})_y \rceil - j) \end{aligned}$$

Schnittstellen herangezogen. Ist l groß, treffen die Näherungen

$$\begin{aligned} \lceil (P_{ex})_x \rceil - i \approx (P_{en})_x - (P_{ex})_x = l \cdot \sin \alpha \quad \text{und} \\ \lceil (P_{ex})_y \rceil - j \approx (P_{en})_y - (P_{ex})_y = l \cdot \cos \alpha \end{aligned}$$

zu.

$$\begin{aligned} \Rightarrow |P| &\approx l \cdot (\sin \alpha + \cos \alpha) \\ &= \sqrt{2} \cdot l \cdot \sin (\alpha + 45\,^\circ) \\ \Rightarrow |P|(l) &\in \mathcal{O}(l) \end{aligned} \tag{2.16}$$

Ähnlich zu SOTA wird auch hier die einzige Schleife, die vom Segment abhängig ist, $|P|$ mal durchlaufen, wie in Algorithmus 2 zu erkennen ist. Die Anzahl der Rechenoperationen innerhalb eines Schleifendurchlaufs ist nach oben durch eine Konstante beschränkt, die nicht von $|P|$ abhängt. Daher ist die asymptotische Laufzeit von SWEN durch $\mathcal{O}(l)$ gegeben. □

Auch an dieser Stelle sei angemerkt, dass P_{en} und P_{ex} ggf. in Polarkoordinaten als Input zur Verfügung stehen und nicht in kartesischen Koordinaten. Dies beeinflusst allerdings nicht die asymptotische Laufzeit, da die Koordinaten mit den Formeln in (1.5) umgerechnet werden können.

Input: Gitter mit Winddaten $w : \mathcal{X} \to \mathbb{R}^2$, Segment gegeben durch P_{en}, P_{ex}.

Output: Windvektor $\begin{pmatrix} w_t(S) \\ w_c(S) \end{pmatrix}$

begin
  for $X \in \mathcal{X}$ **do**
    $\sigma_u(X) \leftarrow 0$, $\sigma_v(X) \leftarrow 0$
  $S \leftarrow \overline{P_{en}P_{ex}}$
  $P \leftarrow \{(x, y) \in S : \lfloor x \rfloor = x \vee \lfloor y \rfloor = y\} \cup \{P_{en}, P_{ex}\}$ `/*` $P_0 \leftarrow P_{en}$, $P_n \leftarrow P_{ex}$, `dazwischen aufsteigende Nummerierung, siehe Abschnitt 1.4 */`
  $n \leftarrow |P| - 1$
  for $i \in [n]$ **do**
    $x \leftarrow \lfloor (P_i)_x \rfloor$, $y \leftarrow \lfloor (P_i)_y \rfloor$
    if $i \in \{0, n\}$ **then**
      $\tau \leftarrow \begin{cases} \frac{1}{2} d(P_0, P_1), & i = 0 \\ \frac{1}{2} d(P_{n-1}, P_n), & i = n \end{cases}$
      `SOTARaiseWeights(`P_i, τ`)`
    else
      $\tau \leftarrow \frac{1}{2} d(P_{i-1}, P_{i+1})$
      $\alpha \leftarrow$ Track am Punkt P
      $X_1 \leftarrow (x, y)$
      if $(P_i)_x = x$ **then**
        $p \leftarrow (P_i)_y - y$
        $X_2 \leftarrow (x, y + 1)$
      else
        $p \leftarrow (P_i)_x - x$
        $X_2 \leftarrow (x + 1, y)$
      $\sigma_u(X_1) += (1 - p) \cdot \tau \cdot \sin \alpha$
      $\sigma_v(X_1) += (1 - p) \cdot \tau \cdot \cos \alpha$
      $\sigma_u(X_2) += p \cdot \tau \cdot \sin \alpha$
      $\sigma_v(X_2) += p \cdot \tau \cdot \cos \alpha$
  return $\frac{1}{d(P_0, P_n)} \cdot \sum_{X \in \mathcal{X}} \begin{pmatrix} \sigma_u(X) \cdot w_u(X) + \sigma_v(X) \cdot w_v(X) \\ \sigma_v(X) \cdot w_u(X) - \sigma_u(X) \cdot w_v(X) \end{pmatrix}$

Algorithmus 2: Pseudocode von SWEN

3 Vergleich: SWEN gegen SOTA

Es folgt nun ein Blick auf die Unterschiede in der Leistungsfähigkeit der beiden Algorithmen zum SWIPP. Als erstes wird analysiert, welche Unterschiede in der Arbeitsweise der Algorithmen auftreten. Anschließend wird überprüft, ob SWEN tatsächlich schneller und akkurater arbeitet, wofür er konzipiert wurde. Beide Algorithmen wurden hierfür in der Software *Mathematica* implementiert. Die Daten, mit denen alle folgenden Tests durchgeführt wurden, sind in Tabelle 3.1 aufgeführt.

Tabelle 3.1: Technische Daten des Computers, auf dem die Algorithmen zum SWIPP implementiert und die Tests durchgeführt wurden

Computer	Mac Mini (Late 2014)
Prozessor	2,6 GHz Intel Core i5
Arbeitsspeicher	8 GB 1600 MHz DDR3
Betriebssystem	macOS High Sierra (10.13.6)
Software	Wolfram *Mathematica* 10.1

Als Arbeitsareal für die beiden Algorithmen dient in einigen Unterbschnitten eine Liste aller bekannten Segmente weltweit, die von Lufthansa Systems zur Verfügung gestellt wurde. Die Segmente sind mit ihren Anfangs- und Endpunkten in Polarkoordinaten angegeben und werden zunächst in die hier verwendeten Koordinaten konvertiert. Ignoriert werden all jene Segmente, die auf der Ebene eine horizontale Ausdehnung von 20 Einheiten überschreiten, da diese nicht ansatzweise realitätsgetreu ausgewertet würden. Außerdem werden werden Segmente ignoriert, die eine Länge von null haben oder Dopplungen von anderen Segmenten sind. Je zwei Segmente, die vertauschte Anfangs- und Endpunkte haben, werden *nicht* ignoriert sondern als verschiedenen betrachtet. Übrig bleiben 370 857 Segmente. Dieses Testset an Segmenten trägt den Namen $\mathcal{T}$.

Um einen Vergleich zur tatsächlichen Situation auf der Erdkugel darzustellen, wird für die Bestimmung der Anzahl der Schnittstellen in SOTA (und ausschließlich dafür) die Länge l_K eines Segments auf der Erdkugeloberfläche benutzt. Anderenfalls würde diese Anzahl verfälscht werden.

3.1 Implementierung der Algorithmen zum SWIPP

Als Vorbereitung dient ein grober Überblick über die Besonderheiten bei der Implementierung der beiden Algorithmen zum SWIPP. Es werden keine Details erläutert, die für die weitere Vorgehensweise nicht von Bedeutung sind.

Implementierung von SOTA

Nachdem die Anzahl der Schnittstellen ermittelt wurde, werden diese rekursiv bestimmt: In der Mitte des Segments wird eine Schnittstelle gesetzt, und mit dem selben Prinzip werden die beiden entstandenen Hälften wieder geteilt. Bereits in diesem Schritt erhalten die Punkte ihr Gewicht

$\tau(P_i)$. Erst bei einer bestimmten Rekursionstiefe, die durch die Anzahl der Schnittstellen gegeben ist, wird abgebrochen.

In einer Schleife über alle Schnittstellen P_i werden nun für jede dieser α_i bestimmt und die beeinflussenden Gitterpunkte X in eine Liste aller relevanten Gitterpunkte eingetragen, sofern sie noch nicht in ihr enthalten sind. Auch die Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ werden in dem Fall in einer Liste gespeichert und mit 0 initialisiert. Anschließend werden die Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ um $\tau(P_i) \cdot \varphi_{P_i}(X) \cdot \sin\alpha$ bzw. $\tau(P_i) \cdot \varphi_{P_i}(X) \cdot \cos\alpha$ erhöht.

Die Query erfolgt analog zu Algorithmus 1 (Zeile 11).

Implementierung von SWEN

Auch SWEN bestimmt die Schnittstellen mit einer rekursiven Methode – allerdings wird hier in der tatsächlichen Reihenfolge der Punkte auf dem Segment vorgegangen, angefangen wird bei $P_0 = P_{en}$, der direkt in die Liste L_P eingetragen wird. Außerdem werden die P_0 beeinflussenden Gitterpunkte X zur Liste L_X der relevanten Gitterpunkte hinzugefügt, und alle ihre Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ werden mit 0 initialisiert und den Listen L_{σ_u} und L_{σ_v} gespeichert. Eine Liste für die Entfernungen wird mit $L_d = \{\}$ initialisiert. Als nächstes wird analysiert, in welche Richtung das Segment die erste Zelle verlässt. Dafür gibt es acht Möglichkeiten: durch die obere, rechte, untere oder linke Gitterkante, oder durch eine der vier Ecken. Für jeden Fall gibt es eine für die jeweilige Richtung optimierte Methode, die nun stellvertretend für eine Zelle aufgerufen wird, wenn das Segment in diese eintritt. Im Folgenden wird auch davon gesprochen, dass „eine Zelle aufgerufen wird“.

In diesem Rekursionsschritt werden zunächst der Schnittpunkt P_i des Segments mit dem Gitter beim Eintreffen in die neue Zelle und die Entfernung von P_i zur letzten Schnittstelle in die Listen L_P und L_d eingetragen. Als nächstes werden die P_i beeinflussenden Gitterpunkte X der Liste L_X hinzugefügt, sofern sie noch nicht darin enthalten sind. In diesem Fall werden auch die Gewichte $\sigma_u(X) = 0$ und $\sigma_v(X) = 0$ der Liste L_{σ_u} bzw. L_{σ_v} hinzugefügt. Je nachdem, wo das Segment die aktuelle Zelle wieder verlässt, wird die nächste Zelle aufgerufen und der Rekursionsschritt beginnt dort von vorne.

Da nun die Entfernungen $d(P_{i-1}, P_i)$ und $d(P_i, P_{i+1})$ in der Liste P_d eingetragen sind, kann $\tau(P_i) = d(P_{i-1}, P_i) + d(P_i, P_{i+1})$ bestimmt werden. Nun werden auch für alle P_i beeinflussenden Gitterpunkte X die Gewichte $\sigma_u(X) = \tau(P_i) \cdot \varphi_{P_i}(X) \cdot \sin\alpha$ bzw. $\sigma_v(X)\tau(P_i) \cdot \varphi_{P_i}(X) \cdot \cos\alpha$ berechnet und zu den entsprechenden Listeneinträgen in L_{σ_u} bzw. L_{σ_v} hinzuaddiert. Hier endet der Rekursionsschritt.

Die Rekursion endet, wenn erkannt wird, dass sich P_{ex} in der aufgerufenen Zelle befindet. Dann werden zunächst die ersten Operationen des Rekursionsschritts ausgeführt. Anschließend werden die verbleibenden Eckpunkte X der aktuellen Zelle zu L_X und die entsprechenden Gewichte $\sigma_u(X) = 0$ und $\sigma_v(X) = 0$ zu L_{σ_u} bzw. L_{σ_v} hinzugefügt, die P_{ex} beeinflussen. Der Rekursionsschritt wird nun nicht mehr aufgerufen. Die Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ werden für alle P_{ex} beeinflussenden Gitterpunkte X ausgerechnet und in den entsprechenden Listeneinträgen eingesetzt.

Nun werden in allen Rekursionsschritten die zweiten Hälften ausgeführt, bis auch die Gewichte $\sigma_u(X)$ und $\sigma_v(X)$ für die P_{en} beeinflussenden Gitterpunkte X eingetragen sind.

Durch geschickte Überlegungen darüber, zu welchem Zeitpunkt welche Gitterpunkte der Liste

L_X hinzugefügt werden, kann das Verfahren um einige Abfragen erleichtert und damit beschleunigt werden. Diese Anpassungen wurden hier auch vorgenommen, doch da sie für die folgenden Unterabschnitte irrelevant ist, wird nicht weiter auf sie eingegangen.

Es gibt außerdem zahlreiche Sonderfälle, für die es optimierte Verfahren gibt. Einige von ihnen sind

- P_{en} und P_{ex} liegen in einer gemeinsamen Zelle. Hier kann viel Zeit gespart werden, da keine Schnittstellen gefunden werden müssen.
- P_{en} liegt auf einer Gitterkante oder sogar auf einem Gitterpunkt. In diesem Fall müssen nicht alle vier umliegenden Gitterpunkte X in die Liste aufgenommen werden, weil deren Gewichte zum Teil ohnehin $\sigma_u(X) = \sigma_v(X) = 0$ wären.
- Das Segment verläuft komplett auf einer Gitterlinie. Hier sind Sonderfälle notwendig, da evtl. die Zelle, in der sich P_{ex} befindet, nie aufgerufen wird. Die Rekursion würde dann nie enden.
- Das Segment verläuft in nordwestliche (oder südwestliche) Richtung und P_{ex} liegt auf einem Gitterpunkt. In diesem Fall würde die P_{ex} beinhaltende Zelle garantiert nicht aufgerufen werden.

Die Query erfolgt ähnlich zu Algorithmus 2 (Zeile 26), nur dass hier noch das Ergebnis durch 2 geteilt wird, da dieser Faktor in dieser Implementation nicht in τ enthalten ist.

3.2 Arbeitsweise

Nun werden die Unterschiede in der Arbeitsweise der beiden Algorithmen betrachtet, indem in Abbildung 3.1 für ein gegebenes Segment die ausgewerteten Punkte nebeneinandergestellt werden.

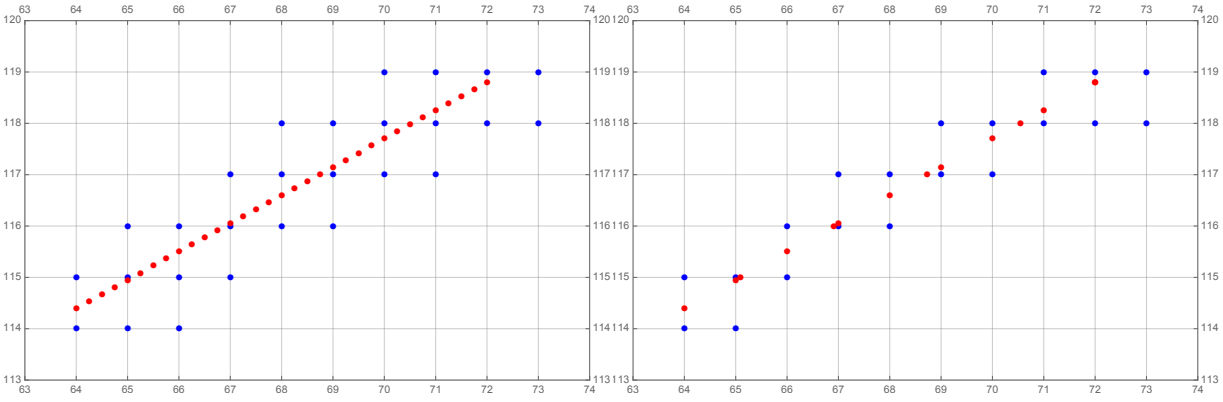

Abbildung 3.1: Gegeneinanderstellung der ausgewerteten Punkte bei SOTA (links) und SWEN (rechts) für ein gegebenes Segment. Rot: Schnittstellen auf dem Segment, blau: Gitterpunkte, die später für die Auswertung benutzt werden.

Während in SOTA die Schnittstellen äquidistant verteilt sind und somit innerhalb der Zellen liegen, liegen diese bei SWEN ausschließlich auf den Gitterkanten. Je nach Länge und geographischer Breite eines Segments variiert diese Anzahl an Schnittstellen zwischen den beiden Algorithmen (in Abbildung 3.1 z.B. 14 (SWEN) statt 33 (SOTA)), was nun für alle Segmente in $\mathcal{T}$ untersucht wird.

Tabelle 3.2: Vergleich der Anzahl der Schnittstellen bei SOTA und SWEN für alle Segmente in $\mathcal{T}$

Algorithmus	min.	max.	∅
SOTA	2	33	14,34
SWEN	2	32	6,34

SWEN braucht durchschnittlich weniger als halb so viele Schnittstellen wie SOTA, wie in Tabelle 3.2 erkennbar ist. Selbst im Worst-Case benötigt SWEN nur 32 Schnittstellen, während SOTA auf allen Segmenten, auf denen das der Fall ist, 33 benötigt.

Zu erkennen ist auch, dass bei SWEN weniger Gitterpunkte zum späteren Auswerten (in Abbildung 3.1 z.B. 20 (SWEN) statt 27 (SOTA)) anfallen. Diese Aussage wird ebenfalls für alle Segmente in $\mathcal{T}$ überprüft.

Tabelle 3.3: Vergleich der Anzahl der zur Auswertung benötigten Gitterpunkte bei SOTA und SWEN. Getestet wurden alle Segmente in $\mathcal{T}$.

Algorithmus	min.	max.	∅
SOTA	4	59	12,78
SWEN	4	45	10,60

In Tabelle 3.3 wird deutlich, dass SWEN für sein Ergebnis durchschnittlich weniger Gitterpunkte in die Rechnung mit einbezieht. Auffällig ist, dass bei SOTA im Durchschnitt mehr Gitterpunkte als Schnittstellen anfallen. Das deutet darauf hin, dass SOTA in einer Zelle oft mehr als zwei Schnittstellen setzt, was bei dem Unwissen über den Wind in der Zelle fragwürdig erscheint.

Tatsächlich ist für alle Segmente die Anzahl der Gitterpunkte bei SWEN nach oben durch die der Gitterpunkte bei SOTA begrenzt, was zu einer Beschleunigung der Query führt.

3.3 Genauigkeit

Als nächstes wird gezeigt, dass SOTA und SWEN das SWIPP mit akkuraten Ergebnissen lösen, die den tatsächlichen Wind gut approximieren. Dazu wird per Überlagerung zweier Wirbelfelder ein stetiges Windfeld erzeugt, das auf der gesamten Ebene bekannt ist. Es sollen Erkenntnisse über das Verhalten der Algorithmen bei bestimmten Windverhältnissen, bzw. Segmenten gewonnen werden. Insbesondere werden folgende Einflussfaktoren untersucht: Länge eines Segments, Verlauf

eines Segments durch Gitterpunkte (oder zumindest nahe an ihnen vorbei) sowie Nullstellen und Extrema in der echten Windfunktion auf einem Segment.

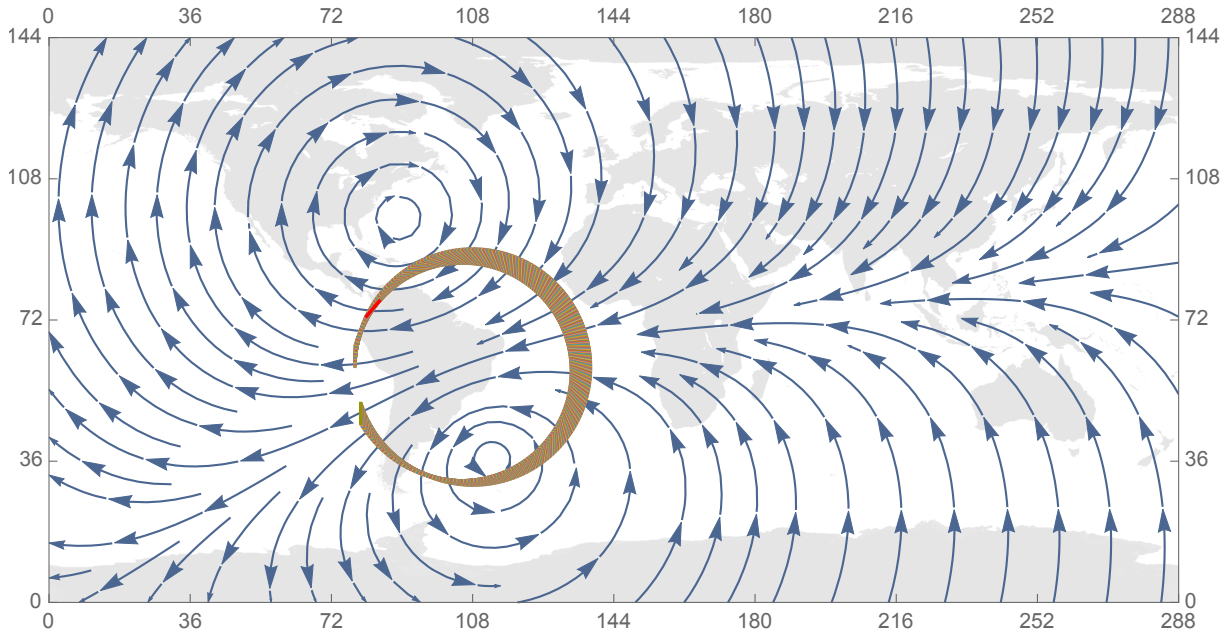

Abbildung 3.2: Übersicht über die getesteten Segmente und den künstlich erzeugten Wind. Die Segmente haben eine Länge von 5 Einheiten. Nummeriert sind die Segmente im Uhrzeigersinn, beginnend auf der linken Seite. Die Feldliniendichte korelliert nicht mit der Windstärke: der obere Wirbel ist doppelt so starke wie der untere und beide nehmen in ihrer Stärke nach außen exponentiell ab.

$k = 1000$ Segmente werden, wie in Abbildung 3.2 zu sehen ist, für die Testreihe so gelegt, dass sie um den Teil des Windfeldes positioniert sind, in dem am meisten Bewegung herrscht. Alle Segmente haben die Länge 5.

An den Gitterpunkten wird der Wind ausgewertet und dient nun als Input der beiden Algorithmen. Diese rechnen anhand dieser diskreten Informationen nach ihren Verfahren den resultierenden Windvektor aus, während der exakte Wind als Wegintegral (vgl. Formel (1.3), aber ohne Betrachtung der Zeit) von der Software bestimmt wird.

Als grafisches Ergebnis dient ein Diagramm, das für jedes Segment und für beide Algorithmen die absoluten Fehler ihrer Ergebnisse bzgl. des tatsächlichen Trackwinds angibt. Der Crosswind wird im gesamten Unterabschnitt vernachlässigt, da er sich ähnlich wie der Trackwind verhält. Mit *Wind* ist in diesem Unterabschnitt der Trackwind gemeint. In Abbildung 3.3 sind die Ergebnisse des Testdurchlaufs dargestellt. Auf den ersten Blick ist zu erkennen, dass die Linie von SWEN meistens dichter zu der x-Achse verläuft, was bedeutet, dass SWEN den Wind akkurater bestimmt. Um diese Aussage zu quantisieren, werden für SOTA und SWEN jeweils die mittleren absoluten und relativen Fehler errechnet. Dazu sei für das i-te Segment ($i \in \{1, 2, \ldots, k\}$) $(w_t)_{\text{SOTA}}(i)$ der von SOTA berechnete Wind, $(w_t)_{\text{SWEN}}(i)$ der von SWEN berechnete Trackwind und $(w_t)_{echt}(i)$ der von der Software berechnete echte Wind. Für jedes Segment und für jeden

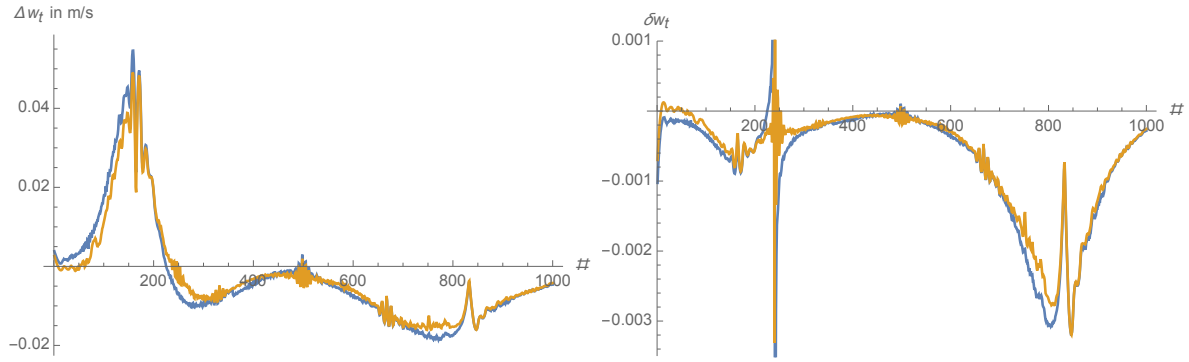

Abbildung 3.3: Grafisches Ergebnis der Genauigkeitsanalyse. Links: absoluter Fehler des Windes von SOTA und SWEN bzgl. des echten Windes, rechts: relativer Fehler des Windes von SOTA und SWEN bzgl. des echten Windes. Blau: SOTA, gelb: SWEN.

Algorithmus sind nun die absoluten Fehler bzgl. des echten Windes wie folgt definiert:

$$\Delta(w_t)_{\text{SOTA}}(i) = |(w_t)_{\text{SOTA}}(i) - (w_t)_{echt}(i)| \tag{3.1a}$$

$$\Delta(w_t)_{\text{SWEN}}(i) = |(w_t)_{\text{SWEN}}(i) - (w_t)_{echt}(i)| \tag{3.1b}$$

Die mittleren absoluten Fehler sind die Mittelwerte dieser Fehler:

$$\Delta_{\varnothing,\text{SOTA}} := \frac{1}{k}\sum_{i=1}^{k} \Delta(w_t)_{\text{SOTA}}(i) = 10,68\,\frac{\text{mm}}{\text{s}} \tag{3.2a}$$

$$\Delta_{\varnothing,\text{SWEN}} := \frac{1}{k}\sum_{i=1}^{k} \Delta(w_t)_{\text{SWEN}}(i) = 9,28\,\frac{\text{mm}}{\text{s}} \tag{3.2b}$$

Dass dieser mittlere Fehler bei SWEN kleiner ist, bedeutet, dass SWEN den Wind besser approximiert, und zwar um durchschnittlich 1,40 mm/s. Auch wenn beide Algorithmen (bis auf die Peaks in Abbildung 3.3 rechts, die durch einen echten Wind von ca. 0 zu erklären sind) bereits keine relativen Fehler von mehr als 0,35% verzeichnen, ist diese Verbesserung von Bedeutung, da dies bestätigt, dass das Konzept von SWEN tatsächlich aufgeht.

Um die Ergebnisse in Gleichung (3.2) in Bezug zur Stärke des tatsächlichen Winds zu setzen, werden für jeden Algorithmus und jedes Segment die relativen Fehler

$$\delta(w_t)_{\text{SOTA}}(i) = \left|\frac{(w_t)_{\text{SOTA}}(i)}{(w_t)_{echt}(i)} - 1\right| \tag{3.3a}$$

$$\delta(w_t)_{\text{SWEN}}(i) = \left|\frac{(w_t)_{\text{SWEN}}(i)}{(w_t)_{echt}(i)} - 1\right| \tag{3.3b}$$

und die mittleren relativen Fehler

$$\delta_{\varnothing,\text{SOTA}} := \frac{1}{k}\sum_{i=1}^{k}\delta(w_t)_{\text{SOTA}}(i) = 0,0776\,\% \tag{3.4a}$$

$$\delta_{\varnothing,\text{SWEN}} := \frac{1}{k}\sum_{i=1}^{k}\delta(w_t)_{\text{SWEN}}(i) = 0,0663\,\% \tag{3.4b}$$

definiert und berechnet. Wie zu erwarten war, ist auch der mittlere relative Fehler bei SWEN geringer als bei SOTA. Die relativen Fehler untscheiden sich um einen Faktor von etwa 1,17.

Es werden nun die Segmente um Nr. 498 untersucht. Dieser Bereich aus Abbildung 3.3 links ist in Abbildung 3.4 vergrößert dargestellt:

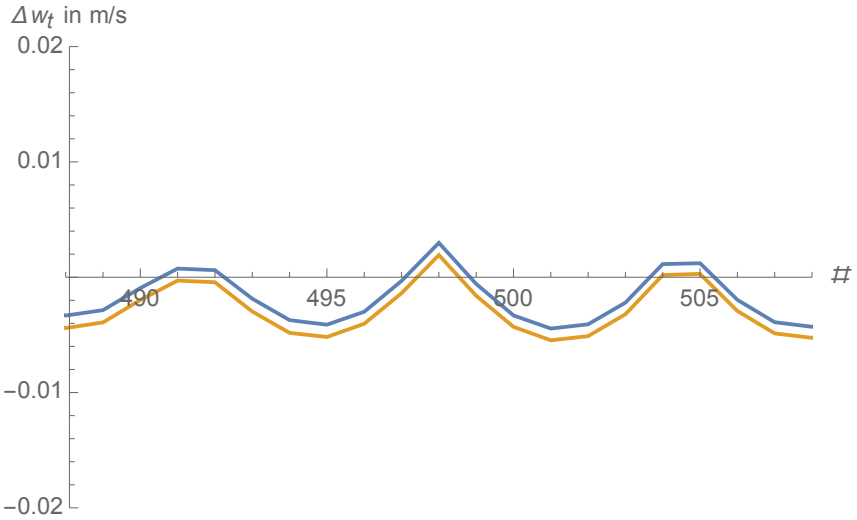

Abbildung 3.4: Ausschnitt aus Abbildung 3.3 (links): absoluter Fehler des Windes von SOTA und SWEN bzgl. des echten Windes. Blau: SOTA, gelb: SWEN

Zu erkennen ist, dass beide Algorithmen alle sechs bis sieben Segmente durch einen Peak in ihrem Graphen den echten Wind extra gut approximieren. Der Graph hat in diesem Bildausschnitt die Form einer Welle. Dieses Phänomen tritt auch auf bei Segmenten um die Nummern 165, 332, 667 und 833, was in Abbildung 3.3 an den gelben Flecken erkennbar ist, die durch die genannten Wellen entstehen. Das liegt daran, dass gerade diese Segmente (insbesondere Segment Nr. 498) nahezu perfekt auf einer Gitterlinie liegen, wodurch sie nahe den Gitterpunkten gelegen sind, die die echten Windinformationen enthalten. Durch diese Nähe steigt die Zuverlässigkeit der Interpolation. In abgeschwächter Form ist dieser Effekt auch zu beobachten bei Segmenten, die genau diagonal verlaufen und dabei nur knapp an den jeweiligen Gitterpunkten vorbeilaufen.

In Abbildung 3.4 ist außerdem zu erkennen, dass SOTA hier genauer arbeitet als SWEN. Eine Analyse der Segmente ergibt, dass gerade an den Gitterpunkten um P_{ex} besonders starker Wind in Trackrichtung herrscht. Das liegt an der Krümmung des Windfelds, durch die am Ende des Segments ein stärkerer Trackwind herrscht als am Anfang. In SOTA erhalten diese äußeren

Gitterpunkte höhere Gewichte, da P_{ex} hier den Wind auf einem größeren Anteil des Segments repräsentiert als bei SWEN: in SWEN repräsentiert $P_{ex} = P_n$ den Wind auf der Hälfte der Strecke bis P_{n-1}. P_{n-1} liegt nun auf einer Gitterkante und wird somit nicht mehr von den äußeren Gitterpunkten beeinflusst, die P_{ex} beeinflussen. Benutzt SOTA in der Zelle von P_{ex} keine weitere Schnittstelle, so ist der Windanteil von P_{ex} am Gesamtsegment hier höher als bei SWEN. Falls SOTA in dieser Zelle eine zusätzliche Schnittstelle benutzt, wird diese ebenfalls von den äußeren Gitterpunkten beeinflusst, weil sie im Inneren der Zelle liegt. SOTA rechnet also besonders auf jenen Segmenten besonders akkurat (und akkurater als SWEN), bei denen am Anfang oder am Ende ein Wind herrscht, der stärker als der Durschnittswind weht und das Gesamtergebnis in die passende Richtung verändert.

In Abbildung 3.3 ist außerdem zu erkennen, dass der Track scheinbar keinen Einfluss auf die Genauigkeit hat. In seperaten Tests wurde diese Hypothese untersucht, indem der Track variiert wurde, und konnte anschließend bestätigt werden. Ebenfalls wurde für einen konstanten Track und eine konstante Länge der Anfangspunkt innerhalb seiner Zelle verschoben. Auch hierbei änderte sich die Genauigkeit nicht – bis auf die oben angesprochene Verbesserung, wenn das Segment nahe entlang der Gitterpunkte verläuft.

Eine Analyse bzgl. der Länge der Segmente ergibt, dass diese keinen signifikanten Einfluss auf die Genauigkeit von SOTA und SWEN hat.

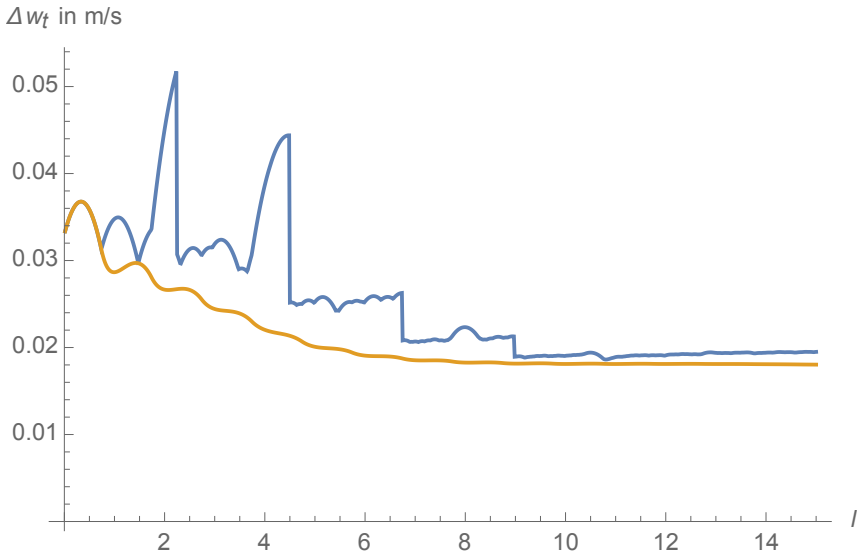

Abbildung 3.5: Absoluter Fehler des von SOTA (blau) und von SWEN (gelb) errechneten Windes bzgl. des echten Windes in Abhängigkeit von der Segmentlänge l

In Abbildung 3.5 ist zu erkennen, dass die absoluten Fehler für größere Längen scheinbar sinken. Dies liegt jedoch zum größten Teil an den entsprechenden herrschenden Winden und ist für beliebige Segmente mit diesen Längen nicht reproduzierbar. Dass in dem Graphen zu SOTA so große Sprünge auftreten, liegt daran, dass die Anzahl der Schnittstellen von SOTA von der Länge abhängt. Wird eine bestimmte Länge überschritten, werden mehr Schnittstellen verwendet, wodurch das Ergebnis genauer wird. Dass beide Algorithmen bis zu einer bestimmten Länge

von ca. 0,8 exakt das selbe Ergebnis ausrechnen, liegt daran, dass in diesem Fall P_{en} und P_{ex} in einer gemeinsamen Zelle liegen und dadurch bei SWEN keine weitere Schnittstelle zustande kommt, während SOTA aufgrund der geringen Segmentlänge ebenfalls keine weitere Schnittstelle zur Berechnung heranzieht.

In den Abbildungen 3.3 und 3.5 konnte außerdem gezeigt werden, dass die analytischen Merkmale der echten Windfunktion (Nullstellen und Extrema) keinen Einfluss auf die Genauigkeit haben. Die absoluten Fehler werden größer, wenn die Winde stärker wehen; die relativen Fehler steigen, wenn der echte Gesamtwind auf einem Segment nahezu null ist. Dies sind aber Befunde, die schon von vornherein zu erwarten waren und somit für die Auswertung irrelevant sind.

3.4 Geschwindigkeit während des Preprocessings

Neben akkuraten Ergebnissen ist die Anforderung an SWEN, dass er das SWIPP schneller löst als SOTA. Dass dies der Fall ist, wird in den folgenden drei Unterabschnitten gezeigt. Zunächst erfolgen theoretische Analysen bzgl. der asymptotische Laufzeit beider Algorithmen und der Anzahl der verwendeten Rechenoperationen. Anschließend werden SWEN und SOTA gegeneinander laufen gelassen und es wird ermittelt, ob SWEN auch in der Praxis schneller arbeitet.

3.4.1 Analyse der asymptotischen Laufzeit

In diesem Unterabschnitt werden die asymptotischen Laufzeiten der beiden Algorithmen zum Lösen des SWIPP miteinander verglichen. Die asymptotische Laufzeit von SOTA ist nach Proposition 1 für ein Segment der Länge l_K auf der Erdkugeloberfläche durch $\mathcal{O}(2^{\frac{l_K}{300\,000}})$ gegeben, während in SWEN die asymptotische Laufzeit für ein Segment der Länge l auf der Ebene nach Proposition 2 durch $\mathcal{O}(l)$ gegeben ist. Die Längen l_K und l sind zwar nicht identisch, aber sie korellieren miteinander: je länger l_K, desto länger ist auch l. Dieser Zusammenhang ist für die Analyse der asymptotischen Laufzeit ausreichend. In beiden Algorithmen hängt die asymptotische Laufzeit also von der Länge eines Segments ab. Die Position eines einzigen Randpunktes, der Track α oder die geographische Breite μ_e sind dafür irrelevant.

In SOTA steigt die Laufzeit exponentiell mit der Segmentlänge, während in SWEN die Laufzeit linear von davon abhängt. Damit besitzt SWEN klar die günstigere asymptotische Laufzeit. Dass sich dieser Vorsprung tatsächlich bemerkbar macht, wird im folgenden Unterabschnitt gezeigt.

3.4.2 Zählung und Vergleich der Rechenoperationen

Zum Vergleich der Rechenoperationen wird von beiden Algorithmen auf allen Segmenten in $\mathcal{T}$ der Wind berechnet. Hier kommt es jedoch nicht auf das Windergebnis an; es wird nur gezählt, welche Rechenoperationen wie oft tatsächlich verwendet werden. Für alle Operationen und Funktionen, die in Tabelle 3.4 aufgeführt sind, wird jeweils ein Zähler mit 0 initialisiert. Der Programmcode wird dahingehend erweitert, dass für jeden Operationsaufruf der entsprechende Zähler um 1 erhöht wird. Dazu wird für jede Zeile vor ihr eine zusätzliche mit den entsprechenden Zählermanipulationsanweisungen eingefügt. Um Rechenzeit zu sparen, und da die Wind-Ergebnisse der Algorithmen für diesen Test irrelevant sind, werden anschließend Code-Blöcke entfernt, die ausschließlich dieses Wind-Ergebnis und nicht den weiteren Rechenverlauf betreffen. Alle Anweisungen, die dafür entscheidend sind, wie ein Algorithmus an einer bestimmten Stelle weiterrechnet,

müssen offensichtlich weiterhin ausgeführt werden. Anschließend werden für jeden Algorithmus und für jede Rechenoperation die Vorkommen in allen Segmentberechnungen aufsummiert und gegeneinander graphisch in Abbildung 3.6 dargestellt.

Für jede Rechenoperation r sei $a_{\text{SOTA}}(r)$ die Anzahl der Vorkommen in SOTA, und $a_{\text{SWEN}}(r)$ die Anzahl der Vorkommen in SWEN. Als *Einsparung* wird für r folgender Quotient definiert:

$$Einsparung(r) := \frac{a_{\text{SOTA}}(r) - a_{\text{SWEN}}(r)}{a_{\text{SOTA}}(r)} \tag{3.5}$$

Tabelle 3.4: Aufsummierte Rechenoperationen bei SWEN und SOTA und jeweilige Einsparung bei SWEN bzgl. SOTA. Getestet wurden alle Segmente in $\mathcal{T}$

r	$a_{\text{SOTA}}(r)$	$a_{\text{SWEN}}(r)$	Einsparung(r) in %
$+$	119 211 147	24 305 589	79,61
$-$	79 982 121	38 153 816	52,30
$\cdot$	132 049 422	27 320 675	79,31
$\div$	17 234 403	5 490 959	68,14
$==$	74 280 335	14 163 431	80,93
$\neq$	5 317 797	2 349 583	55,82
$<$	370 857	4 014 235	−982,42
$\leq$	76 299 860	14 840	99,98
$>$	5 317 797	2 761 583	48,07
$\geq$	0	1 015	*k.A.*
Length	21 271 188	3 544 093	83,34
Append	23 366 019	9 837 186	57,90
$\lfloor\rfloor$	11 321 328	1 517 872	86,59
sin	6 402 378	2 349 583	63,30
cos	6 430 368	2 349 583	63,46
arccos	370 857	0	100,00
arctan	5 265 900	2 336 242	55,63
$\sqrt{}$	0	2 294 133	*k.A.*
Mod	5 317 797	2 349 583	55,82
Min	685 734	0	100,00
Abs	685 734	0	100,00
Pow	342 867	0	100,00

In Tabelle 3.4 wird deutlich, dass SWEN von fast allen Rechenoperationen weniger Gebrauch macht als SOTA. Dass bei SWEN mehr als zehn mal so viele „<“-Vergleiche wie bei SOTA anfallen, wird dadurch kompensiert, dass SOTA dafür mehr als 5 000 mal so oft $\leq$ verwendet. Dies passiert vor allem in den For-Schleifen, die in dieser Implementierung verwendet wurden. Während SWEN als einziger der beiden Algorithmen $\geq$ und $\sqrt{}$ verwendet (letzte Operation für die Entfernungs-

messung auf der Ebene, die bei SOTA nicht verwendet wird, um die Anzahl der Schnittstellen nicht zu verfälschen), kommen die Operationen *Min*, *Abs* und *Pow* nur bei SOTA vor.

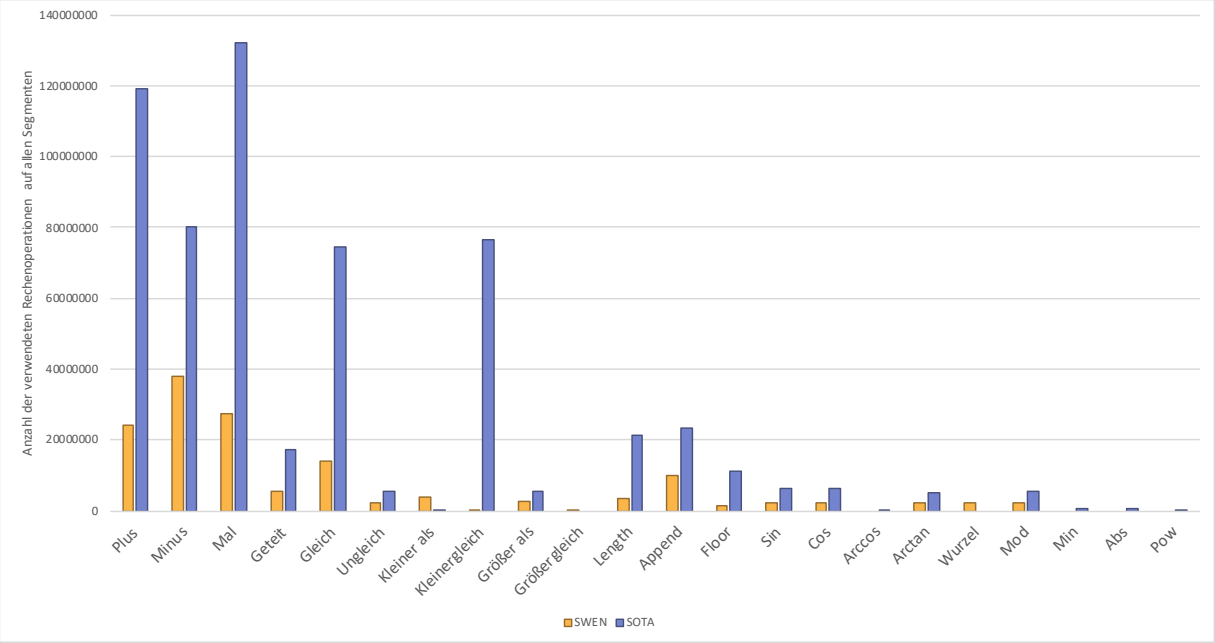

Abbildung 3.6: Anzahl der verwendeten Rechenoperationen in SWEN und SOTA bei der Berechnung aller Segmente in $\mathcal{T}$

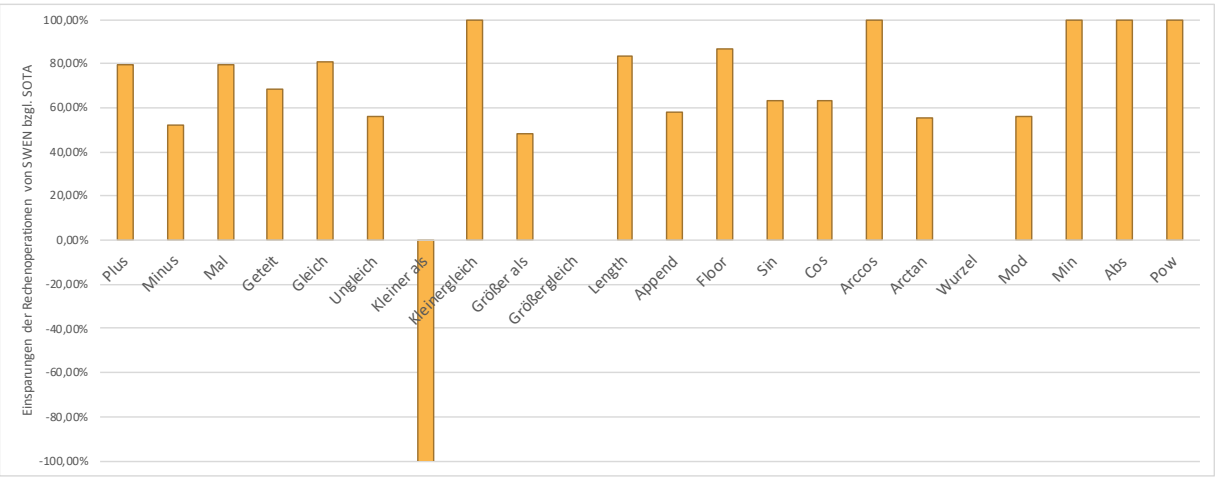

Abbildung 3.7: Einsparung der Rechenoperationen bei SWEN bzgl. SOTA nach Definition (3.5) bei der Berechnung aller Segmente in $\mathcal{T}$

In allen anderen Rechenoperation ist SWEN SOTA deutlich überlegen: SWEN benötigt fast durchgehend weniger als die Hälfte der Operationen, die SOTA benutzt. Die Anzahlen der Rechenoperationen bzw. deren Einsparungen sind in Abbildung 3.6 bzw. Abbildung 3.7 graphisch dargestellt. Hier wird das Ausmaß der Unterschiede deutlich.

Es wurde bestätigt, dass SWEN nicht nur theoretisch (siehe Unterabschnitt 3.4.1) weniger aufwändig ist, sondern auch in der Praxis deutlich weniger Rechenoperationen verwendet als sein Vorgänger SOTA.

3.4.3 Laufzeitanalyse

Zu guter Letzt wird das Verhalten der tatsächlichen Laufzeit in Abhängigkeit verschiedener Eigenschaften der Segmente betrachtet, auf denen der Wind bestimmt werden soll. Hier zeigt sich, ob SWEN tatsächlich schneller arbeitet als SOTA.

Als Arbeitsareal dient $\mathcal{T}$. Aus diesen 370 857 Segmenten werden $k = 25\,000$ zufällig ausgewählt. Diese Menge wird $\mathcal{T}_L$ genannt. Das i-te Segment ($i \in \{1, 2, \ldots, k\}$) wird $m = 50$ mal jeweils erst von SOTA und dann von SWEN bearbeitet. Im j-ten Durchlauf ($j \in \{1, 2, \ldots, m\}$) werden die für das Preprocessing benötigten Zeiten $t_{\text{SOTA},i,j}$ für SOTA und $t_{\text{SWEN},i,j}$ für SWEN festgehalten. Von diesen m Zeiten pro Segment und pro Algorithmus wird jeweils ein Mittelwert gebildet:

$$t_{\text{SOTA},i} := \frac{1}{m} \sum_{j=1}^{m} t_{\text{SOTA},i,j} \tag{3.6a}$$

$$t_{\text{SWEN},i} := \frac{1}{m} \sum_{j=1}^{m} t_{\text{SWEN},i,j} \tag{3.6b}$$

Für das i-te Segment wird nun der *Speedup(i)* definiert als

$$Speedup(i) := \frac{t_{\text{SOTA},i}}{t_{\text{SWEN},i}}. \tag{3.7}$$

Er gibt den Faktor an, um den SWEN auf dem i-ten Segment schneller arbeitet als SOTA.

Für jedes Segment werden zusätzlich Anfangs- und Endpunkt P_{en} und P_{ex}, der Track α und seine geographische Breite μ_e festgehalten. Letztere ist für die Auswertung interessant, da die Anzahl der Schnittstellen in SOTA davon abhängt (vgl. Formel(2.1)).

Als Wetterinformationen dienen echte Daten von November 2016.

In den folgenden Abbildungen (3.8, 3.9 und 3.10) ist jeweils der Speedup in Abhängigkeit einer Eigenschaft der Segmente aufgetragen. Jeweils ein Punkt steht für ein getestetes Segment. Dabei werden die Punkte nach der Anzahl der Schnittstellen des jeweiligen Segments in SOTA farblich codiert: Blau: 3 - Gelb: 5 - Grün: 9 - Rot: 17 - Violett: 33. Der Fall, dass SOTA gar keine zusätzlichen Schnittstellen benutzte, trat bei keinem Segment in $\mathcal{T}_L$ ein. Jeder Punkt, der sich oberhalb der Eins-Linie (bzw. außerhalb der grauen Fläche) befindet, steht für ein Segment, das von SWEN schneller berechnet wurde, jeder Punkt unterhalb dieser Linie für eins, auf dem SOTA schneller war.

Es wird nun der Einfluss der Länge l eines Segments auf den Speedup untersucht.

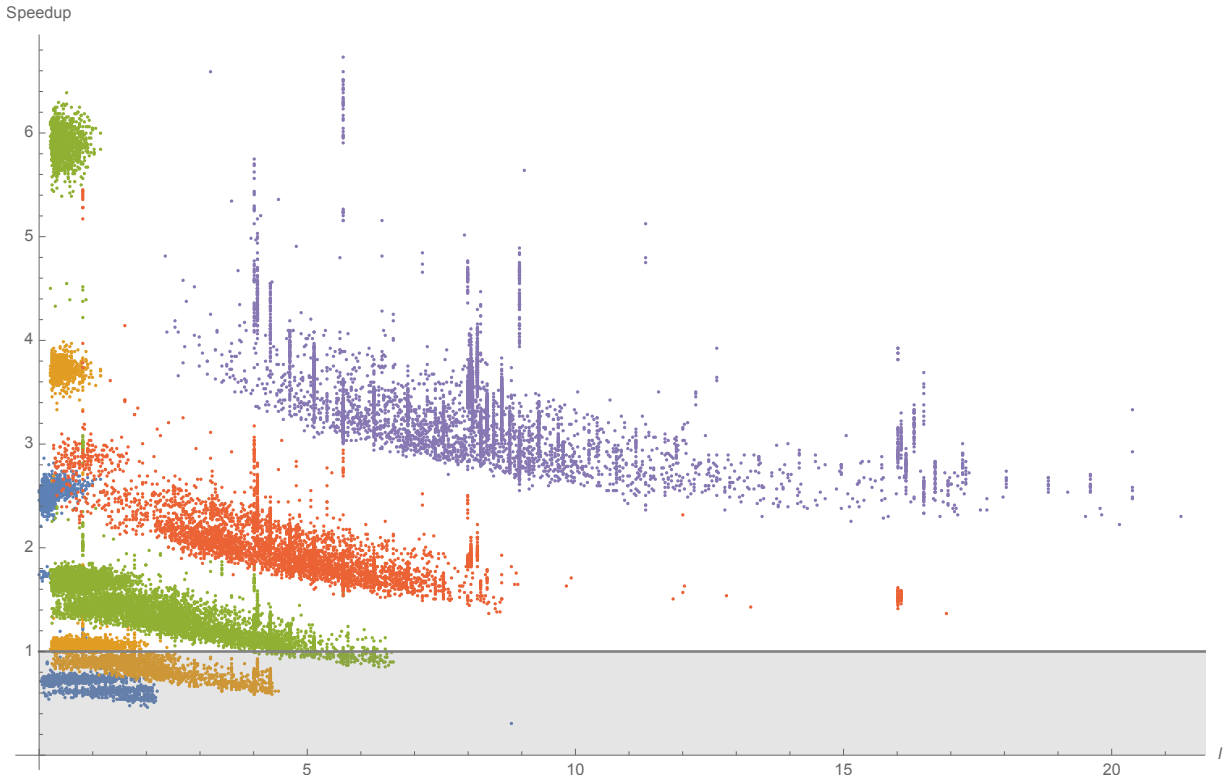

Abbildung 3.8: Speedup in Abhängigkeit der Länge l der Segmente in $\mathcal{T}_L$

Wie in Abbildung 3.8 zu erkennen ist, hängt der Speedup klar von der Anzahl der Schnittstellen in SOTA ab: je mehr Schnittstellen SOTA benutzt, desto größer ist der Speedup. Das ergibt Sinn, da SOTA für jede Schnittstelle Extra-Rechenzeit benötigt. Für eine gegebene Anzahl an Schnittstellen in SOTA schneidet SWEN bei kürzeren Distanzen besser ab, da die Distanz hier maßgebend für die Anzahl der Schnittstellen ist, da für jeden weiteren Schnittpunkt mit dem Gitter eine weitere Schnittstelle anfällt. Und je länger ein Segment ist, desto mehr Schnittpunkte mit dem Gitter sind zu erwarten. Während auf kurzen Segmenten (ca. $1 < d < 7$) SWEN häufig langsamer arbeitet als SOTA, ist der Speedup für längere Segmente deutlich höher. Auf kurzen Segmenten benötigen beide Algorithmen allerdings nicht so viel Rechenzeit wie auf langen, da hier weniger innere Punkte anfallen. Der Großteil der Zeit wird also bei den langen Segmenten gespart, was die Zeitverluste der kürzeren deutlich überwiegt. Erwähnenswert sind auch einige Segmente der Länge $d \leq 1$, auch wenn hier absolut kaum Rechenzeit gespart wird: hier treten teilweise Speedups von 6 und höher auf. Diese Segmente sind genau jene, die sehr weit nördlich bzw. südlich liegen und gleichzeitig sehr kurz sind. In SOTA werden aufgrund der geographischen Breite bis zu sieben innere Punkte benutzt (nach Formel (2.1), siehe grüne Punktwolke oben links in Abbildung 3.8). In SWEN spielt die geographische Breite keine Rolle, weswegen nicht mehr Schnittstellen als bei Segmenten, die näher am Äquator liegen, gewählt werden.

Welchen Einfluss die geographische Breite (im Folgenden nur *Breite* genannt) auf den Speedup

eines Segments hat, wird nun genauer untersucht. Die Breite wird hierfür mit

$$\mu_e := \min\{|(P_{en})_\mu|, |(P_{ex})_\mu|\} \tag{3.8}$$

präzise definiert. Dies ist auch die Definition der Breite, die in SOTA verwendet wird, um die Anzahl der Schnittstellen zu bestimmen.

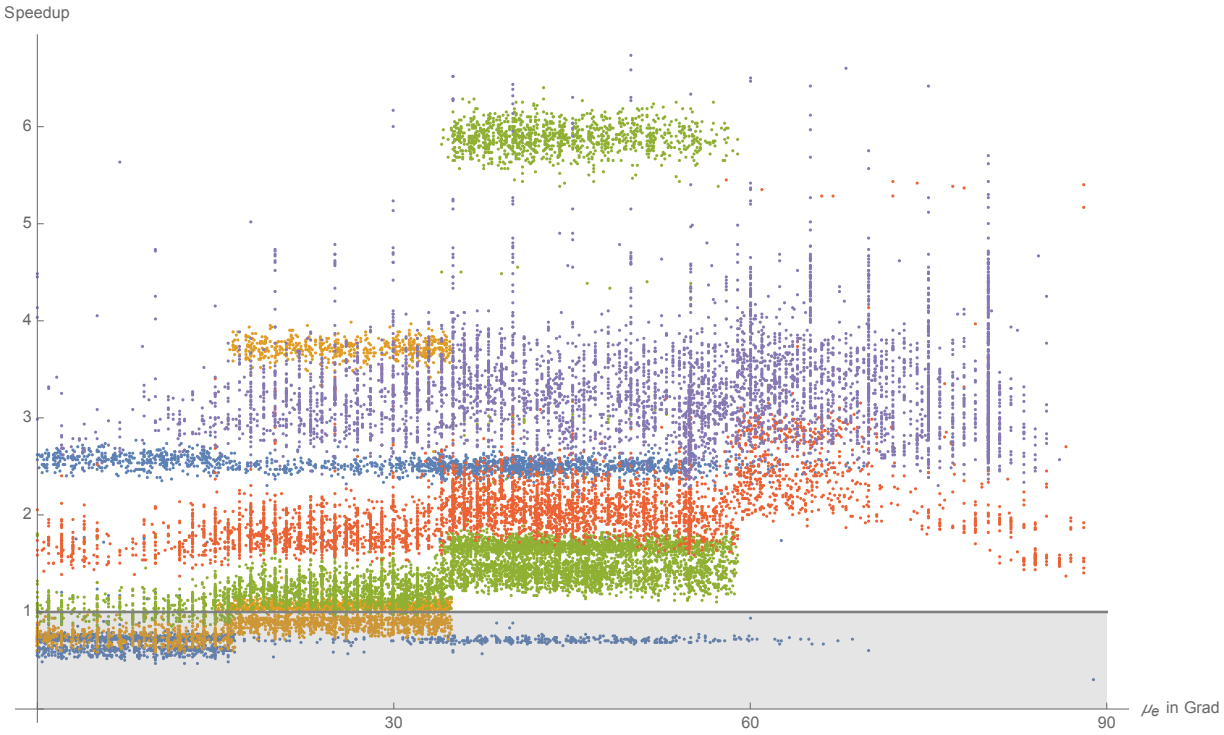

Abbildung 3.9: Speedup in Abhängigkeit der geographischen Breite μ_e der Segmente in $\mathcal{T}_L$

In Abbildung 3.9 ist zu erkennen, dass SWEN desto besser abschneidet, je weiter ein Segment vom Äquator entfernt ist. Das leuchtet ein, da die Anzahl der Schnittstellen n in SOTA (und damit n) mit der Breite μ_e wächst. Erkennbar ist außerdem, dass die einzelnen Punktwolken sich jeweils horizontal ausbreiten und rechteckige Umrisse haben. Das bedeutet, dass der Speedup nur von diskreten Stufen von μ_e abhängt. Das lässt sich damit erklären, dass n in SOTA von $\lfloor 3,5 \cdot \sin \mu_e \rfloor$ abhängt. Damit gibt es vier Klassen von Breiten, die in SOTA jeweils den gleichen Einfluss auf n haben. Für eine gegebene Länge l_K eines Segments auf der Erdkugeloberfläche gilt in SOTA also:

$$n = 2 \cdot 2^{\lfloor \frac{l_K}{300\,000\,\mathrm{m}} \rfloor} \cdot \begin{cases} 1 & 0\,^\circ \leq \mu_e < \arcsin \frac{1}{3,5} \\ 2 & \arcsin \frac{1}{3,5}\,^\circ \leq \mu_e < \arcsin \frac{2}{3,5} \\ 4 & \arcsin \frac{2}{3,5}\,^\circ \leq \mu_e < \arcsin \frac{3}{3,5} \\ 8 & \arcsin \frac{3}{3,5}\,^\circ \leq \mu_e \leq 90\,^\circ \end{cases}, \tag{3.9}$$

was in Abbildung 3.9 gut an den Stufen erkennbar ist.

Ein weiterer Einfluss auf den Speedup ist der Track eines Segments.

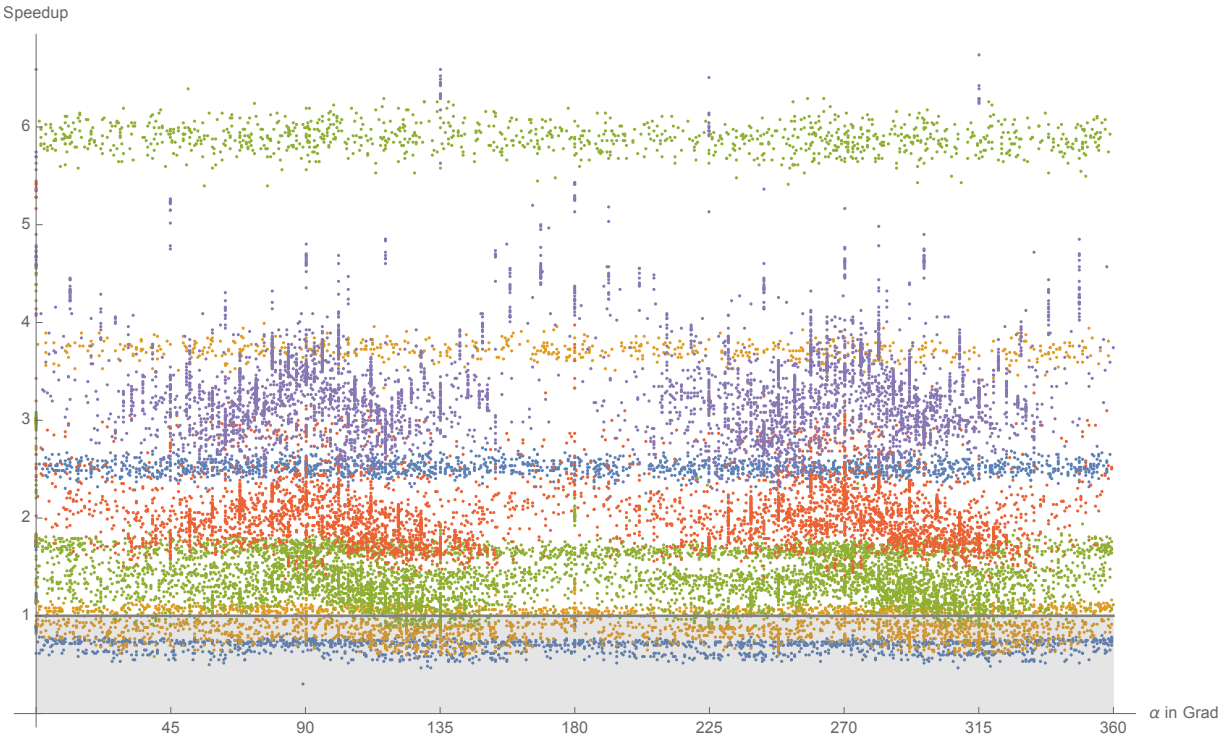

Abbildung 3.10: Speedup in Abhängigkeit des Tracks α der Segmente in $\mathcal{T}_L$

Dass in Abbildung 3.10 in Ost- und West-Richtung (90 ° und 270 °) vergleichsweise dichte Punktewolken liegen, hat zur Ursache, dass besonders viele Segmente in (fast) horizontaler Richtung verlaufen. Dazu gehören unter anderem die vielen Verbindungen über dem (insbesondere Nord-)Atlantik oder dem Pazifik. Auf den zweiten Blick ist außerdem zu erkennen, dass bei Segmenten, deren Track in einer der vier Haupthimmelsrichtungen verläuft, ein größerer Speedup zu erwarten ist. Das ist dadurch zu erklären, dass bei SWEN auf diesen Segmenten nicht so viele Schnittstellen anfallen, da bspw. bei einem genau nach Norden verlaufenden Segment kein einziger Längengrad geschnitten wird. Ein Segment, dessen Track genau 45 ° beträgt, schneidet dagegen bei selber Länge zwar nur ca. $\frac{1}{\sqrt{2}}$ der Breitengrade, dafür aber zusätzlich genau so viele Längengrade. Insgesamt schneidet es also ungefähr $\sqrt{2}$ mal so viele Gitterlinien wie ein Segment, das in eine der vier Haupthimmelsrichtungen verläuft. Auch wenn diese Angabe natürlich nur eine großzügige Abschätzung ist, die erst für sehr lange Segmente genau wird, ist dieser Effekt in abgeschwächter Form in Abbildung 3.10 sichtbar.

Nun wird der tatsächliche Speedup analysiert. Zunächst erfolgt eine Einteilung in Klassen, sodass analysiert werden kann, wie die Speedups verteilt sind.

Tabelle 3.6: Kleinster, größter und mittlerer Speedup der Segmente in $\mathcal{T}_L$ in Abhängigkeit der Anzahl der Schnittstellen in SOTA

$n+1$	Vorkommen	Min	Max	∅
3	3467	0,31	2,86	1,67
5	3527	0,58	3,98	1,41
9	7718	0,85	6,40	2,07
17	4976	1,36	10,68	2,20
33	5312	2,22	6,73	3,31
gesamt	25 000	0,31	10,68	**2,21**

Tabelle 3.5: Einteilung der Speedups der Segmente in $\mathcal{T}_L$ in Klassen. Ein Segment wird einer Klasse zugeordnet, wenn sein Speedup größer als die Untergrenze und kleiner gleich der Obergrenze der Klasse ist.

Speedup		Vorkommen
von	bis	
0,0000	0,5000	6
0,5000	0,7071	999
0,7071	1,0000	2675
1,0000	1,4142	4414
1,4142	2,0000	5441
2,0000	2,8284	4834
2,8284	4,0000	4954
4,0000	5,6569	457
5,6569	8,0000	1125
8,0000	∞	95

Zu erkennen ist, dass knapp 80 % der Segmente Speedups von 1,0 bis 4,0 verzeichnen. Dies lässt sich besonders gut in Abbildung 3.9 sehen. Die Segmente, die Speedups von 5,6569 bis 8,0 haben, sind gerade die, die sehr nah an einem der beiden Pole liegen und verhältnismäßig lang sind (obere grüne Punktwolke in allen Abbildungen). Die 95 Segmente, deren Speedup höher als 8 ist liegen noch näher an einem der Pole und sind extrem kurz, sodass SOTA hier 17 Schnittstellen benutzt. Die entsprechende Punktwolke ist in den Abbildungen 3.8 bis 3.10 nicht mehr im Abbildungsbereich.

Abhängig von der Anzahl $n+1$ der Schnittstellen in SOTA werden nun kleinster, größter und mittlerer Speedup ermittelt. Unüberraschenderweise tritt der kleinste Speedup (0,31) in Tabelle 3.6 bei einem sehr kurzen Segment auf, für das SOTA lediglich eine zusätzliche Schnittstelle benutzt. Der größte Speedup (10,68) gehört zu der oben angesprochenen Menge an Segmen-

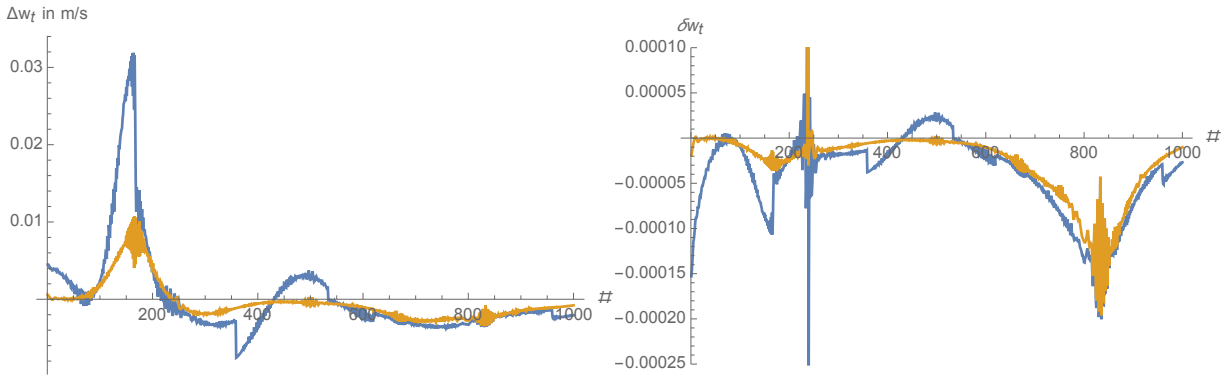

Abbildung 4.1: Absoluter Fehler des Winds von SOTA und SWEN bzgl. des echten Winds (links) und relativer Fehler des Winds von SOTA und SWEN bzgl. des echten Winds (rechts). Blau: SOTA, gelb: SWEN

ten, bei denen SOTA aufgrund ihrer Länge und geographischer Breite 17 Schnittstellen benutzt. Währenddessen benutzt SWEN nur sehr wenige Schnittstellen. Für ein zufälliges Segment ist zu erwarten, dass SOTA 2,21 mal so viel Zeit benötigt wie SWEN.

4 Ausblick: Verfeinerung des Wettergitters

Es ist geplant, für die Windberechnung in Zukunft einen Wetterdatensatz zu verwenden, in dem die Informationen in einem Gitter vorliegen, das einen Abstand von nur 0,25 ° hat. Das entspricht einer Entfernung von etwa 25 km [Wet17]. Es wird nun untersucht, wie sich diese Gitterverfeinerung auf die Genauigkeits- und Geschwindigkeitsverbesserung von SWEN gegenüber SOTA auswirkt. Da in SWEN die Schnittstellen gerade die Schnittpunkte von Segment und Gitter sind, während in SOTA die Schnittstellen unabhängig vom Gitter sind, wird erwartet, dass der Speedup zwar geringer ausfällt, da SWEN aufgrund des feineren Gitters nun mehr Schnittstellen benutzt als auf dem groben Gitter. Allerdings wird ebenfalls erwartet, dass SWENs Genauigkeit durch das feine Gitter deutlich erhöht wird.

4.1 Auswirkungen auf die Genauigkeit

Zunächst werden die Algorithmen auf dem feinen Gitter bzgl. ihrer Genauigkeit verglichen. Diese Analyse geschieht mit denselben fiktiven Segmenten, die auch in Unterabschnitt 3.3 herangezogen wurden. Auch das hier verwendete Windfeld ist identisch zu dem, das auch auf dem groben Gitter zur Bestimmung der Genauigkeit verwendet wurde. Zunächst wird der Wind an den neuen Gitterpunkten ausgewertet und den Algorithmen zum Input gegeben. Gleichzeitig wird mit *Mathematica* der echte Wind als Wegintegral des Windfelds über die Segmente ausgerechnet. Wie in Abbildung 4.1 zu erkennen ist, lässt sich der tatsächliche Wind durch die nun höhere Präzision der Interpolation von SOTA besser approximieren als auf dem gröberen Gitter (vgl. Abbildung 3.3). Noch auffälliger ist allerdings, dass SWEN in Abbildung 4.1 deutlich kleinere absolute Fehler verzeichnet als SOTA. Die Verbesserung bzgl. der absoluten Fehler auf dem groben Gitter fällt

bei SWEN stärker aus als bei SOTA.

Während die Schnittstellen in SOTA nicht vom Gitter abhängen und dadurch unverändert bleiben, passt SWEN sich automatisch dem nun feineren Gitter an und benutzt somit mehr Schnittstellen, was zu einem genaueren Ergebnis führt. Die in Gleichung 3.2 definierten absoluten Fehler $\Delta_{\varnothing,\text{SOTA}}$ und $\Delta_{\varnothing,\text{SWEN}}$ betragen nun

$$\Delta_{\varnothing,\text{SOTA}} = 3,617\,\frac{\text{mm}}{\text{s}} \quad \text{und} \tag{4.1a}$$

$$\Delta_{\varnothing,\text{SWEN}} = 1,776\,\frac{\text{mm}}{\text{s}}. \tag{4.1b}$$

Die in Gleichung 3.4 definierten relativen Fehler $\delta_{\varnothing,\text{SOTA}}$ und $\delta_{\varnothing,\text{SWEN}}$ betragen jetzt

$$\delta_{\varnothing,\text{SOTA}} = 0,00515\,\% \quad \text{und} \tag{4.2a}$$

$$\delta_{\varnothing,\text{SWEN}} = 0,00285\,\%. \tag{4.2b}$$

Die Werte in den Gleichungen 4.1 und 4.2 bestätigen, dass SWEN auf dem feinen Gitter den Wind weiterhin besser approximiert als SOTA. Der absolute Fehler bei SWEN ist nun um ca. 7,5 mm/s kleiner. Zudem sind der absolute und der relative Fehler von SOTA auf dem neuen Gitter ca. doppelt so hoch wie die Fehler von SWEN, während auf dem alten Gitter diese Unterschiede signifikant kleiner waren (vgl. Gleichungen 3.2 und 3.4).

Dieser zusätzliche Vorsprung von SWEN kommt dadurch zustande, dass er von sich aus das feinere Gitter zu seinem Gunsten ausnutzt und durch mehr Schnittstellen eine höhere Genauigkeit erzielt. Da die verwendeten Schnittstellen in SOTA nicht vom Gitter abhängen, passt SOTA sich nicht automatisch an und liefert als Konsequenz ungenauere Ergebnisse.

4.2 Auswirkungen auf die Geschwindigkeit

Da SWEN auf dem feineren Gitter mehr Schnittstellen zur Berechnung heranzieht, liegt es auf der Hand, dass er nun langsamer arbeitet. In diesem Unterabschnitt soll untersucht werden, ob SWEN dadurch von SOTA zeitlich unterboten wird.

Dazu werden aus $\mathcal{T}$ zufällig $k = 25\,000$ Elemente ausgesucht (dieses neue Testset wird $\mathcal{T}_{L,fein}$ genannt) und jeweils $m = 25$ mal ausgewertet. Für das i-te Segment ($i \in \{1, 2, \ldots, k\}$) und den j-ten Durchlauf ($j \in \{1, 2, \ldots m\}$) ist $t_{\text{SOTA},i,j}$ die von SOTA benötigte Zeit, und $t_{\text{SWEN},i,j}$ die von SWEN benötigte Zeit. Für jedes Segment und jeden Algorithmus werden diese Zeiten analog zu Formel (3.6) gemittelt. Für jedes Segment ist der Speedup wie in Definition (3.7) definiert.

Für diesen Test stehen keine Winddaten zur Verfügung. Da der Ergebnisvektor allerdings für diesen Test ohnehin irrelevant ist, reicht es für diesen Zweck, ein Gitter mit zufälligen Windvektoren zu verwenden, die der Genauigkeit der des tatsächlichen Gitters entsprechen.

Wie in Abbildung 4.2 an den Punkten innerhalb des grauen Bereichs zu erkennen ist, schneidet SWEN diesmal nicht ganz so gut von der Geschwindigkeit her ab. Bei vielen Segmenten ist SOTA schneller und die Speedups der Segmente, auf denen SWEN schneller rechnet, sind alle kleiner als 5. Eine genauere Angabe darüber liefert Tabelle 4.1. Auch wenn auf einigen Segmenten SOTA ca. acht mal so schnell ist, was einem Speedup von 0,12 entspricht, liegt der durchschnittliche Speedup nach wie vor über 1. Damit ist gezeigt, dass selbst auf dem feineren Gitter SWEN durchschnittlich 1,16 mal so schnell rechnet wie SOTA.

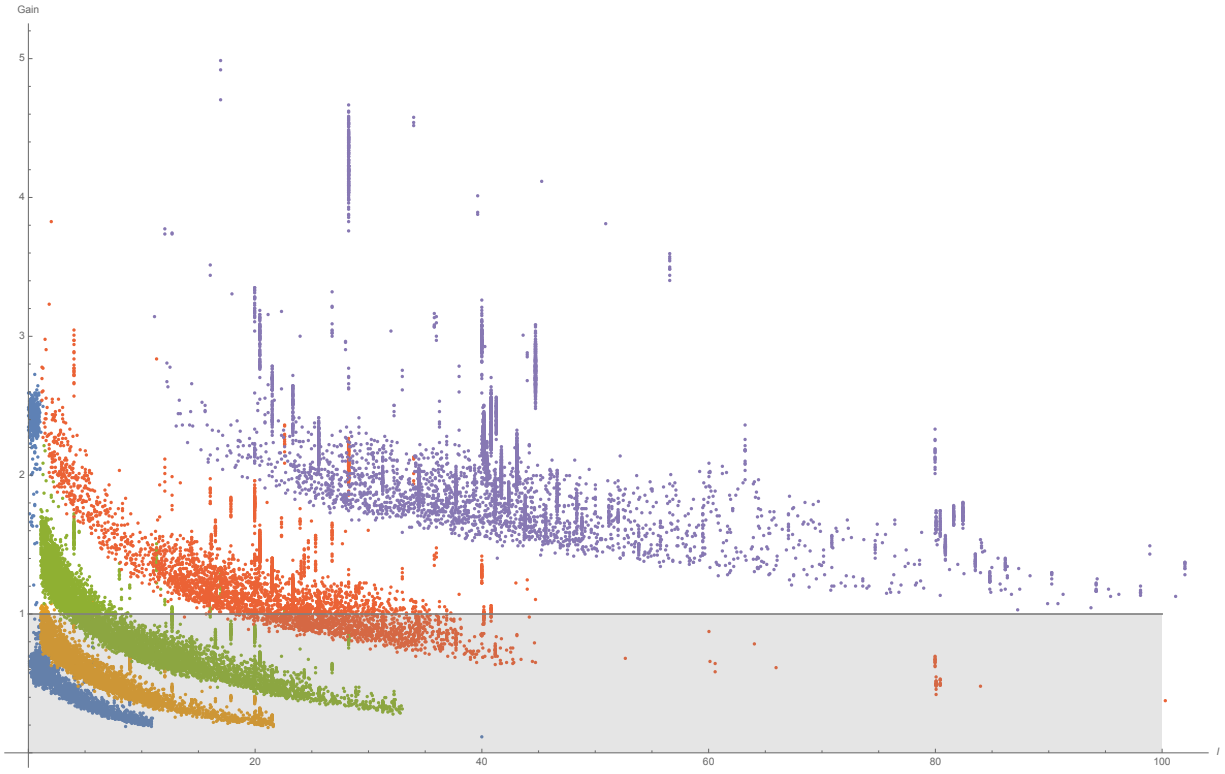

Abbildung 4.2: Speedup in Abhängigkeit der Länge l der Segmente in $\mathcal{T}_{L,fein}$ auf dem feineren Gitter. Der Farbcode ist derselbe wie in Abschnitt 3.4.3

Tabelle 4.1: Kleinster, größter und mittlerer Speedup der Segmente in $\mathcal{T}_{L,fein}$ auf dem feineren Gitter in Abhängigkeit der Anzahl der Schnittstellen in SOTA

$n+1$	Vorkommen	Min	Max	∅
3	3411	0,12	2,72	0,77
5	3616	0,19	1,27	0,58
9	7663	0,28	2,22	0,90
17	4873	0,37	3,82	1,22
33	5437	1,03	4,99	2,11
gesamt	25 000	0,12	4,99	**1,16**

5 Fazit

In dieser Arbeit wurde das Statische Wind-Interpolation-Problem auf einer Planfläche (SWIPP) vorgestellt. Dabei handelt es sich um einen Spezialfall des Wind-Interpolation-Problems (WIP), bei dem der Wind auf einer Luftstraße aus Wetterdaten approximiert werden soll, die nur diskret in einem Gitter vorliegen. Beim SWIPP wird dieser Wind auf einer Ebene betrachtet. Außerdem wird angenommen, dass sich das auszuwertende Segment auf einer konstanten Höhe befindet, auf der die zeitunabhängigen Wetterinformationen vorliegen.

Mit SWEN wurde ein Algorithmus gefunden, der das SWIPP prozesssicher löst. Er wurde in dieser Arbeit mit dem bisherigen State-of-the-Art-Algorithmus SOTA verglichen. Beide Algorithmen unterteilen das Segment für ihre Berechnung an gewissen Schnittstellen, berechnen an diesen Stellen den herrschenden Wind und fahren mit der Annahme fort, dass sich der Wind dazwischen linear verhält. Der Unterschied der beiden Algorithmen liegt darin, wie diese Schnittstellen ausgewählt werden: während in SOTA eine bestimmte Anzahl in konstanten Abständen über das Segment verteilt wird, bilden in SWEN die Schnittpunkte des Segments mit den Kanten des Wettergitters die Schnittstellen.

Im Vergleich der beiden Algorithmen wurde gezeigt, dass SWEN das SWIPP durchschnittlich akkurater löst als SOTA. Dazu wurde ein künstliches Windfeld erzeugt, an den Gitterpunkten ausgewertet und den Algorithmen zum Input gegeben. Gleichzeitig wurde der echte Wind per Software als Wegintegral des Windfeldes über die Segmente bestimmt. Während SOTA auf einigen Segmenten, die in dieser Arbeit genauer untersucht wurden, das Ergebnis präziser approximierte, schaffte SWEN auf den meisten Segmenten ein akkurateres Ergebnis. Durchschnittlich approximierte SWEN das Ergebnis um 1,40 mm/s besser als SOTA. Der durchschnittliche relative Fehler von SOTA ist um einen Faktor von etwa 1,17 größer als der von SWEN.

SWEN arbeitet ebenfalls schneller als SOTA. Die Analyse zum asymptotischen Laufzeitverhalten ergab, dass SWENs asymptotische Laufzeit linear mit der Segmentlänge wächst, während die asymptotische Laufzeit von SOTA exponentiell zu ihr wächst. In der praktischen Analyse wurde gezeigt, dass SOTA deutlich mehr Rechenoperationen benötigt als SWEN.

In der Laufzeitanalyse wurde ermittelt, um welchen Faktor SWEN durchschnittlich schneller rechnet als SOTA. Dieser Faktor wurde mit 2,21 bestimmt. Besonders schnell arbeitet SWEN im Vergleich zu SOTA auf Segmenten, die nahe den Polkappen gelegen sind und gleichzeitig eher kurz sind oder die in eine der vier Haupthimmelsrichtungen verlaufen.

Es wurde zudem untersucht, wie sich beide Algorithmen zum SWIPP auf einem feineren Gitter verhalten. Durch die erhöhte Genauigkeit der Interpolation arbeiteten beide Algorithmen präziser. Die Verbesserungen bzgl. des groben Gitters waren bei SWEN deutlich größer als bei SOTA. Der absolute und der relative Fehler von SOTA waren auf dem feinen Gitter ungefähr doppelt so groß wie die entsprechenden Fehler von SWEN. Im Vergleich zum groben Gitter hat sich der absolute Fehler von SWEN um gut 7,5 mm/s verbessert.

Obwohl SWEN beim Rechnen mit dem neuen Gitter mehr Schnittstellen als auf dem alten Gitter benutzt, ist er dennoch durchschnittlich um einen Faktor von 1,16 schneller als SOTA.

SWEN passt sich also automatisch an das feinere Gitter an und nutzt die höhere Gitterpunktdichte von sich aus zu seinem Vorteil aus, während SOTA auf eine Gitteränderung nicht reagiert. Damit ist gezeigt, dass SWEN auch in Zukunft eine bessere Lösung des SWIPP als SOTA darstellt.

Literatur

[Air18] Airbus S.A.S.: *Price List Press Release*. https://www.airbus.com/content/dam/corporate-topics/publications/backgrounders/Airbus-Commercial-Aircraft-list-prices-2018.pdf. Version: Januar 2018. – Zuletzt aufgerufen am 28.07.2018

[Bar16] Bartels, Sören: *Numerik 3x9: drei Themengebiete in jeweils neun kurzen Kapiteln*. Heidelberg : Springer Spektrum, 2016. – ISBN 978–3–662–48203–2

[Beh15] Behrends, Ehrhard: *Analysis 1: Ein Lernbuch für den sanften Wechsel von der Schule zur Uni*. 6., erw. Aufl. Wiesbaden : Springer Spektrum, 2015. – ISBN 978–3–658–07122–6

[Beu14] Beutelspacher, Albrecht: *Lineare Algebra: eine Einführung in die Wissenschaft der Vektoren, Abbildungen und Matrizen*. 8., aktualisierte Aufl. Wiesbaden : Springer Spektrum, 2014. – ISBN 978–3–658–02412–3

[Deu18] Deutsche Lufthansa AG: *Balance - Nachhaltigkeitsbericht*. https://www.lufthansagroup.com/fileadmin/downloads/de/verantwortung/balance-2018-epaper/epaper/Balance_2018_D.pdf. Version: Juni 2018

[Dij59] Dijkstra, Edsger W.: A note on two problems in connexion with graphs. In: *Numerische mathematik* 1 (1959), Nr. 1, S. 269–271

[Hra12] Hradecky, Simon: *Thunderstorms in Madrid on Jul 26th 2012, landings, diversions, fuel emergencies and Ryanair*. http://avherald.com/h?article=454af355. Version: August 2012. – Zuletzt aufgerufen am 14.08.2018

[HS79] Hale, Francis J. ; Steiger, Arthur R.: Effects of wind on aircraft cruise performance. In: *Journal of Aircraft* 16 (1979), Nr. 6, S. 382–387

[Int17] International Civil Aviation Organization, Civil Aviation Statistics of the World and ICAO staff estimates: *Air passengers carried include both domestic and international aircraft passengers of air carriers registered in the country*. https://data.worldbank.org/indicator/IS.AIR.PSGR. Version: 2017

[Int18] International Air Transport Association: *Jet Fuel Price Monitor*. https://www.iata.org/publications/economics/fuel-monitor/Pages/index.aspx. Version: 2018. – Zuletzt aufgerufen am 28.07.2018

[OO89] O'brien, Edward M. ; Otto, Donald R.: *Aircraft fuel loading management system*. April 1989. – US Patent 4,819,183

[Sny97] Snyder, John P.: *Flattening the earth: two thousand years of map projections*. University of Chicago Press, 1997

[Wet17] Wetterdienst, Deutscher: *WAWFOR (World Aviation Weather FORecast)*. https://www.dwd.de/DE/leistungen/lf_20_wawfor/wawfor_node.html. Version: 2017. – Zuletzt aufgerufen am 07.08.2018

[Wor] World Of Maps: *Earth Spec - the world maps*. http://mapsof.net/uploads/static-maps/earth-spec.jpg. – Zuletzt aufgerufen am 11.07.2018