Konzeption und Implementation einer optimalen Rückführung in ATEO SAM

Studienarbeit

HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT II
INSTITUT FÜR INFORMATIK

eingereicht von: Weidner-Kim, Helmut
geboren am: 05.01. 1958
in: Kettwig

Gutachter: Professor Dr. Klaus Bothe
Betreuer: Dipl. Inf. Nicolas Niestroj

eingereicht am:
19.03. 2014

Verzeichnisse

Inhaltsverzeichnis

Tabellen

Formeln

Abbildungen

1 Einleitung

1.1 Das ATEO-Projekt

Das Graduiertenkolleg *prometei* (*Pro*spektive Gestaltung von *Me*nsch-*Te*chnik-*I*nteraktion) am *Zentrum Mensch-Maschine-Systeme* der TU Berlin [25] untersucht die Mensch-Maschine-Interaktion und entwickelt Methoden, Verfahren und Werkzeuge, um diese schon in frühen Phasen der Gestaltung technischer Systeme zu berücksichtigen. Der Forschungsschwerpunkt 8 von *prometei* ist *Funktionsteilung Mensch-Maschine und Arbeitsteilung Entwickler-Operateur: Zwei Perspektiven auf Mensch-Maschine-Systeme*, betreut von Prof. Dr. Wandke vom *Lehrstuhl für Ingenieurpsychologie und Kognitive Ergonomie* an der Humboldt-Universität zu Berlin, in dessen Rahmen das Projekt *ATEO* (*A*rbeits*t*eilung *E*ntwickler *O*perateur) durchgeführt wird [24].

1.1.1 SAM

Der „Kern“ von ATEO ist *SAM* (*S*ocially *A*ugmented *M*icroworld), bei der es darum geht eine einfache Trackingaufgabe durchzuführen: steuere ein kreisrundes Objekt (das *Tracking Object*, TO) mit Hilfe von Joysticks entlang einer Fahrbahn in einer auf dem Bildschirm angezeigten Strecke. In diese „Mikrowelt“ wird dadurch Komplexität eingeführt, dass diese Aufgabe von zwei Versuchspersonen durchgeführt wird, die nicht miteinander kommunizieren können[1]. Außerdem werden den Beiden unterschiedliche Ziel-

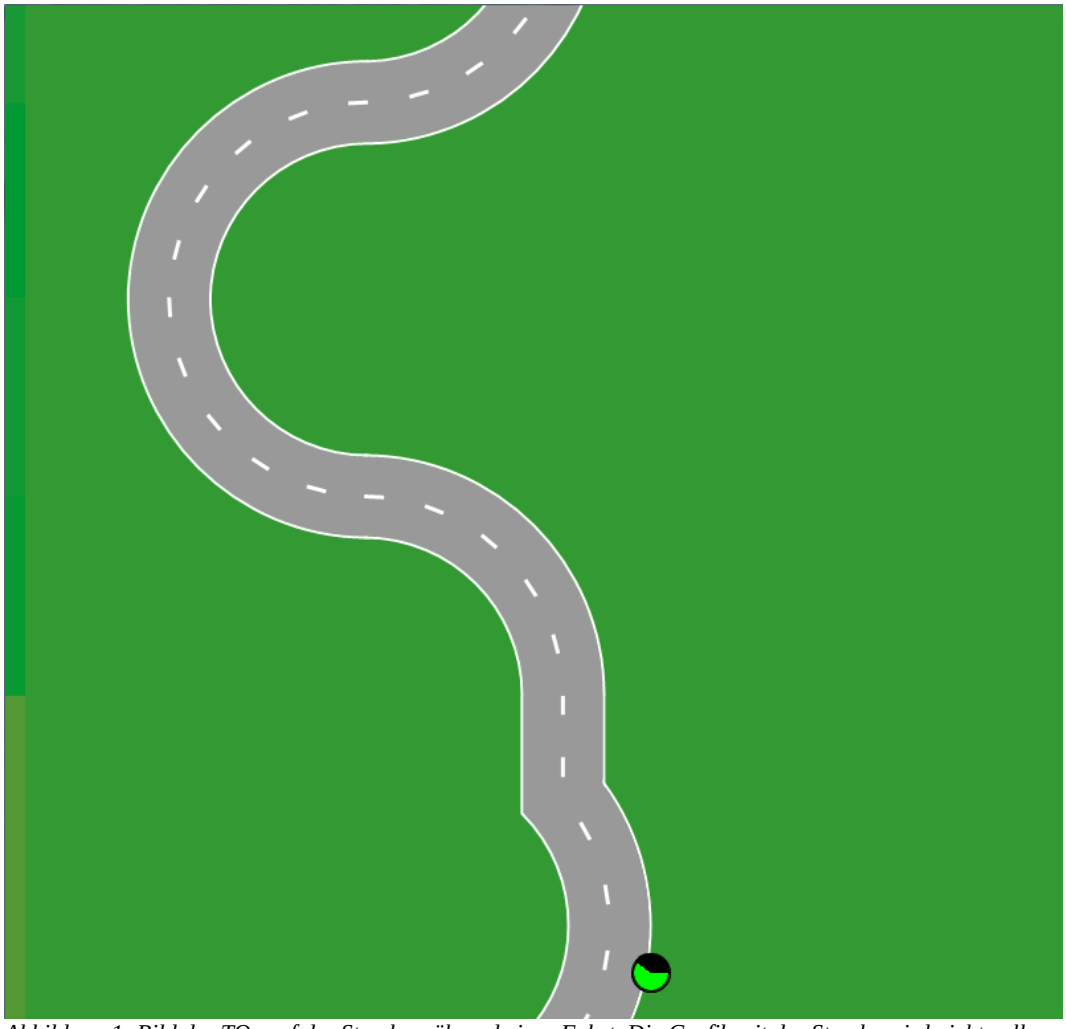

Abbildung 1: Bild des TOs auf der Strecke während einer Fahrt. Die Grafik mit der Strecke wird nicht vollständig angezeigt, sondern nur ein Ausschnitt wie auf diesem Bild, mit dem TO unten.

1 Wenn davon abgesehen wird, dass ein MWB durch Vergleich seiner Lenkbewegungen mit den Bewegungen des TOs Rückschlüsse auf die Lenkbewegungen des anderen MWB ziehen kann.

vorgaben für das Team genannt: dem einem wird mitgeteilt, dass sie vor allem schnell (und in zweiter Linie genau) fahren sollen, dem anderen, dass das TO möglichst genau (und in zweiter Linie schnell) bewegt werden soll. Die Strecke enthält zudem „Schikanen“, nämlich Hindernisse und Gabelungen, die vor allem dazu dienen, den Konflikt zwischen den unterschiedlichen Zielvorgaben an beide MWB zu verstärken.[2]

Die Daten einer Versuchsfahrt werden protokolliert und ausgewertet, allerdings sind die Leistungen der Versuchspersonen nicht der eigentliche Gegenstand der Untersuchung, sondern vielmehr Mittel zum Zweck, weshalb die Versuchspersonen als Teil der Mikrowelt zu betrachten sind sind und deshalb als *Mikroweltenbewohner* (MWB) bezeichnet werden [1, Kap. 3.2].

Im Mittelpunkt des Projekts ATEO stehen nämlich die im vollen Namen genannten zwei Personengruppen, Entwickler und Operateure. Ein Operateur unterstützt die MWB[3], wenn sie das TO steuern, ein Entwickler antizipiert die Versuchssituation und konzipiert eine entsprechende Automatik. Verglichen werden also Leistungen zu unterschiedlichen Zeitpunkten: Entwickler, antizipieren und planen zu einem frühen Zeitpunkt die Interaktion von Operateuren, die zu einem späteren Zeitpunkt mit den von den Entwicklern implementierten automatischen Systemen in Echtzeit interagieren müssen [12, Kap. 1.1].

1.1.2 Weitere Komponenten von ATEO

Für Operateure existiert ein *Operateurarbeitsplatz* (OA), durch den ein menschlicher Operator, der nicht im gleichen Raum wie die MWB sitzt, sie über ein Netzwerk durch auditive und visuelle Hinweise sowie durch Kontrolleingriffe unterstützen kann. Außerdem können die MWB nach Abschluss einer Versuchsfahrt ihre subjektive *mentale Anstrengung* (MA) bewerten, die dann dem Operateur als Information zur Verfügung steht [8]. Für die vorliegende Arbeit sind diese Komponenten von ATEO kaum relevant, weshalb hier nicht weiter auf Details eingegangen wird.

In einer Studie von Saskia Kain wurden von 30 dreiköpfigen Entwicklerteams Automatiken konzipiert, die ähnlich wie ein Operateur die MWB mit Hinweisen und Steuerungseingriffen unterstützen können. „Unterstützen“ bedeutet insbesondere, dass eine Automatik die MWB nicht vollständig ausschalten darf.[4] Anhand dieser Konzepte kann die Antizipationsfähigkeit von Entwicklern in Abhängigkeit verschiedener Parameter[5] untersucht werden [6]. Außerdem wird von Nicolas Niestroj die Leistung der als am besten bewerteten Automatiken mit der eines optimal trainierten Operateurs verglichen [10]. Für die Implementation der von den Entwicklerteams vorgeschlagene Automatiken gibt es das *ATEO Automation Framework* (AAF), das es erlaubt, komplexe Automatiken aus einfachen Automatiken zu entwickeln [5]. Darauf baut eine Benutzerschnittstelle auf, mit der komplexe Automatiken unkompliziert erstellt und konfiguriert werden können [2].

Im weiteren Verlauf des ATEO-Projekts geht es um das Zusammenwirken von Operateur und Automatiken, wenn die MWB von beiden unterstützt werden. Insbesondere sollen Automation und Operateur einander die Verantwortung für eine Funktion bzw. Aufgabe flexibel übertragen können. „Für die Entwickler besteht hier die Herausforderung nun nicht mehr nur in der Festlegung von Verantwortlichkeiten, sondern in der Definition von Regeln und Kriterien für die gegenseitige Kommunikation und Kooperation von Automation und Operateur. Für den Operateur besteht die Herausforderung im Erlernen und korrekter Anwendung dieser Kommunikationsregeln sowie in Auswirkungen dieser Kooperation mit der Automation auf Workload und Situationsbewusstsein.“[24]

Eine weitere, bisher nicht genannte, Komponente von ATEO stellt die *Logfile-Auswertung* (LFA) dar. Die Logdateien umfassen sämtliche Daten über eine Versuchsfahrt, die

2 Details dazu siehe Abschnitt 2.4, S. 7.

3 Dieser Plural wird analog zu anderen Abkürzungen (z.B. PKWs) verwendet

4 Zwar kann eine Automatik vorsehen, dass in bestimmten Fällen die Steuerung des TOs zu 100% automatisch, also unabhängig von den Joystickbewegungen der MWB, erfolgt, aber dies muss im Verlauf einer Fahrt die Ausnahme und nicht die Regel sein.

5 Insbesondere vom Ausmaß der dem Team zur Verfügung stehenden Informationen.

möglicherweise von Interesse sein könnten, und sind entsprechend umfangreich. Mit dem LFA-Tool können für eine bestimmte Untersuchung relevante Daten ausgewählt und verschiedene Versuchsfahrten miteinander verglichen oder zusammengefasst werden [3,4]. Wie sich im Lauf der vorliegenden Arbeit herausstellte, werden dadurch Standards gesetzt bzw. dokumentiert, die bisherigen Auswertungen in früheren Stufen des ATEO-Projekts zu Grunde liegen und aus Gründen der Vergleichbarkeit mit künftigen Untersuchungen kaum geändert werden können.

1.2 Motivation und Ziel der Arbeit

Die von den oben erwähnten Entwicklerteams aufgestellten 30 Automatik-Konzepte wurde von Kain analysiert und alle darin enthaltenen Funktionen dokumentiert. Die für die Implementation dieser Konzepte bzw. Funktionen im AAF benötigten Elementarautomatiken werden einer Gruppe angehender Informatiker unter der Leitung von Kain und Niestroj implementiert.

Viele der insgesamt 30 Automatik-Konzepte, die von den oben erwähnten Entwicklerteams aufgestellt wurden, sehen vor, dass den MWB die optimale Richtung, in die sie zu steuern haben, angezeigt wird, oder dass die Automatik zuweilen an der Steuerung beteiligt oder diese vollständig übernimmt, um das zu steuernde Objekt in einem optimalen Kurs zur Fahrbahn der Tracking-Strecke zurückzuführen. Ziel dieser Arbeit ist es, Funktionen zu erstellen, mit denen dieser optimale Kurs und damit die jeweils optimale Richtung bestimmt werden kann.

1.3 Gliederung dieser Arbeit

In Kapitel 2 werden die benutzten Werkzeuge vorgestellt sowie auf für diese Arbeit wichtige Details von SAM eingegangen, die noch nicht erwähnt wurden. In Kapitel 3 wird die theoretische Grundlage gelegt, auf die der in dieser Arbeit entwickelte Algorithmus aufbaut, in Kapitel 4 wird eine wichtige praktische Vorarbeit, die Implementation der *racing line*, beschrieben. Kapitel 5 beschreibt den Basisalgorithmus, und Kapitel 6 und 7 zwei Spezialfälle, die vom Basisalgorithmus nicht abgedeckt werden. In Kapitel 8 wird schließlich Bilanz gezogen und insbesondere erläutert, warum die Arbeit an diesem Algorithmus zugunsten der Entwicklung eines neuen Algorithmus eingestellt wurde, der Thema der anschließenden Diplomarbeit sein wird.

2 Systemvoraussetzungen

ATEO ist auf einem *Squeak*-System implementiert,[6] *Squeak* ist ein *Smalltalk*-System. Außerdem wurden verschiedene Fragen, die im Rahmen dieser Arbeit auftraten, mit Hilfe von *[qt]octave* untersucht.

2.1 Smalltalk und Squeak

Smalltalk ist eine Programmiersprache bzw. Entwicklungsumgebung, die in den 70-er Jahren am Xerox PARC Forschungszentrum von einem Team unter der Leitung von Alan Kay entwickelt wurde. Der Name bezieht sich auf das grundlegende Konzepte, dass Objekte Nachrichten austauschen. Smalltalk wurde mit dem Ziel entwickelt, ein Computersystem zu schaffen, mit dem auch Laien und insbesondere Kinder umgehen können, einschließlich der Entwicklung von Programmen [21,19]. In das Konzept flossen Ideen von zahlreichen Vorbildern ein, meist werden die objektorientierten Konzepte der der Programmiersprache SIMULA sowie Sketchpad für die graphische Benutzeroberfläche als wichtigste Vorbilder genannt.[7]

Die Programmiersprache Smalltalk wurde mit *Smalltalk-80* im Jahr 1981 standardisiert.[8] In ihr ist das Konzept der Objektorientiertheit konsequent durchgeführt: *alles* ist

6 Es existiert auch eine Java-Version von SAM genannt SAMj.

7 In [19] werden noch zahlreiche andere Impulse erwähnt, die laut Kay zur Vorgeschichte von Smalltalk gehören.

8 Seitdem haben sich verschiedene Dialekte entwickelt, die sich aber meist nur minimal voneinander unterscheiden.

ein Objekt, auch jede Klasse ist eine Instanz einer Metaklasse und ihre Methoden Instanzen einer entsprechenden Klasse[9]. Variablen sind nicht typisiert,[10] sondern können Instanzen jeder Klasse beinhalten. Auch Operatoren wie + sind Methoden, was zur Folge hat, dass im Ausdruck `3+4*5` die Nachricht `*5` an das durch `3+4` erzeugte Objekt geschickt wird, da + und * zwei Methodennamen sind, die syntaktisch gleichwertig behandelt werden.

Smalltalkprogramme werden in Bytecode kompiliert, der durch eine virtuelle Maschine übersetzt wird.

Squeak ist eine Smalltalk-Implementierung, die ursprünglich von einer Arbeitsgruppe bei Apple entwickelt wurde, aber seit 2009 ein Open-Source-Projekt ist. Die Arbeitsgruppe überschnitt sich personell mit dem Team, das Smalltalk entwickelte hatte [14], und auch hier war eins der Hauptziele, dass Kinder damit einen spielerischen Zugang zum programmieren erhalten. *Squeak* ist deshalb mit Projekten wie *E-Toys* und dem *100-Dollar-Laptop-Programm* verbunden [23]. Andererseits ist es dank seiner Bibliotheken für Multimedia-Formate und verschiedenen Netzwerkprotokollen für Untersuchungen zur Mensch-Maschine-Interaktion interessant und auch als Simulationswerkzeug beliebt. [18].

Ein *Squeak-System* besteht aus der *virtuellen Maschine* (VM), einem betriebssystemspezifischen Programm, den Smalltalk-*Quellen* von *Squeak*[11] und dem Abbild, das wiederum aus einer *image*-Datei mit dem Zustand sämtlicher Objekte und einer *change*-Datei mit allen Änderungen am Quellcode besteht. Der Quellcode eines Elements (z.B. einer Klasse) bzw. Änderungen daran kann als Smalltalk-Datei (Erweiterung .st) oder „Changeset“-Datei (Erweiterung .cs) gespeichert bzw. eingelesen werden. Das ATEO-Projekt benutzt ein standardisiertes Abbild, in das die jeweils aktuelle Version des Quellcodes eingelesen wird, sowie die VM-Version 3.9, da das Abbild mit neueren Versionen nicht voll kompatibel ist [11].

2.2 octave und qtoctave

Zur Analyse und Darstellung verschiedener Aspekte der Aufgabenstellung wurde auf *octave* bzw. *qtoctave* zurückgegriffen, unter anderem auch deswegen, weil diese Programme dem Autor schon vertraut und zugänglich waren.

Die freie Software *GNU octave* ist ein Programm zur Berechnung mathematischer Aufgaben, das ursprünglich (1988) als Begleitsoftware für ein Chemielehrbuch gedacht war, bald aber zu einem flexibleren Programm weiterentwickelt wurde. In einer Kommandozeile können zu berechnende Ausdrücke in einer Skriptsprache eingegeben werden, die weitgehend zum proprietären Programm *Matlab* kompatibel ist.[12] Dabei kann nicht nur auf Elemente der Skriptsprache zurückgegriffen werden, sondern auch auf Dateien, die auch vom Benutzer geschrieben sein können. Solche Dateien können Kommandos und Funktionsdefinitionen enthalten, die ggf. gleichnamige, vorher definierte Funktionen überschreiben. Außerdem gibt es die Möglichkeit, berechnete Werte in Dateien zu speichern und aus diesen Dateien in *octave* einzulesen.[13]

Die Namensräume der einzelnen Funktionen sind strikt voneinander getrennt, auch die außerhalb einer Funktion existierende Variablen sind innerhalb einer Funktion nicht sicht-

9 In *Squeak* heißt diese Klasse `CompiledMethod`.

10 Hier von „dynamisch typisiert“ zu sprechen entspricht nicht dem Konzept von Smalltalk: zur Laufzeit findet keinerlei Typüberprüfung statt. Wenn eine Nachricht gesandt wird, wird die aufzurufende Methode im Pool der Klasse gesucht, auf die der interne Klassenpointer des Empfängers zeigt (und ggf. rekursiv in den Methodenpools ihrer Oberklassen), d.h. es ist die in der jeweiligen Variablen (bzw. ggf. auf dem Prozedurstack) gespeicherte Instanz, die darüber entscheidet, wie auf eine Nachricht reagiert wird und ob eine „Typverletzung“ (unbekannte Methode) vorliegt.

11 Dieser Teil ist insofern optional, als *Squeak* auch ohne den Sourcecode funktioniert, aber zur Programmentwicklung unerlässlich, da sonst Namen von Variablen o.ä. nicht angezeigt werden können.

12 Die Entwicklung von *octave* zu einem vollwertigen Matlab-Klone ist ein „high priority“-Projekt der *Free Software Foundation* [17].

13 Hier können alle in *Matlab* bekannten Dateitypen verwendet werden.

bar. Allerdings kann mit dem Schlüsselwort `global` eine Variable als „global“ deklariert werden, d.h. in jeder Funktion, in der diese Deklaration steht, wird damit auf die gleiche Variable referenziert.[14] Dies ist die einzige Möglichkeit, eine Variable zu deklarieren; grundsätzlich wird eine Variable zur Laufzeit angelegt, wenn in sie geschrieben wird. Das bedeutet auch, dass Variablen dynamisch typisiert werden.

Neben der Matrizenmultiplikation[15] bietet *octave* auch die Möglichkeit, Matrizen gleicher Dimension elementweise zu multiplizieren. Damit ist es möglich, einer entsprechend programmierten Funktion eine Reihe von Werten in einem Parameter zu übergeben und so eine Reihe gleichartiger Berechnungen mit einem Befehl durchführen zu lassen.

Für die Ausgabe von Grafiken wird das Programm *Gnuplot*[16] benutzt,[17] das von *octave* mittels einer *Pipe* angesprochen wird. Auch existieren diverse grafische Front-Ends für *octave*, benutzt wird *qtoctave*, das wie *octave* als Linux- und Windowsversion existiert. In der Version 4.0 soll *octave* mit einer eigenen IDE ausgestattet sein, weshalb *qtoctave* inzwischen nicht mehr weiterentwickelt wird.

Im Verlauf der Arbeit wurden beide Programme mehrfach aktualisiert, die benutzen Versionen reichen von *octave* 3.0.2 bis 3.6.4 bzw. bis *qtoctave* 0.8.2 bis 0.91.

2.3 Weitere Details zu SAM

Wie schon in der Einleitung (Abschnitt 1.1.1) gesagt, steuern zwei *Mikroweltenbewohner* (MWB) ein *tracking object* (TO) mit Hilfe von Joysticks entlang einer Strecke. Die horizontale und die vertikale Bewegung des TOs werden dabei auf verschiedene Weise dargestellt: bei einer horizontalen Bewegung wird das TO auf dem Bildschirm entsprechend horizontal bewegt, bei einer vertikalen Bewegung die Grafik der Strecke nach unten verschoben, so dass sich der visuelle Eindruck einer Fahrt nach oben ergibt, obwohl des TO eine feste vertikale Position auf dem Bildschirm hat.

Die in SAM benutzte Längeneinheit ist damit das `Px`, der Abstand von zwei Pixeln auf dem Bildschirm.[18] Zwar können die Grafiken des TOs und der Strecke nur an ganzzahligen Pixelkoordinaten angezeigt werden, aber intern wird auch mit Bruchteilen von Pixeln gerechnet, so dass sich das TO auch an Orten mit nicht ganzzahligen Koordinaten befinden kann, die von *Squeak* bei der Anzeige auf ganzzahlige Werte gerundet werden.

Im Normalfall werden die Joystickeingaben als gleichwertig behandelt, was vom Operateur (Bediener des OA) oder einer Automatik geändert werden kann. Dieser Aspekt ist allerdings für die vorliegende Arbeit irrelevant, da ein optimaler Kurs des TO für beide MWB gleich ist, und bei Bezug auf die Lenkbewegungen der beiden MWB niemals zwischen ihnen unterschieden werden braucht. Im weiteren Verlauf der Arbeit wird also stets von der Lenkbewegung der MWB gesprochen, ohne darauf einzugehen, wie diese gemeinsame Lenkbewegung zustande kommt.

Die Joystick-Eingaben der MWB werden in regelmäßigen Abständen von 39 `ms` gemessen, so dass die Bewegung des TOs innerhalb einer Zeitscheibe (im Folgenden Tick genannt) atomar ist. Diese Bewegung wird sowohl horizontal wie vertikal in Längeneinheiten 1 `Px` aus der jeweils aktuellen Joystick-Position berechnet. Damit ist ein Tick als Zeiteinheit für die Länge einer Fahrt geeignet. Zwar können Rundungsfehler, Verzögerungen durch das Betriebssystem oder Aktivitäten des *garbage collectors* in *Squeak* zu einer längeren Zeitspanne als 39 `ms` führen,[19] jedoch sind diese Abweichungen erstens

14 Diese Deklaration kann natürlich auch außerhalb einer Funktion stehen, und damit den Zugriff auf diese Variable z.B. von der Kommandozeile erlauben. Dabei spielt es keinen Rolle, ob diese Deklaration von der Kommandozeile oder von einer Skriptdatei eingelesen wurde.

15 Letztlich ist in *octave* auch ein Vektor eine einzeilige bzw. einspaltige Matrix und ein Skalar eine 1×1-Matrix.

16 Siehe http://www.gnuplot.info/.

17 Optional kann *octave* auch so konfiguriert werden, dass ein anderes Programm benutzt wird.

18 Nicht zu verwechseln mit dem Pixel als Flächeneinheit der Größe 1 $\mathtt{Px}^2$!

19 Theoretisch können auch längere Berechnungen auf einem langsamen System die Ursache sein, allerdings wurde bei der Entwicklung des Systems bzw. bei der Beschaffung der Hardware darauf geachtet, dass das

meist minimal[20] und zweitens von der Software nicht vorhersehbar, so dass für die Zwecke dieser Arbeit die Zeit grundsätzlich in Ticks (T) gemessen wird. Entsprechend wird auch die horizontale bzw. vertikale Geschwindigkeit des TOs in Px/T angegeben.

Ein Joystick liefert stets zwei ganzzahlige Werte im Bereich von -1023 bis 1024, einen für die Auslenkung zum Benutzer hin (positive Werte) bzw. vom Benutzer weg (negative Werte)[21] und einen für die Auslenkung nach rechts und links (negative Werte links). Diese Joystick-Koordinaten müssen noch in Pixelkoordinaten umgerechnet werden, Wenn M_x und M_y die aus den Joystickwerten ermittelte „gemeinsame Joystickauslenkung“ ist, so berechnen sich die Bewegungswerte v_x und v_y als:

$$v_x = \frac{M_x}{75}\,\mathrm{Px}; \quad v_y = \frac{1024 - M_y}{100}\,\mathrm{Px}.^{22} \qquad \textit{(Formel 1)}$$

Die Formel für die vertikale Geschwindigkeit besagt insbesondere, dass , dass das TO nicht nach unten fahren kann, sondern nur oben.

Von diesen Formeln gibt es Ausnahmen, die folgendermaßen implementiert sind: In der Methode `processInputData` in `SAMControllerInput` wird vor der Umrechnung getrennt für jeden der beiden MWI jeder Joystickwert, dessen Betrag kleiner als 30 ist, auf Null gesetzt [12, Kap. 3.3], und nach der Umrechnung wird jeder y-Wert, der kleiner als 0,025 Px ist, ebenfalls auf Null gesetzt [13, Kap. 4.5.2]. Mit diesen *dead bands* wird einerseits erleichtert, das TO mit dem Joystick in Ruhestellung zu steuern, andererseits wird so ermöglicht, das TO nicht bzw. exakt horizontal zu bewegen. Denn gemäß Formel 1 würde das TO sich um 0,01 Px vertikal bewegen, wenn der Joystick vollständig an den Körper herangezogen wird.[23]

Aus Formel 1 ergibt sich $13{,}65\overline{3}$ Px/T[24] als Wert für die maximale horizontale Geschwindigkeit v_{xmax} sowie 20,48 Px/T für die maximale vertikale Geschwindigkeit v_{ymax}. Da je nach Stellung des Joysticks jeder Wert für horizontale Auslenkung mit jedem Wert für vertikale Auslenkung kombiniert sein kann, sind horizontale und vertikale Bewegungen unabhängig voneinander. Definiert man den Abstand von zwei Punkten über die minimale Zeit, die benötigt wird, um das TO von einem Punkt zum anderen zu bewegen. In Abbildung 2 ist gestrichelt die Linie mit dem „zeitlichen Abstand“ von 1 T sowie durchgezogen mit 2 T um einen Ausgangspunkt eingezeichnet. Außerdem sind zwei mögliche Bewegungen mit jeweils maximaler horizontaler bzw. vertikaler Geschwindigkeit ab diesem Punkt eingezeichnet. Wie leicht zu sehen ist, ist es nur in zwei *diagonalen* Richtungen (nach oben/ links bzw. oben/rechts) möglich, sowohl horizontal wie auch vertikal mit maximaler Geschwindigkeit zu fahren. Damit ähnelt das so defi-

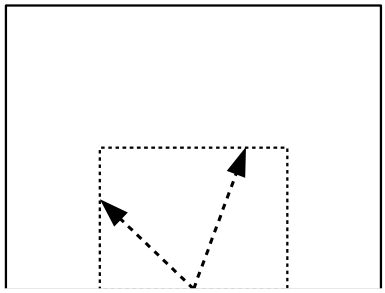

Abbildung 2: Linien gleichen Abstands und zwei Bewegungen mit maximaler Geschwindigkeit.

in den Versuchen nicht der Fall ist.

20 In den Logdateien sind ab und zu Zeitspannen von 40 oder 41 ms zu sehen. Sehr selten kamen längere (z.T. deutlich längere) Zeiten vor.

21 In den Logdateien werden diese Werte (Felder `UpDownMWI1` und `UpDownMWI2`) negiert, so dass eine Auslenkung des Joysticks vom Benutzer weg als positiver Wert notiert wird.

22 Die originalen Formeln (mit der Berücksichtigung der Gewichtung der MWB) sind z.B. in [13] zu finden.

23 Im dazugehörenden Kommentar in `SAMControllerInput»processInputData` ist von den Werten 0,02 bzw. 0,024 die Rede, was von [13] als Hinweis auf Defizite bei Joysticks gedeutet wird. Allerdings ist der Wert 0,024 so nicht vorstellbar, da Joystickwerte grundsätzlich ganzzahlig sind. Möglicherweise traten diese erwähnten Werte in einer frühen Version von SAM mit anderer Umrechnung als nach Formel 1 auf.

24 Streng genommen gilt dieser Wert nur für Bewegungen nach links, nach rechts kann nur mit 13,64 Px/T gefahren werden. Dieser Unterschied kann aber vernachlässigt werden, zumal eine Automatik problemlos angeben kann, dass sich das TO mit v_{xmax} nach rechts bewegt.

nierte Abstandsmaß einer Maximums-Metrik[25], und es kann zwischen vorwiegend vertikalen und vorwiegend horizontalen Abständen unterschieden werden,[26] bzw. zwischen vorwiegend horizontalen und vorwiegend vertikalen Bewegungen des TOs.

2.4 Weitere Details zur Strecke

Zunächst müssen hier einige Begriffe auseinandergehalten werden, die leicht verwechselt werden können: das Wort *Strecke* bezeichnet in SAM die am Bildschirm angezeigte Grafik mit grünem Hintergrund, auf der die graue *Fahrbahn* eingezeichnet ist. Relevant ist aber eigentlich nur die gestrichelte weiße Mittellinie bzw. eine daraus berechnete Linie, die in dieser Arbeit als *Racing Line* (RL) bezeichnet wird.[27]

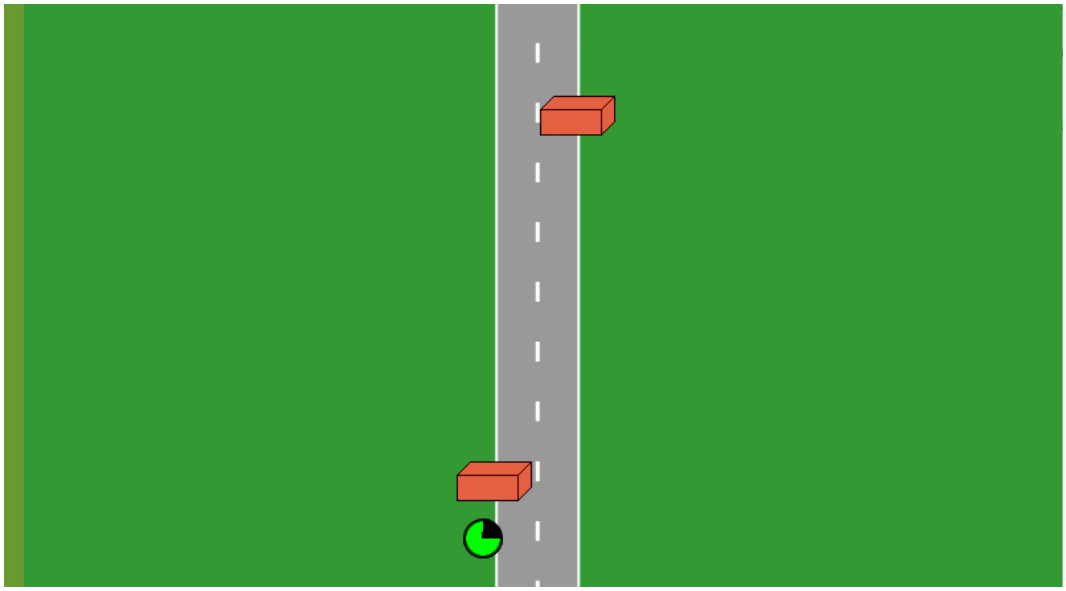

Abbildung 3: Das TO vor einem statischen Hindernis (Ausschnitt über die volle Streckenbreite).

Wie schon oben erwähnt,[28] gibt es auf der Strecke, noch zwei Arten „Schikanen“, die vor allem dazu dienen, den Konflikt zwischen den unterschiedlichen Zielvorgaben an beide MWB zu verstärken. Dies sind einerseits Hindernisse, von denen es wiederum zwei Arten gibt. Abbildung 3 zeigt zwei *statische* Hindernisse, die wie stets paarweise am linken und rechten Fahrbahnrand auftreten, daneben gibt es auch *dynamische* Hindernisse, die auf einem Standbild kaum von einem statischen Hindernis unterschieden werden können,[29] aber sich horizontal von links nach rechts bewegen. Bei einer Kollision mit einem Hindernis wird das TO nach einer „Strafzeit“ von einer Sekunde links neben die Fahrbahn gesetzt, was die Dauer der Fahrt erhöht und einen Fahrfehler (Abweichen von der RL) produziert.

Die zweite Art von „Schikane“ besteht in Gabelungen, an denen sich die Strecke teilt und die MWB sich für den rechten bzw. linken Zweig entscheiden müssen. Heute werden nur noch die beiden in Abbildung 4 (nächste Seite) gezeigten Gabelungen benutzt; andere Gabelungen gelten als obsolet. Die Darstellung mit unterschiedlicher Fahrbahnbreite soll bei den MWB den Eindruck erwecken, dass ein Zweig für eine genaue, der andere für eine schnelle Fahrt besser geeignet ist.

Die in den Versuchen benutzten Strecken sind nach dem Baukastenprinzip aufgebaut. Jede Strecke besteht aus einzelnen *Kacheln* (*tiles*), die unterschiedlich lang sind, aber alle

25 Unterschiede zur Maximum-Metrik ergeben sich aus zwei Eigenschaften des TOs (1) seine maximale horizontale und maximale vertikale Bewegung pro Tick sind verschieden (2) es kann nicht nach unten fahren.

26 Diese für ganze Ticks angewandte Überlegung lässt sich theoretisch auch für Bruchteile von Ticks und damit auf alle Abstände erweitern.

27 Vgl. dazu auch Abschnitt 3.1 und Fußnote 35.

28 Abschnitt 1.1.1, S. 2.

29 Grundsätzlich ist dies allerdings möglich, da sie nur einzelnen auftreten und meist nicht an einer Position sind, an der statische Hindernisse platziert werden.

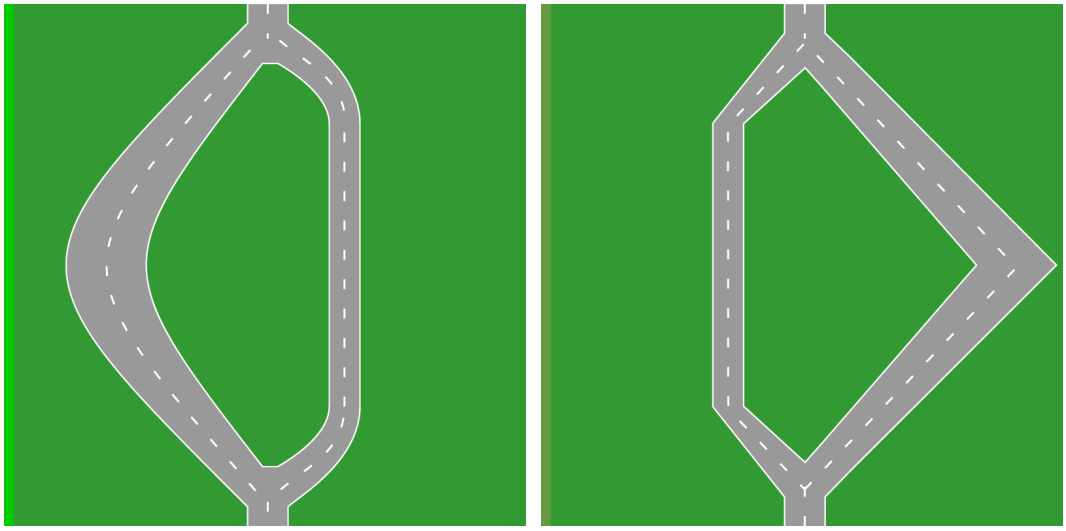

Abbildung 4: runde Gabelung RLl und eckige Gabelung Elr (nicht maßstäblich zu Abb. 1 und 3).

die Strecke den oberen und unteren Rand an der gleichen Stelle erreichen lassen, so dass diese Kacheln in beliebiger Reihenfolge zu einer Strecke kombiniert werden können. In einer früheren Version von SAM geschah die Anzeige der Strecke so, dass ein oder zwei Kacheln an der entsprechenden Stelle des Bildschirms angezeigt wurden, aus Performanzgründen wird inzwischen aber stets eine einzige aus den Kachelbildern erstellte Grafik benutzt, um die jeweilige Strecke anzuzeigen.

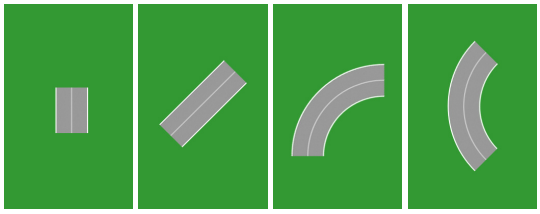

Abbildung 5 : Beispiele für Streckenelemente.

Von zwei Ausnahmen abgesehen, sind alle Kacheln aus kleineren Streckenelementen zusammengesetzt. Diese sind entweder geradlinig, (vertikal oder diagonal im Winkel von 45°) oder Viertelkreise, in denen die RL von vertikal zu horizontal bzw. umgekehrt oder an den Enden diagonal im Winkel von 45° verläuft, wobei die Treppenfunktion der RL nicht in jedem Fall die beste Annäherung an diese Idealisierungen darstellt.

Bei den oben erwähnten Ausnahmen handelt es sich um die Kacheln mit den Gabelungen. Bei der eckigen Gabelung sind beide Zweige aus geradlinigen Abschnitten zusammengesetzt, bei denen die diagonalen Abschnitte aber nicht exakt diagonal bei 45° verlaufen, und bei der runden Gabelung liegt ein Verlauf vor, dessen mathematische Beschreibung nicht bekannt ist.[30]

3 Theorie

In diesem Kapitel werden, ausgehend von einer Diskussion des Begriffs „optimal“, die für das nähere Verständnis des Problems und die Erstellung eines Algorithmus grundlegenden Konzepte erarbeitet.

3.1 Grundbegriffe

Für eine optimale Rückführung ist es unerlässlich, dass „optimal“ exakt definiert ist. Dies war zu Beginn der Aufgabenstellung noch nicht der Fall: zwar sind ein Zeit- und Fahrfehler schon definiert, aber eine Definition, welche Kombination von Zeit- und Fahrfehler optimal ist, war bislang noch nicht erarbeitet worden.[31]

30 Vermutlich ist es eine *spline*-Funktion eines Grafikprogramms.

31 Das impliziert auch, dass den Entwicklerteams keine eindeutige Definition von „optimal“ vorgelegen hat. Wenn ein Team eine eigene Definition erarbeitet hat, die von der in diesem Kapitel erarbeiteten erkennbar abweicht, sollte natürlich diese Definition bei der Realisierung der entsprechenden Automatik benutzt werden. In den Konzept 30 und 41 geht es lediglich darum, dass die vertikale Geschwindigkeit zeitweise herabgesetzt werden muss, wenn die Strecke nicht verlassen werden soll.

Zunächst aber sollten die wichtigsten Grundbegriffe, die bei den dazu nötigen Überlegungen eine Rolle spielen, kurz erläutert werden.

Wie schon gesagt[32], steuern in einem Versuch die MWB ein Objekt, das *tracking object* (TO), mit Hilfe von Joysticks über verschiedene Strecken, jede Fahrt wird auch als Schritt (*step*) bezeichnet. Die relevanten Maßeinheiten für Bewegung und Zeit sind Pixelgröße `Px` und Tick `T` und entsprechend für die Geschwindigkeit `Px/T`.

Der *Zeitfehler* ist definiert als die „zusätzliche“ Zeit, welche die MWB in einer Fahrt über die minimale Zeit hinaus gebraucht haben, die für die Fahrt des TOs über die Strecke benötigt wird.[33] Diese ergibt sich aus der Länge der Strecke[34], geteilt durch die maximal mögliche vertikale Geschwindigkeit v_{ymax} von 20,48 `Px/T`.

Der *Fahrfehler* ist definiert als die Fläche zwischen der Fahrt des TOs und der Mittellinie der Fahrbahn auf der Strecke (RL),[35] es handelt sich also um einen *Flächenfehler*. Die entsprechende Maßeinheit ist damit das Quadrat der Länge eines Pixels (`Px²`).

Diese Fehlerdefinitionen haben aus Sicht dieser Arbeit den „Nachteil“, dass sie über eine ganze Fahrt definiert sind. Für die Entscheidung, welche Joystick-Position in einem Tick optimal ist, müssen aus einer Joystickposition bzw. der dadurch erzeugten Bewegung des TOs auch der im entsprechenden Tick verursachte *lokalen* Zeit- und Flächenfehler berechnet werden, weshalb diese Definitionen lokal herunterzubrechen sind.[36]

Der *lokale Fahrfehler* ist schon als der Flächenfehler definiert, der in einem Tick anfällt. Da das TO nicht rückwärts[37] fahren kann, gibt es auch keine Möglichkeit, einen einmal aufgetretenen Flächenfehler zu korrigieren.

Für die Definition des *lokalen Zeitfehlers* wird die Bewegung des TOs innerhalb eines Ticks als kontinuierlich angenommen. Damit kann für eine beliebige vertikale Distanz *d* die Zeit bestimmt werden, die nötig ist, um d mit maximaler vertikaler Geschwindigkeit zu durchlaufen, sie beträgt d/v_{ymax}. Damit beträgt der Zeitfehler in einem Tick, in dem sich das TO vertikal um *d* bewegt

$$1\,\mathrm{T} - \frac{d}{v_{ymax}} = \frac{v_{ymax} * 1\,\mathrm{T} - d}{v_{ymax}}.$$

3.2 Beziehung zwischen Zeit- und Flächenfehler

3.2.1 Notwendigkeit einer Beziehung

Bei der initialen Formulierung der Aufgabenstellung gab es die Vorgabe, dass Zeitfehler und Flächenfehler möglichst unabhängig voneinander optimiert werden sollten, und auf jeden Fall der zu berechnende optimale Kurs in beiden Fehlerkategorien besser sein soll-te als der von dem MWB gesteuerte Kurs. Aber wenn diese Vorgabe eingehalten würde, könnten die Ergebnisse in manchen Fällen kaum als optimal bezeichnet werden. Dazu ein Beispiel:

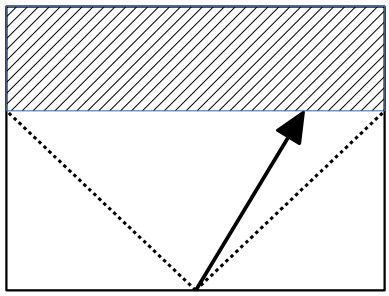

Abbildung 6: Joystickposition und mögliche Bewegungsrichtungen

Die nebenstehende Abbildung 6 bezeichne die Menge der möglichen Bewegungen des TOs in einem Tick. Der

32 Abschnitt 1.1.1, S.1.

33 Persönliche Mitteilung von Herrn Nicolas Niestroj.

34 Genauer: der Distanz, die auf dieser Strecke zurückzulegen ist.

35 Dies war im Projekt nicht allgemein bekannt (es herrschte der Eindruck vor, dass eine Fahrt, welche die Fahrbahn nicht verlässt, keinen Fahrfehler verursacht) und wurde deshalb auch nicht an die Entwicklerteams weitergegeben. Auch weitere Details (siehe Kapitel 8.3, S.33) konnten nur durch Inspektion des Java-Quellcodes des LFA-Tools gefunden werden.

36 Dies schließt natürlich nicht aus, dass bei der Bewertung einer Joystickposition auch die in nachfolgenden Ticks entstehenden „Folgefehler“ mit berücksichtigt werden.

37 Also nach unten!

Pfeil zeigt vom „Nullpunkt“ (keine Bewegung)[38] auf die Bewegung, die der aktuellen Joystickposition entspricht. Wenn nun in der optimalen Position der (lokale) Zeitfehler nicht erhöht werden darf, kann die vertikale Geschwindigkeit nicht erniedrigt werden. Die dann möglichen Bewegungen liegen alle in dem schraffierten Bereich, und wie leicht zu erkennen ist, liegen damit auch die Richtungen, in die sich das TO bewegen darf, alle im Winkel, der von den gestrichelten Linien gebildet wird.

Abbildung 7 zeigt das TO und die an dieser Stelle beste „erlaubte“ Richtung, falls die Joystickposition die gleiche ist wie laut Abbildung 6. Wenn ein entsprechender „optimaler“ Kurs von einer Automatik empfohlen wird und sich die MWB an diese Empfehlung halten, liegt im nächsten Tick im Prinzip die gleiche Situation vor. Damit fährt das TO auf der eingezeichneten gestrichelten Linie zur Fahrbahn, was einen ziemlich großen Flächenfehler erzeugt. Aber sobald die MWB sich nicht an diese Empfehlung halten und die vertikale Geschwindigkeit verringern, kann auch eine Empfehlung gegeben werden, die der Entscheidung der MWB folgt und so den Flächenfehler deutlich verringert! Damit können die so entstandenen Empfehlungen kaum als *optimal* bezeichnet werden.

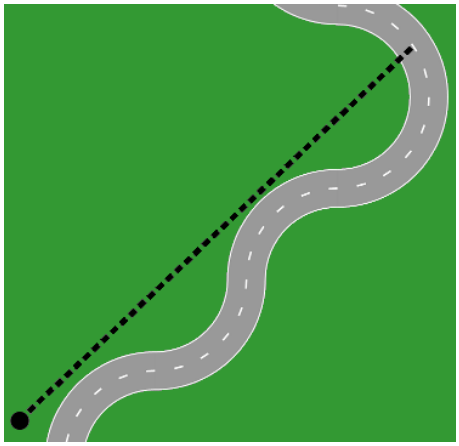

Abbildung 7: Schlechter „optimaler“ Weg

Damit führt kein Weg daran vorbei, für einen optimalen Kurs Zeit- und Flächenfehler gegeneinander abzuwägen. Die einfachste Möglichkeit besteht darin, beide zu addieren und nach dem Kurs zu suchen, bei dem die Summe minimal wird. Das setzt aber voraus, dass die unterschiedlichen Maßeinheiten (T und Px²) ineinander umgerechnet werden können. Als quasi natürliches Maß wurde bestimmt, dass der maximale globale Zeitfehler gleich dem maximalen globalen Flächenfehler ist.

3.2.2 „maximale Fehler“

Streng genommen kann es keine maximalen Zeitfehler geben, da das TO auch stehen bleiben bzw. sich horizontal bewegen kann (vertikale Geschwindigkeit beträgt 0 Px), und somit beliebig lange Fahrten denkbar sind. Als maximaler Zeitfehler wurde daher der Fehler bestimmt, der auftritt, wenn das TO mit 1 Px/T fährt.[39] Während bei dieser „Minimalgeschwindigkeit“ das TO einen Tick pro Pixel benötigt, sind es bei schnellstmöglicher Fahrt nur 1 Px/v_{ymax} Ticks pro Pixel. Der Zeitfehler F_T beträgt damit für eine Fahrt über die Distanz von einem Pixel

$$1\,\mathrm{T} - \frac{1\,\mathrm{Px}}{v_{ymax}} = \frac{v_{ymax} * 1\,\mathrm{T} - 1\,\mathrm{Px}}{v_{ymax}},$$

und damit ist der Zeitfehler über einen Distanz d

$$\begin{aligned} F_T &= \frac{d}{1\,\mathrm{Px}} * \frac{v_{ymax} * 1\,\mathrm{T} - 1\,\mathrm{PX}}{v_{ymax}} \\ &= d * \frac{v_{ymax} * 1\,\mathrm{T/Px} - 1}{v_{ymax}} \\ &= d * 0{,}951171875\ \mathrm{T/Px}. \end{aligned}$$

Der „maximale“ Zeitfehler ist damit proportional zur Länge der zu fahrenden Strecke.

38 Dieser sollte nicht mit der Ruhestellung des Joysticks verwechselt werden, die einer vertikale Bewegung mit ½*v_{xmax} entspricht und sich natürlich in der Mitte der Fläche befindet.

39 Dies ist anscheinend noch in keiner Publikation erwähnt worden (persönliche Mitteilung von Herrn Nicolas Niestroj).

Der maximale Flächenfehler kann bestimmt werden als der Flächenfehler einer Fahrt, in der das TO stets einen maximalen Abstand zur RL hat, oder sich horizontal bewegt. Die horizontalen Bewegungen sind notwendig, da der Punkt mit dem maximalen Abstand sich je nach Streckenverlauf am rechten oder linken Rand oder auch in der Mitte einer Gabelung befinden kann, und deshalb von Zeit zu Zeit zwischen diesen Positionen gewechselt werden muss. Damit ist der maximale Flächenfehler nicht proportional zur Länge der Strecke, und damit das Verhältnis zwischen beiden von der zu fahrenden Strecke abhängig.

Um einen Umrechnungsfaktor herzuleiten, wird zunächst angenommen, dass die durchschnittliche Breite *b* der Fehlerfläche bekannt ist, also der maximale Flächenfehler für eine Strecke der Länge *L* sich als $F_A = b * L$ darstellen lässt. Der Zeitfehler ist dann

$$F_T = L * * \frac{v_{ymax} * 1\,\mathrm{T/Px} - 1}{v_{ymax}}.$$

Der Umrechnungsfaktor vom Zeitfehler zum Flächenfehler ergibt sich somit als

$$\frac{F_A}{F_T} = \frac{b * L}{L * \frac{v_{ymax} * 1\,\mathrm{T/Px} - 1}{v_{ymax}}} = \frac{b * v_{ymax}}{v_{ymax} * 1\,\mathrm{T/Px} - 1}. \qquad \textit{(Formel 2)}$$

Nun müssen nur noch die b-Werte für die einzelnen Strecken bestimmt werden. Hierzu wurden zunächst die einzelnen Kacheln, aus denen die Strecken zusammengesetzt sind, untersucht. Für jede Kachel wurden die Werte aus der entsprechenden CSV-Datei[40] in eine Tabellenkalkulation eingelesen und dann für jeden Eintrag die Abstände zum rechten und linken Rand sowie ggf. der Abstand zwischen den beiden Zweigen einer Gabelung berechnet und das Maximum dieser drei Werte ermittelt. Der Flächenfehler für eine Kachel ergab sich somit als Summe dieser Maxima, multipliziert mit dem Abstand der Punkte von jeweils 1 Px. Damit konnte nun für jede Strecke der maximale Flächenfehler als Summe der Flächenfehler der entsprechenden Kacheln bestimmt werden. Zur Bestimmung des b-Werts für eine Strecke musste dann nur noch dieser Flächenfehler durch die Länge der Strecke dividiert werden. Die so gefunden Werte sind in der Tabelle 1 aufgeführt.

Strecke	**B-Wert (in Px)**
Kacheldurchschnitt	*480,248826*
hauptabschnitt_lang	482,028590
hauptabschnitt1	481,317696
hauptabschnitt2	481,317696
hauptabschnitt3	481,663794
hauptabschnitt4	482,009892
hauptabschnitt5	480,971598
hauptabschnitt6	481,663794
Durchschnitt über benutzte Strecken	481,898699

Tabelle 1: b-Werte der einzelnen Strecken

Neben den Werten für die einzelnen Strecken[41] ist auch noch der Durchschnitt über alle Kacheln angegeben, sowie der Durchschnitt über die Strecken, die tatsächlich in der Versuchsanordnung benutzt werden, also in der Konfigurationsdatei *steps.txt* eingetragen sind.[42] Die so gefundenen Werte weichen nicht wesentlich voneinander ab, weshalb es nicht sinnvoll ist, für jede Strecke einen eigenen Wert anzusetzen, der dem exakten maximalen Flächenfehler entspricht. Stattdessen wird für alle Strecken der gleiche Wert benutzt, und zwar 481,9352 Px. Die Motivation für diesen Wert besteht nicht nur darin, dass er nahe an dem gewichteten Durchschnitt aller benutzten Strecken liegt, er ist auch deshalb gut geeignet, weil so die im nächsten Abschnitt vorgestellten Konstanten verhältnismäßig runde Werte erhalten.

40 Siehe Kapitel 4.2, S. 14.

41 Nicht berücksichtigt wurden Strecken, in denen keine Gabelungen vorkommen, da für sie nie vorgesehen war, Automatiken einzusetzen (persönliche Mitteilung von Herrn Nicolas Niestroj).

42 Die Testfahrten blieben dabei unberücksichtigt. Somit waren handelte es sich um: hauptabschnitt_ohne_manipulationen, hauptabschnitt1, hauptabschnitt2, hauptabschnitt_lang.

3.2.3 Beziehung von Wegstrecken zu Flächenmaß

Aus den bisherigen Überlegungen ergibt sich, dass sich Zeitfehler stets aus nicht gefahrenen Distanzen ergeben bzw. diesen proportional sind. Bisher waren dies stets vertikale Distanzen, was aus der Definition des lokalen Zeitfehlers folgt.

Nun kann es weit von der Strecke entfernt optimal sein, zunächst horizontal zur Fahrbahn zu fahren, und um den globalen Zeitfehler klein zu halten, sollte dies mit maximaler horizontaler Geschwindigkeit geschehen. Damit kann analog zu dem bisher besprochenen „vertikalen" Zeitfehler analog ein „horizontaler" Zeitfehler bestimmt werden, der auftritt, wenn in so einer Situation nicht mit maximaler horizontaler Geschwindigkeit gefahren wird. Da der „zeitliche Abstand" einer Maximum-Metrik ähnelt[43], hängen der horizontale und der vertikale Zeitfehler stets nur von der nicht gefahrenen Distanz (im Vergleich zur maximal möglichen Distanz innerhalb der entsprechenden Zeit) in der entsprechenden Dimension ab, die gleichzeitige Bewegung in der anderen Dimension spielt keinen Rolle. Und weil die maximale horizontale Geschwindigkeit nur ⅔ der maximalen vertikalen Geschwindigkeit beträgt, ist ein horizontaler Zeitfehler 1,5-mal so groß wie ein vertikaler Zeitfehler über die gleiche Distanz.

Für die Konzeption des Algorithmus spielt der oben genannte Fall, in dem ein horizontaler Zeitfehler auftreten kann, eine geringe Rolle. Wichtiger ist der Fall, in dem ein TO, dass sich auf der RL befindet, nicht vertikal mit v_{ymax} fahren kann, wenn es der RL folgen will, so dass eine Fahrt mit weniger als v_{xmax} einen lokalen horizontalen Zeitfehler erzeugt. Allgemein enthält eine Fahrt, die in einem Punkt P_1 auf der RL endet, einen lokalen Zeitfehler im Vergleich mit einer anderen Fahrt, die an einem Punkt P_2 endet, der später auf der RL liegt.[44] Wenn also die RL zwischen P_1 und P_2 *vorwiegend horizontal* verläuft, ist dies ein horizontaler Zeitfehler, verläuft sie *vorwiegend vertikal*, ist dieser lokale Zeitfehler auch vertikal.[45]

Bei der Abwägung eines Zeitfehlers gegen einen Flächenfehler wird also aus einer nicht gefahrenen Distanz ein Zeitfehler berechnet und dieser dann in einen gleich großen Flächenfehler umgerechnet. In der Praxis kann nun die Zwischenstufe „Zeitfehler in Ticks" übergangen und der Flächenfehler direkt aus der Distanz berechnet werden. Der Umrechnungsfaktor von einer Distanz (in `Px`) in eine Fläche (in `Px²`) ist natürlich wieder eine Distanz. Diese Vereinfachung wurde allerdings erst während der Konzeption des Algorithmus gefunden, in dem diese Distanzen eine wichtige Rolle als „Standardabstände" spielen. Dabei ist der *horizontale* Abstand δ_x der Umrechnungsfaktor für die Distanz eines *vertikalen* Zeitfehlers und der *vertikale* Abstand δ_y für die eines *horizontalen* Zeitfehler. Es muss also stets ein „x-Wert" mit einem „y-Wert" multipliziert werden, um eine Fläche zu erhalten.

Ausgehend von Formel (2) lässt sich δ_x herleiten, indem ein „vertikaler" Zeitfehler mit einem Rechteck verglichen wird, dessen Breite *d* die Distanz ist, die diesem Zeitfehler erzeugt:

$$\delta_x * d = \frac{b * v_{ymax}}{v_{ymax} * 1\,\mathtt{T/Px} - 1} * \frac{1}{v_{ymax}} * d$$

$$\Leftrightarrow \delta_x = \frac{b}{v_{ymax} * 1\,\mathtt{T/Px} - 1} \qquad \textit{(Formel 3)}$$

$$\Leftrightarrow \delta_x = \frac{481{,}9352\,\mathtt{Px}}{19{,}48} \quad \Leftrightarrow \delta_x = 24{,}74\,\mathtt{Px}\,.$$

43 Siehe Abschnitt 2.3, S.6 und die damit zusammenhängende Erläuterung der Begriffe „vorwiegend horizontal/vertikal".

44 Dazu kommt ggf. ein Unterschied in dem Zeitfehler, der sich aus der jeweiligen Länge der beiden Fahrten in Ticks ergibt.

45 Wenn die RL zwischen P_1 und P_2 teils vorwiegend horizontal, teils vorwiegend vertikal verläuft, reicht diese Überlegung natürlich nicht aus.

Die Herleitung von δ_y geht analog, nur dass der Zeitfehler für einen Tick maximal von einer Strecke der Länge v_{xmax} herrühren kann:

$$d * \delta_x = \frac{b * v_{ymax}}{v_{ymax} * 1\,\mathrm{T/Px} - 1} * \frac{1}{v_{xmax}} * d$$

$$\Leftrightarrow \delta_y = \frac{b * 3}{(v_{ymax} * 1\,\mathrm{T/Px} - 1) * 2} \qquad \textit{(Formel 4)}$$

$$\Leftrightarrow \delta_y = \frac{481{,}9352\,\mathrm{Px} * 3}{19{,}48 * 2}$$

$$\Leftrightarrow \delta_y = 37{,}11\,\mathrm{Px}\,.$$

Im weiteren Verlauf der Arbeit wird nicht mehr begrifflich zwischen einem Zeitfehler und der nicht gefahrenen Distanz, durch die er bewirkt wird, unterschieden.

Für spätere Überlegungen ist schließlich der Begriff des *relativen* Fehlers nützlich. Hiermit ist stets die Differenz zwischen zwei Fehlern gemeint. Im einfachsten Fall ist der relative Zeitfehler zwischen zwei Kursen des TOs, die jeweils in den Punkten P und Q auf der RL enden, der horizontale bzw. vertikale Abstand zwischen P und Q.

Mit den Begriffen des (vorwiegend) *vertikalen* bzw. *horizontalen Verlaufs der RL*, des vertikalen und horizontalen *Zeitfehlers* und den Werten δ_x und δ_y, die als Umrechnungsfaktoren von diesen Fehlertypen in eine Fläche dienen und so erst den Vergleich mit einem *Flächenfehler* ermöglichen, steht das theoretische Grundgerüst bereit, um den Algorithmus zu konzipieren. Zunächst wird aber über eine Arbeit gesprochen, die erst die Voraussetzung für die Implementation eines Algorithmus zur optimalen Rückführung auf die Strecke schafft: die Implementation der RL.

4 *Racing Line*: Integration in das System

Zu Beginn der Aufgabenstellung war die RL in SAM nur rudimentär vorhanden. Einerseits lag sie als gestrichelte Linie in den Grafiken der jeweils zu fahrenden Strecken vor, die die MWB zu sehen bekamen. Außerdem gab es in diesen Grafiken noch „Rotwerte“, die etwas über den Streckenverlauf aussagten. Eine Methode, die ein Algorithmus benutzen könnte, um den Streckenverlauf zu erfahren, existierte nicht.

4.1 Rotwerte

Abbildung 8: Drei Farbkodierungen

Der äußerste linke Rand der Strecke liegt stets außerhalb der Fahrbahn, also in einem Bereich, der normalerweise grün eingefärbt ist. Ein 16 Pixel breiter Streifen am linken Rand wird aber benutzt, um durch andere Farben Informationen über den Verlauf der Strecke zu kodieren. Wie der Name „Rotwert“ andeutet, unterscheiden sich die anderen Farbtöne meist nur durch den Rot-Wert der im RGB-Farbraum gemessenen Farbwerte, allerdings gibt es auch Fälle, in denen auch die anderen Farbwerte sich vom Grün neben der Fahrbahn unterscheiden.[46] In Abbildung 8 ist ein vergrößerter Ausschnitt aus einer Streckengrafik zu sehen, neben dem Hintergrund-Grün (RGB-Werte: 51, 153, 51) ist deutlich der Streifen links zu erkennen, in dem (von unten nach oben) die Farbkodierungen für die runde Gabel *RLl* (RGB-Werte: 1, 204, 0), für *speed measurement start* (RGB-Werte: 44, 153, 51) und für einen vertikal verlaufenden Abschnitt (RGB-Werte: 104, 153, 51) zu sehen sind. Grundsätzlich steht ein Rotwert für ein elementares Streckenteil.[47]

46 Mehr zu den Rotwerten und den „atomaren“ Elementen, die damit kodiert werden, ist in der Datei *BilddetailsATEO final bgr 16.07.09.xls* zu finden, die in ihrer ursprünglichen Form Teil einer unveröffentlichten Dissertation von Barbara Gross ist, und aus der auch die Grafiken für die Streckenelemente in Abbildung 9 entnommen sind, sowie in der Datei *colorcodes.xls*.

47 Vgl. dazu Abschnitt 2.4, S.8.

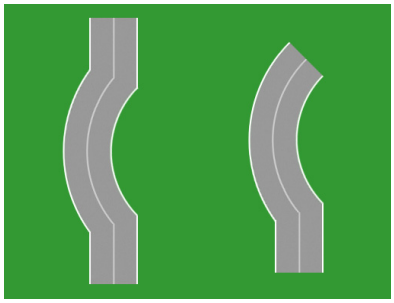

Abbildung 9: Streckenelemente mit Rotwerten 88 (links) und 56 (rechts)

Gegen einen Algorithmus zur Bestimmung des Verlaufs der Strecke mit Hilfe von Rotwerten sprechen mehrere Gründe. Einmal kodieren die Rotwerte zwar den Verlauf, aber nicht die Lage der RL. Ein vertikaler Abschnitt, der mit dem Rotwert 104 kodiert wird,[48] kann links, rechts oder mittig liegen - welcher Fall vorliegt, hängt davon ab, wo der vorherige Abschnitt endet; das Gleiche gilt entsprechend für alle anderen Abschnitte. Dann gibt es Rotwerte, die zusammengesetzte Abschnitte bezeichnen, z.B. die links abgebildeten Elemente (Abbildung 9), bei denen das Streckenelement mit Rotwert 56 gewissermaßen eine Teil-Strecke des Elements mit Rotwert 88 ist. Die Beziehung zwischen Rotwert und Streckenverlauf ist also durchaus komplex. Schließlich hat zwar in der Theorie jeder durch einen Rotwert kodierte Abschnitt eine feste Länge, in der Praxis gibt es aber zuweilen leichte Abweichungen von ein oder zwei Pixeln davon, so dass das exakte Ende des aktuellen Abschnitts vom Algorithmus abgefragt werden müsste (z.B. in einer binären Suche) - hier müsste auch noch der Fall berücksichtigt werden, dass zwei gleiche Abschnitte aufeinander folgen.

4.2 Von XML zu CSV

Für die Berechnung des Flächenfehlers mit Hilfe des LFA-Tools lag die RL schon in Form von XML-Dateien vor [3]. Diese Dateien enthalten allerdings nicht die Daten für eine ganze Strecke, sondern jeweils nur für eine der Kacheln, aus denen die einzelnen Strecken zusammengesetzt sind.[49] Damit mussten für eine Strecke nicht nur die entsprechenden Daten aus den jeweiligen Dateien eingelesen, sondern auch die Daten der einzelnen Kacheln zu einer Strecke miteinander verbunden werden. Da in einer früheren Version von SAM auch die angezeigte Strecke nicht als eine einzige Grafik vorlag, sondern aus den Grafiken der einzelnen Kacheln zusammengesetzt wurde, existierten schon Konfigurationsdateien, die vom Algorithmus zur Konstruktion der Fahrbahn benutzt werden konnten. Der Name einer solchen Konfigurationsdatei besteht aus dem Namen der entsprechenden Strecke sowie der Dateiendung „.txt“, für die Strecke *hauptabschnitt1* lautet er also beispielsweise *hauptabschnitt1.txt*.

Statt nun die Daten der einzelnen Kacheln direkt aus den XML-Dateien einzulesen, wurde entschieden, sie in ein Format umzuwandeln, das weniger Aufwand beim Einlesen erfordert. Als Dateiformat bot sich CSV an. Der Name, der zugleich die dafür vorgesehene Dateierweiterung ist, steht für *comma-separated values*, eine CSV-Datei ist eine Textdatei, in der in jeder Zeile ein oder mehrere durch Kommas getrennte Werte stehen. Das Format ist in der RFC 4180 [20] beschrieben, die allerdings darauf hinweist, dass sie nicht jede Implementierung abdeckt, die zur Zeit der Formulierung dieser RFC existierte. So kann beispielsweise statt des Kommas auch ein anderes Trennzeichen benutzt werden, weshalb CSV z.B. im entsprechenden Wikipedia-Artikel [22] auch als *character-separated values* gedeutet wird. Das CSV-Format wurde oft benutzt, um Daten zwischen Tabellenkalkulationsprogrammen auszutauschen, wird aber inzwischen zunehmend von XML-basierten Formaten wie ODF abgelöst.

Die meisten von SAM benutzten Konfigurationsdateien können als CSV-Dateien mit dem Semikolon als Trennzeichen beschrieben werden, auch deshalb bot sich dieses Format an.

Neben dem Format gibt es noch einen weiteren Unterschied zwischen den XML-Dateien des LFA-Tools und den CSV-Dateien: Die XML-Dateien sind aus Referenzgrafiken für die entsprechenden Kacheln erstellt worden, in denen die RL als durchgehende rote Linie gezeichnet ist. Jedes Pixel der Linie hat einen Eintrag in der entsprechenden XML-

48 Daneben gibt es auch den Rotwert 112 für einen kürzeren vertikalen Abschnitt.
49 Siehe Abschnitt 2.4, S.8.

Datei, so dass an Stellen, an denen diese nahezu horizontal verläuft, es für eine y-Koordinate mehrere x-Koordinaten gibt. In den CSV-Dateien gibt es aber für eine y-Koordinate auch nur eine x-Koordinate, bzw. bei den Kacheln für Gabelungen zwei, was für die Beschreibung des Verlaufs der RL auch vollkommen ausreicht. Außerdem war es so beim Einlesen der Dateien einfacher, zu entscheiden, ob nun eine Gabelung vorlag und welche Werte zum rechten bzw. linken Zweig der Gabelung gehören: bei zwei x-Koordinaten ist dies eine Gabelung und der erste Wert gehört zum linken, der zweite zum rechten Zweig, in allen anderen Fällen ist nur ein Wert vorhanden. Der Aufwand beim Einlesen der Daten wird auch dadurch verringert, dass weniger Koordinaten eingelesen werden und bei Gabelungen weniger Bedingungen geprüft werden müssen.

Die Umwandlung der Dateien erfolge „manuell“ in ein bzw. drei Schritten: zunächst wurde jede XML-Datei mit mehreren globalen Ersetzungsbefehlen in einem Editor in eine CSV-Datei umgewandelt, in der in jeder Zeile eine y-Koordinate sowie die dazu gehörenden x-Koordinaten standen.[50] Für Kacheln, in denen die RL stets so steil verläuft, dass immer nur eine x-Koordinate pro y-Koordinate vorhanden ist, war damit die Konversion abgeschlossen. Die Dateien der übrigen Kacheln wurden in ein Tabellenkalkulationsprogramm (*OpenOffice Calc*) geladen und dort dann mit Hilfe von Formeln (die in entsprechende Tabellenfelder kopiert wurden) die benötigten x-Koordinaten berechnet.[51] Die so gewonnenen Daten wurden dann im letzten Schritt als CSV-Datei (mit Semikolons als Trennzeichen) exportiert.

4.3 Implementierung

Die RL ist als eine eigene Klasse `AAFSupportRacingLine` implementiert. Die Methoden zum Einlesen der Daten einer Kachel konnten problemlos nach dem Vorbild des Einlesens anderer Konfigurationsdateien geschrieben werden, und auch die Implementation des Zusammenführens der einzelnen Kacheln zu einem Array stellte keinen großen Aufwand dar. Das Einlesen einer Kachel geschieht mit Hilfe von `processCsvFile:`, das von `loadRacingLineForCurrentTrack` aufgerufen wird.

Für den Zugriff auf die RL wurden mehrere Methoden geschrieben, um bei Bedarf mehr als einen Wert gleichzeitig abzurufen. Die dahinterstehende Überlegung ist, dass es weniger Aufwand erfordert, auf einen kleinen Array zuzugreifen als auf den mehrere zehntausend Werte umfassenden Array in `AAFSupportRacingLine`.

Außer Parametern für die y-Koordinaten kann auch der Zweig angegeben werden, aus dem die x-Koordinaten stammen, falls eine Gabelung vorliegt. Außerhalb von Gabelungen hat dies natürlich keinen Einfluss auf den zurückgegebenen Wert. Methoden ohne diesen Parameter bedeuten, dass die rufende Instanz keinen Einfluss darauf hat, welcher Zweig benutzt wird, sie sind v.a. für den Fall vorgesehen, in dem ihr bekannt ist, dass keine Gabelung vorliegt. Daneben gibt es noch Methoden wie `rawRacingLine:`, die auf die direkt im Array gespeicherten Werte zugreifen und so bei Gabelungen zwei, ansonsten eine x-Koordinate pro y-Koordinate liefern.

Außerdem gibt es noch in `AAFSupportRacingLine` eine weitere Methode namens `getYAtX:from:to:Branch:`, mit der anhand einer x-Koordinate die y-Koordinate für einen Punkt abgefragt werden kann, in der die Vertikale mit dieser x-Koordinate in einem angegebenen Bereich von y-Koordinaten von der RL geschnitten wird.

Die RL wird nicht nur für die optimale Rückführung benötigt, Automatiken können auch aus anderen Gründen auf sie zugreifen wollen. Nach Implementation der RL zeigte sich dann ein Missverständnis, das mit der Interpretation der y-Koordinaten zusammenhängt[52]. SAM benutzt ein Koordinatensystem, bei dem die y-Koordinaten am unteren

50 Streng genommen sollte in CSV-Dateien jede Zeile gleich viele Zellen enthalten, was hier nicht der Fall war. Da diese Dateien aber nur temporär benutzt wurden, wurde darauf verzichtet, alle Zeilen durch zusätzliche Semikolons auf die gleiche Länge zu bringen, zumal diese Forderung auch nicht allgemein befolgt wird und die meisten Tabellenkalkulationen auch solche Dateien problemlos akzeptieren.

51 Hier fehlende Details werden im folgenden Abschnitt 4.3 nachgeliefert.

52 Vergleiche dazu auch [12], S. 25ff.

Rand der Strecke, also in der Nähe des Startpunkts des TOs, beginnen. Bei Grafiken[53] liegt dagegen der Nullpunkt am oberen Rand, so dass die y-Koordinaten von oben nach unten zunehmen und nicht von unten nach oben wie in SAM. Deshalb wurde die Klasse `AAFSupportRacingLine` von einem anderen Projektmitarbeiter, Andreas Wickert, um Methoden ergänzt, die die korrekten x-Koordinaten für in Graphikkoordinaten angegebene y-Koordinaten liefern.

4.4 Anpassungen und Probleme (Bugfixes)

Bald stellte sich heraus, dass die x-Koordinaten in `AAFSupportRacingLine` teilweise inkorrekt waren. Die Ursache dafür waren Fehler in den XML-Dateien, die bis dahin unentdeckt geblieben waren. Die Datei *SbQ2_rot.xml* enthielt die im Vergleich zur Kachel an der Achse x = 403 gespiegelten x-Koordinaten, *SbJ_rot.xml* enthielt Koordinaten, die z.T. um ein Pixel zu niedrig waren und hatte außerdem eine y-Koordinate zu wenig (womit alle Koordinaten der RL nach dieser Kachel vertikal um 1 `Px` verschoben waren, bzw. um mehrere `Px`, wenn diese Kachel schon mehrfach aufgetreten war), und in *ELr_rot.xml* waren die x-Koordinaten um durchschnittlich 7 `Px` nach links verschoben.[54] Nachdem dann korrekte XML-Dateien für diese Kacheln zur Verfügung standen, wurden diese Fehler dann auch in den CSV-Dateien behoben.

Ein anderer Fehler betraf die Interpretation des Ausdrucks „Abstand zur RL" für den Fall, dass diese nahezu horizontal verläuft. Um dies zu erläutern, sei ein kleiner Ausschnitt aus der Datei *SbF_rot.xml* angeführt:

```
<Y_702>
  <x>325</x>
  <x>326</x>
</Y_702>
<Y_703>
  <x>322</x>
  <x>323</x>
  <x>324</x>
</Y_703>
<Y_704>
  <x>320</x>
  <x>321</x>
</Y_704>
```

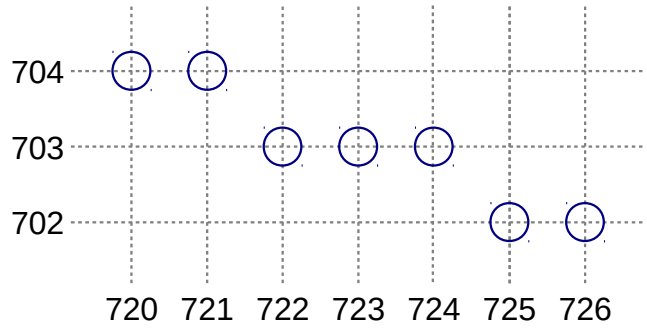

Abbildung 10: Einige Punkte der Racing Line (SbF_rot.xml)

Für die drei y-Koordinaten 702, 703 und 704[55] gibt es also insgesamt sieben x-Koordinaten. Welchen (horizontalen) Abstand hat nun Das TO, wenn es sich im Punkt (300;703) befindet? Die logische Antwort lautet: es sind 22 `Px`, da der nächstgelegene Punkt mit der y-Koordinate 703 bei x = 322 liegt. Entsprechend ist der Punkt (349,4;703) 25,4 `Px` von der RL entfernt, da hier der Punkt (724;703) der horizontal nächste Punkt der RL ist. Der letztgenannte Abstand ist aber von geringer praktischer Bedeutung, da es in diesem Fall optimal ist, so lange horizontal zu fahren, bis die RL erreicht ist, um ihr dann zu folgen[56]. Somit sah *SbFx260806* an der dem obigen Abschnitt von *SbF_rot.xml* entsprechenden Stelle folgendermaßen aus:

```
702;325
703;322
704;320
```

53 Und damit auch in *Squeak*.

54 Die Namen der Kacheln lauten in SAM *SbQ2x260806*, *SbJx260806* und *Elr*, die CSV-Dateien erhielten also diese Namen (mit Dateiendung „.csv").

55 Diese sind relativ zum Beginn (unterem Rand) der Kachel, da diese ja meist mehrfach in einer Strecke auftreten, die Kachel *SbFx260806* kommt beispielsweise in der Strecke *hauptabschnitt1* zweimal vor. Für echte Streckenkoordinaten müssten die y-Werte um den Startpunkt der „Kachelinstanz" erhöht werden.

56 Wie bei der Besprechung des Algorithmus deutlich wird, ist dies leicht vereinfacht, da der weitere Verlauf der RL mit zu beachten ist, allerdings spielt auch dann die Lage der RL in Höhe des TOs keine Rolle.

Der zweite Schritt bei der Konvertierung der XML-Dateien in das CSV-Format[57] bestand also darin, dass für jede Zeile, die mehr als eine x-Koordinate enthielt, je nach Verlauf der Fahrbahn entweder das Minimum oder das Maximum dieser Werte als x-Wert für die CSV-Datei genommen wurde.

Dann stellte sich jedoch heraus, dass dies nicht dem Verlauf der RL entspricht, der vom LFA-Tool benutzt wird. Vielmehr werden in der `function1` in `LFAGetError`[58] erst alle entsprechenden Werte[59] addiert und dann die Summe durch die Anzahl der Werte geteilt,[60] was auf das Gleiche hinausläuft wie das arithmetische Mittel zwischen dem kleinsten und dem größten Wert. Der entsprechende Abschnitt der CSV-Datei muss also folgendermaßen aussehen:

```
702;325.5
703;323
704;320.5
```

Damit mussten also die CSV-Dateien neu erstellt werden, indem im zweiten Schritt statt eines Maximums bzw. Minimums der Mittelwert der betreffenden Werte gebildet wurde.

Die Abbildung 11 stellt den Unterschied dieser beiden Interpretationen graphisch dar. Zwar ist in beiden Fällen die RL eine Treppenfunktion[61], aber die jeweiligen Werte unterscheiden sich voneinander. Außerdem ist in dem Fall, dass das TO sich an einem Punkt mit ganzzahliger y-Koordinate befindet, und die RL eher horizontal verläuft, die Interpretation des Abstands unterschiedlich.

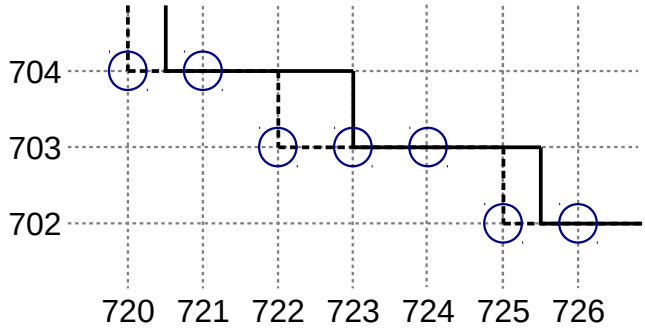

Abbildung 11: Alte (gestrichelt) und neue (durchgezogen) Interpretation der Racing Line (Punkte wie in Abbildung 10)

5 Der Basisalgorithmus

5.1 Basisalgorithmus und Spezialfälle

Streng genommen müssten zur Berechnung einer optimalen Richtung sämtliche Ticks berücksichtigt werden, bis sicher gestellt ist, dass eine lokal optimale Lösung nicht in späteren Ticks zu Folgefehlern führt, die größer sind als ein durch diese Richtung vermiedener Fehler im Vergleich zu einer Lösung, die diese Folgefehler vermeiden. Das kann u.U. bedeuten, den optimalen Kurs des TOs bis ins Ziel zu berechnen. Dies wäre nicht nur zu aufwändig[62], auch würde es der Vorgabe widersprechen, nach denen Automatiken das TO nicht durchgehend zu 100% kontrollieren dürfen, also bei einem längeren Fahrtabschnitt auch die nicht berechenbaren Eingaben der MWB in Betracht gezogen werden müssen. Die Vorausberechnung sollte demnach auf ein sinnvolles Maß beschränkt werden.

Eine sinnvolle Beschränkung besteht darin, mit der Berechnung aufzuhören, sobald das TO die RL erreicht hat. In den meisten Fällen ist es dann sinnvoll, weiter der RL zu folgen, auch dürften die meisten Automatiken den MWB spätestens dann die Freiheit geben,

57 Siehe Abschnitt 4.2, S.15.

58 Diese Java-Methode heißt tatsächlich so! Die Quelldatei befindet sich im ATEO-git auf *assembla* unter tools\lfa\LFA2.0\src\ateo\lfaItem\LFAGetError.java.

59 Bei Gabelungen also nur die Koordinaten des betreffenden Zweiges.

60 Dies ist anscheinend außerhalb des Quellcodes nirgends dokumentiert, jedenfalls ist in [3] und [4] keine entsprechende Angabe zu finden.

61 Und zwar von den y-Koordinaten in die x-Koordinaten, ihre Umkehrung ist natürlich keine Funktion.

62 Entweder müssten in (fast) jedem Tick mehrere Möglichkeiten in Betracht gezogen werden, was zu zahllosen Pfaden führt, die alle durchgerechnet werden müssten, oder es werden für sämtliche denkbaren Positionen die optimalen Richtungen im Voraus berechnet, was sehr viel Speicherplatz für die gefundenen Richtungen bedeuten würde.

wieder selber den Kurs des TOs zu beeinflussen. Der *Basisalgorithmus* hat die Aufgabe, anhand des Verlaufs der RL einen Kurs zu berechnen, der das TO optimal zur RL führt, falls es sich nicht schon auf der RL befindet.

Damit bleiben aber zunächst einige Fälle unberücksichtigt, die während einer Fahrt vorkommen können:

- Bei einem *Hindernis* könnte der Basisalgorithmus einen Kurs berechnen, der zu einer Kollision führt. Dies gilt insbesondere für Hindernisse, bei denen ein Verbleiben auf der RL zu einer Kollision führen würde.
- Bei einer *Gabelung* muss erst geklärt werden, zu welchem Zweig der Gabelung das TO geführt werden soll, bevor der Basisalgorithmus verwendet werden kann.
- Schließlich kann es ein, dass es Stellen gibt, wo es besser ist, nicht der RL zu folgen, sondern eine *Abkürzung* zu nehmen, deren Flächenfehler geringer ist als der so vermiedene Zeitfehler.

Alle diese Fälle bleiben bei dem in diesem Kapitel vorgestellten Algorithmus unberücksichtigt, und müssen dann als *Spezialfälle* in diesen Algorithmus integriert werden.

5.2 Konzeption

5.2.1 Grundsätzliche Überlegungen

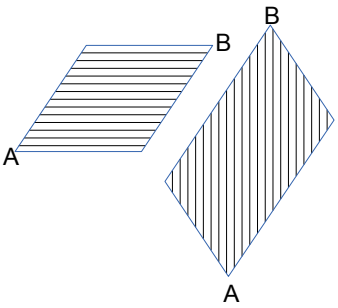

Abbildung 12: Flächen möglicher Wege von A nach B (horizontaler und vertikaler Fall)

Ausgangspunkt der Konzeption war zunächst ein Objekt, dass sich vom TO dadurch unterscheidet, dass es sich kontinuierlich bewegt, im Folgenden KBO (*kontinuierlich bewegtes Objekt*) genannt. Für eine schnellstmögliche Fahrt von A nach B gibt es damit in der Regel[63] unendlich viele Möglichkeiten, die zusammen genommen eine ganze Fläche abdecken. Abbildung 12 zeigt die entsprechenden Flächen für je einen Fall, in dem die Zeit für den Weg von A nach B entweder vom horizontalen oder vom vertikalen Abstand zwischen A und B abhängt. Wenn sich die RL nun rechts von diesen Flächen befindet, so verläuft der schnellste Weg mit dem kleinsten Flächenfehler am unteren bzw. rechten Rand dieser Flächen. Damit besteht ein optimaler Weg für dieses Objekt nur aus horizontalen und diagonalen Abschnitten, wobei *diagonal* bedeutet, dass das Objekt mit maximaler Geschwindigkeit (horizontal v_{xmax}, vertikal v_{ymax}) fährt.

Ein optimaler Weg zur RL besteht also stets nur aus horizontalen und diagonalen Teilstrecken. Abbildung 13 zeigt den prinzipiellen Verlauf eines Wegs für unser Objekt, falls die RL so verläuft, dass es sich lohnt, einen Umweg zu fahren, um an Flächenfehler mehr zu vermeiden, als an Zeitfehler durch den Umweg erzeugt wird. Im letzten Abschnitt zeigt die horizontale Komponente der Bewegungsrichtung von der RL weg, die Annäherung an die RL erfolgt vertikal. Die entscheidende Frage lautet nun: wie lang sind die einzelnen *horizontalen*, normal *diagonalen* und *invers* diagonalen Streckenabschnitte des Wegs?

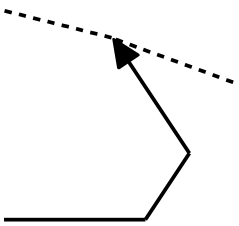

Abbildung 13: Schema für optimalen Weg

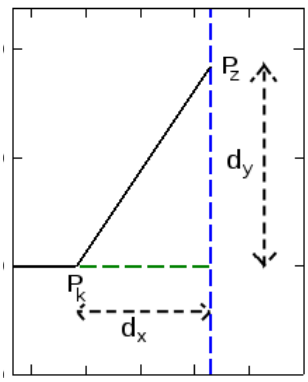

Abbildung 14: Abstände bei Kurs auf eine vertikale RL.

Betrachten wir zunächst den einfachen Fall, dass die RL vertikal verläuft. Hier besteht der optimale Kurs für unser Objekt aus einem horizontalen Weg, der in einem bestimmten Abstand zur RL auf einen diagonalen Kurs abknickt. Je später der Knickpunkt P_K an der RL liegt, desto kleiner ist der Flächenfehler, allerdings ist der Zeitfehler größer, weil zwar die

63 Die Ausnahme ist der Fall, in dem B nicht höher als A liegt: liegt B tiefer als A, ist keine Fahrt möglich, bei gleicher Höhe nur eine horizontale Fahrt.

RL in der gleichen Zeit erreicht wird, aber dann der Auftreffpunkt P_Z niedriger liegt. Da das Objekt ab P_K diagonal mit maximaler Geschwindigkeit (horizontal v_{xmax}, vertikal v_{ymax}) fährt, ist d_y 1,5-mal so lang wie d_x.

Der relative Gesamtfehler im Vergleich zu einer rein horizontalen Fahrt zur RL beträgt somit:

$$\frac{d_x * d_y}{2} - d_y * \delta_x = \frac{1{,}5 * d_x^2}{2} - 1{,}5 * d_x * \delta_x$$

und die Ableitungen nach d_x dazu sind

$$\frac{\partial\left(\frac{1{,}5*d_x^2}{2} - 1{,}5*d_x*\delta_x\right)}{\partial d_x} = 1{,}5*d_x - 1{,}5*\delta_x = d_y - \delta_y .$$

$$\frac{\partial^2\left(\frac{1{,}5*d_x^2}{2} - 1{,}5*d_x*\delta_x\right)}{\partial d_x^2} = \frac{\partial(1{,}5*d_x - 1{,}5*\delta_x)}{\partial d_x} = 1{,}5 > 0 .$$

Damit liegt das Minimum des Gesamtfehlers bei $d_x = \delta_x$ bzw. $d_y = \delta_y$.

Wenn die RL anfangs nicht vertikal verläuft, wirkt sich das zwar auf den Flächenfehler aus, aber für alle Punkte nach dem Beginn des vertikalen Abschnitts der RL ist der Unterschied zu einem Flächenfehler bei exakt vertikal verlaufender RL konstant. Damit liegt auch der optimale Knickpunkt P_K in beiden Fällen an der gleichen Stelle, außer wenn die RL den so gefundenen Kurs kreuzt. Da die RL eine Treppenfunktion von vertikalen zu horizontalen Koordinaten darstellt, kann diese Überlegung somit auf alle Fälle erweitert werden, in denen die RL vorwiegend vertikal verläuft.[64] Wenn die RL vorwiegend horizontal verläuft, ist es auf einer Seite der RL optimal, horizontal bis zur RL zu fahren und ihr dann zu folgen. Auf der entgegengesetzten Seite wird ein Objekt, das genügend weit von der RL entfernt ist, den in Abbildung 13 beschriebenen Kurs fahren.

Wenn nun der Punkt P_k, an dem das Objekt von horizontaler zu diagonaler Fahrt abknickt, um d verschoben wird, und sich die diagonale und inverse Fahrt um d verschiebt (Abbildung 15), so entsteht ein Zeitfehler von $2*d*\delta_y$, da das Objekt erst um d in Richtung RL fährt, um dann diese horizontale Distanz später, wenn es der Strecke folgt, in umgekehrter Richtung zu durchfahren. Die so eingesparte Fläche (in Abbildung 24 eingefärbt) beträgt dagegen $d*s_y$,[65] wobei s_y die vertikale Distanz zwischen P_k und dem Punkt ist, an dem das Objekt auf die RL stößt.

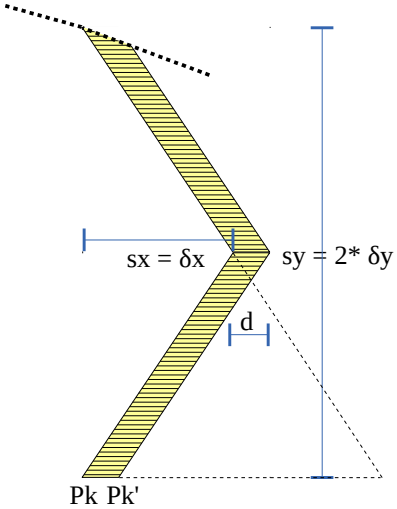

Abbildung 15: diagonale und inverse Fahrt des KBOs.

Daraus ergibt sich, dass der optimale Abknickpunkt so gewählt sein muss, dass der Auftreffpunkt genau $2*\delta_y$ höher liegt als P_k, also $s_y = 2*\delta_y$ gilt. Die Frage, wie lang der diagonale bzw. der inverse Abschnitt eines optimalen Kurses ist. Um dies zu beantworten, hilft die oben gemachte Herleitung des optimalen Knickpunktes für den Fall, dass die RL vorwiegend vertikal verläuft, nur wird statt der RL ein Kurs genommen, der zunächst horizontal verläuft und dann invers diagonal abknickt (gestrichelte dünne Linie in Abbildung 15). Damit ist der diagonale Abschnitt

64 Falls die RL in der Höhe des Zielpunktes horizontal verläuft, ist es irrelevant, welcher Punkt zum Zielpunkt genommen wird: ein um eine Distanz d weiter entfernter Zielpunkt gleicher Höhe erfordert eine zusätzliche horizontale Fahrt der Länge d mit dem entsprechenden horizontalen Zeitfehler, der aber exakt der so eingesparten Fläche von $d*\delta_x$ entspricht, d.h. der Gesamtfehler bleibt gleich.

65 Streng genommen je nach Verlauf der RL nur ungefähr diesen Wert, doch der Unterschied verschwindet, wenn d gegen Null tendiert.

horizontal δ_x bzw. vertikal δ_y lang, woraus sich ergibt, dass auch der inverse Abschnitt des optimalen Kurses die Standardlänge von (δ_y;δ_x) hat. Der optimale Knickpunkt P_k liegt also genau vertikal im Abstand von $2*\delta_y$ unter dem Punkt, an dem das Objekt auf die RL trifft, oder anders formuliert: wenn sich das Objekt in dem Punkt (x_0,y_0) befindet, liegt er bei ($x_{2\delta}$,y_0), wobei $x_{2\delta}$ die x-Koordinate der RL bei $y_0+2*\delta_y$ ist.

An dieser Stelle sei noch kurz darauf hingewiesen, dass zwar ziemlich schnell klar war, dass der inverse Abschnitt eine Länge von horizontal δ_x und vertikal δ_y hat, jedoch ging der Autor irrtümlich davon aus, dass auch in diesem Fall der Knickpunkt zwischen horizontalem und diagonalem Kurs wie beim vorwiegend vertikalem Verlauf der RL eine *Standardlänge* von der RL entfernt ist.[66] Der Fehler fiel erst auf, als der Basisalgorithmus schon implementiert war, so dass dieser entsprechend umgebaut werden musste.

Insgesamt ergibt sich: das Objekt fährt grundsätzlich zunächst *horizontal*, dann *diagonal* und schließlich *invers* diagonal zur RL. Je nach Verlauf der RL kann der optimale Kurs auch schon nach dem horizontalen bzw. diagonalen Abschnitt an der RL landen. Außerdem fällt, wenn das Objekt nah genug an der RL ist, der horizontale und ggf. auch der diagonale Abschnitt weg. Damit ergeben sich folgende Möglichkeiten:

- rein horizontal
- horizontal und dann diagonal
- horizontal, dann diagonal, dann invers
- rein diagonal
- diagonal, dann invers
- rein invers

5.2.2 Der eigentliche Algorithmus

Die Überlegungen im vorigen Abschnitt bezogen sich auf das KBO, das kontinuierlich fährt. Das TO bewegt sich aber diskret in Ticks, in den es nicht die Richtung ändern kann. Der hier vorgestellte Algorithmus besteht nun im Prinzip darin, die für das KBO optimale Bahn zu benutzen und das TO stets zu dem Punkt bewegen (bzw. eine entsprechende Bewegung vorzuschlagen), den das KBO nach einem Tick erreicht hätte.

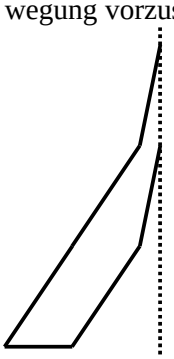

Abbildung 16: horizontal/ diagonal

Hinter dieser Entscheidung steht folgende Motivation: Der Algorithmus soll nicht nur dafür benutzt werden, das TO zu steuern (je nach Konzept mit vollständiger Kontrolle oder einem zusätzlicher Beitrag), sondern auch für die Anzeige einer optimalen Richtung. Nun ist beispielsweise der Übergang von horizontaler zu diagonaler Fahrt in der Regel abrupt: Abbildung 16 zeigt zwei Wege zu einer vertikal verlaufenden RL mit gleichem Fehlerwert, von jedem Punkt, der näher an der RL liegt, ist ein diagonaler Kurs optimal, von jedem Punkt, der weiter von der RL entfernt ist, ist im ersten Tick eine horizontale Bewegung optimal. Ein solcher abrupter Übergang würde die MWB überfordern. In dem in dieser Arbeit vorgestellten Algorithmus gibt es dagegen einen allmählichen Übergang von horizontal zu diagonal.

Eine solche Abweichung vom exakten Optimum ist nur gerechtfertigt, wenn der dadurch erzeugte zusätzliche Fehler nicht zu groß ist.

5.2.3 Fehlerabschätzung

Wenn das TO sich im Abstand *d* vor dem Abknickpunkt von horizontal zu diagonal befindet (Abbildung 17, nächste Seite), beträgt die zusätzliche Fläche im Vergleich zur Fahrt des KBOs:

66 Der Fehler bestand im Wesentlichen darin, dass der Zeitfehler nur mit δ_y angesetzt wurde und nicht korrekt mit $2*\delta_y$.

$$\Delta_F = \frac{d * \frac{v_{ymax}}{v_{xmax}} * (v_{xmax} - d)}{2} = \frac{d * \frac{3}{2} * (v_{xmax} - d)}{2} = \frac{3}{4} * d * v_{xmax} - \frac{3}{4} * d^2.$$

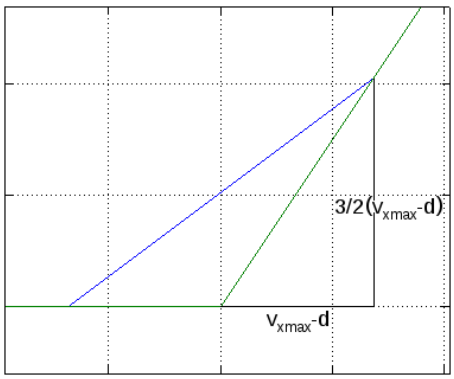

Abbildung 17: "Zusätzliche" Fläche beim Übergang von horizontal zu diagonal, dargestellt wird der worst case.

Die Ableitungen dazu betragen:

$$\frac{\partial \Delta_F}{\partial d} = \frac{3}{4} * v_{xmax} - \frac{3}{2} * d$$

$$\frac{\partial^2 \Delta_F}{\partial d^2} = -\frac{3}{2} < 0.$$

Das Maximum von Δ_F liegt damit bei $d = v_{xmax}/2$, die maximale zusätzliche Fläche beträgt demnach:

$$\Delta_{Fmax} = \frac{3}{4} * \frac{v_{xmax}}{2} * v_{xmax} - \frac{3}{4} * \left(\frac{v_{xmax}}{2}\right)^2 = \left(\frac{3}{8} - \frac{3}{16}\right) * v_{xmax}^2 = 34{,}9525\,\overline{3}\ \mathrm{Px}^2.$$

Wenn das TO sich dagegen im Abstand *h* vor dem Abknickpunkt von vertikal zu invers befindet, so berechnet sich die zusätzliche Fläche wie folgt (vgl. Abbildung 18):

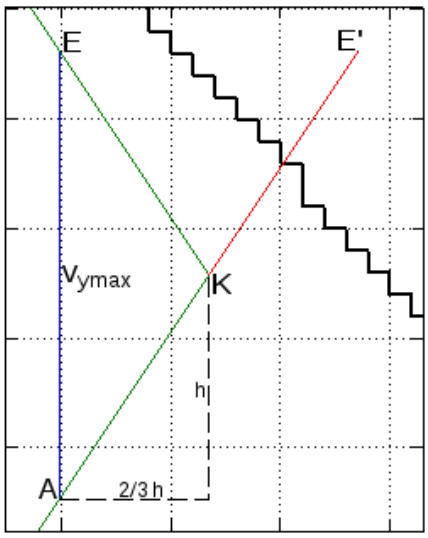

Abbildung 18: "Zusätzliche" Fläche beim Übergang von diagonal zu invers, dargestellt wird der worst case.

Der horizontale Abstand des Anfangspunkt *A* von dem Kurs des TOs zum Knickpunkt *K* beträgt somit $v_{xmax}/v_{ymax} * h = 2/3 * h$, und der Abstand zwischen dem Endpunkt *E* und dem Knickpunkt *K* ist dann $v_{xmax}/v_{ymax} * (v_{ymax} - h) = 2/3 * (v_{ymax} - h)$. Der Abstand von *E* zum Punkt E', den das TO erreichen würde, wenn es rein diagonal fahren würde, ist doppelt so groß und beträgt also $4/3 * (v_{ymax} - h)$.

Die Fläche des Dreiecks $\overline{AKE}$, also die zusätzliche Fläche, ergibt sich als Differenz der Flächen der Dreiecke $\overline{AEE'}$, und $\overline{AKE}$ und beträgt somit

$$\Delta_F = \frac{\frac{4}{3} * (v_{ymax} - h) * v_{ymax}}{2} - \frac{\frac{4}{3} * (v_{ymax} - h) * (v_{ymax} - h)}{2} = \frac{2}{3} * (v_{ymax} - h) * h = \frac{2}{3} * (v_{ymax} * h - h^2).$$

Die Ableitungen dazu betragen:

$$\frac{\partial \Delta_F}{\partial d} = \frac{2}{3} * v_{ymax} - \frac{4}{3} * h$$

$$\frac{\partial^2 \Delta_F}{\partial d^2} = -\frac{4}{3} < 0.$$

Das Maximum liegt damit bei $h = v_{ymax}/2$, im *worst case* beträgt die zusätzliche Fläche somit

$$\Delta_{Fmax} = \frac{2}{3} * \left(v_{vymax} * \frac{v_{ymax}}{2} - \left(\frac{v_{ymax}}{2} \right)^2 \right)$$
$$= \frac{2}{3} * \left(\frac{v_{ymax}^2}{2} - \frac{v_{ymax}^2}{4} \right) \qquad \textit{(Formel 5)}$$
$$= \frac{v_{ymax}^2}{6} = 69{,}9050\overline{6}\ \mathrm{Px}^2.$$

Diese Werte sind eigentlich zu hoch, um solche Abweichungen zu akzeptieren. Jedoch gibt es mehrere Überlegungen, die zeigen, dass die tatsächlich in Betracht zu ziehenden Abweichungen geringer sind:

In Abbildung 19 ist der Kurs des TO zu einem vertikalem Abschnitt der RL laut diesem Algorithmus mit zwei anderen Kursen verglichen, von denen einer je nach dem Abstand zwischen der Startposition des TOs und dem Knickpunkt zwischen horizontaler und diagonaler Fahrt optimal ist. Im Vergleich zum oberen Kurs erzeugt der laut Algorithmus optimale Kurs einen größeren Zeitfehler, aber auch einen geringeren Flächenfehler. Die Differenz der Zeitfehler (umgerechnet in Fläche) beträgt:

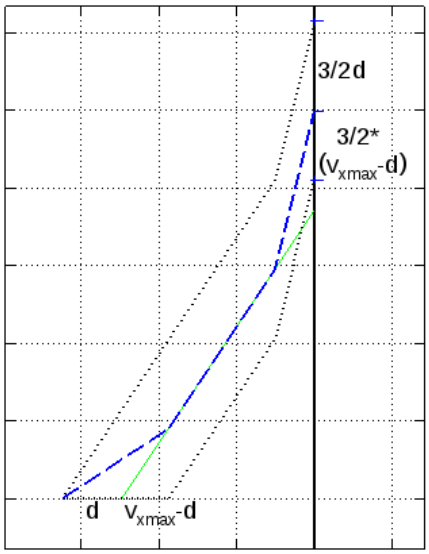

Abbildung 19: Kurs laut Algorithmus (gestrichelt), Alternativen (gepunktet), die je nach Betrag von d optimal sind

$$\Delta_T = \delta_x * \frac{3}{2} * d.$$

Die Differenz der Flächenfehler beträgt nun

$$\Delta_F = -\frac{3}{2} * d * \left(\frac{v_{xmax}}{2} + \delta_x + d - v_{xmax} \right)$$
$$= -\frac{3}{2} * d * \left(\delta_x + d - \frac{v_{xmax}}{2} \right),$$

und damit ergibt sich der relative Gesamtfehler als

$$\Delta_G = \Delta_T + \Delta_F$$
$$= \delta_x * \frac{3}{2} * d - \frac{3}{2} * d * \left(\delta_x + d - \frac{v_{xmax}}{2} \right)$$
$$= \frac{3}{2} * d * \left(\frac{v_{xmax}}{2} - d \right)$$
$$= \frac{3}{4} * v_{xmax} * d - \frac{3}{2} * d^2.$$

Die Ableitungen nach *d* dazu sind:

$$\frac{\partial \Delta_G}{\partial d} = \frac{3}{4} * v_{xmax} - 3 * d$$
$$\frac{\partial^2 \Delta_G}{\partial d^2} = -3 < 0.$$

Und damit ist der maximale Gesamtfehler an der Position $d_{max} = 3/4 * v_{xmax}$ zu finden. Der maximale Gesamtfehler errechnet sich somit als

$$\Delta_{Gmax} = \frac{3}{4} * v_{xmax} * \frac{3}{4} v_{xmax} - \frac{3}{2} * \left(\frac{3}{4} v_{xmax} \right)^2 = \left(\frac{3}{16} - \frac{3}{32} \right) * v_{xmax}^2 \qquad \textit{(Formel 6)}$$
$$= 17{,}47262\overline{6}\ \mathrm{Px}.$$

Analog ergibt sich für den Vergleich mit unteren Alternativweg in dem Abbildung 19 ein relativer Zeitfehler von

$$\Delta_T = -\delta_x * \frac{3}{2} * (v_{xmax} - d)$$

sowie ein relativer Flächenfehler von

$$\begin{aligned}\Delta_F &= \frac{3}{2} * (v_{xmax} - d) * (\frac{v_{xmax}}{2} + \delta_x + d - v_{xmax}) \\ &= \frac{3}{2} * (v_{xmax} - d) * (\delta_x + d - \frac{v_{xmax}}{2})\end{aligned}$$

und damit ein Gesamtfehler von

$$\begin{aligned}\Delta_G &= \Delta_T + \Delta_F \\ &= -\delta_x * \frac{3}{2} * (v_{xmax} - d) + \frac{3}{2} * (v_{xmax} - d) * (\delta_x + d - \frac{v_{xmax}}{2}) \\ &= \frac{3}{2} * (v_{xmax} - d) * (d - \frac{v_{xmax}}{2}) \\ &= \frac{3}{2} * \left(\frac{3}{2} * v_{xmax} * d - \frac{v_{xmax}^2}{2} - d^2\right) \\ &= \frac{9}{4} * v_{xmax} * d - \frac{3}{4} v_{xmax}^2 - \frac{3}{2} * d^2.\end{aligned}$$

Der maximale Relativfehler findet sich hier an der Stelle $d_{max} = 3/4 * v_{xmax}$, und als relativer Gesamtfehler ergibt sich

$$\begin{aligned}\Delta_{Gmax} &= \frac{3}{4} * v_{xmax} * \frac{3}{4} v_{xmax} - \frac{3}{2} * \left(\frac{3}{4} v_{xmax}\right)^2 \\ &= 17{,}4726\overline{26}\ \text{Px}\end{aligned} \qquad \textit{(Formel 7)}$$

und damit der gleiche Wert wie im ersten Fall. Der maximal mögliche zusätzliche Gesamtfehler ist also bei einer Rückführung zu einer (vorwiegend) vertikalen RL nur halb so groß wie der maximal mögliche zusätzliche Flächenfehler. Die Abweichung vom Optimum beträgt damit weniger als 20 Px² und kann somit für diesen Fall auf ein akzeptables Maß beschränkt werden.

In dem in Abbildung 18 (Seite 21) dargestellten Fall, in dem die zusätzliche Fläche besonders groß ist, hilft die Beobachtung, dass die Bewegungen des TOs in den Ticks unmittelbar davor und danach genau auf der diagonalen Ideallinie liegen und somit keinen weiteren zusätzlichen Flächenfehler erzeugen. Wird der gesamte Flächenfehler für den Kurs der Rückführung berücksichtigt, die diesen Fall enthält, so ergeben sich $129{,}9284\overline{6}$ Px² innerhalb von fünf Ticks,[67] also $25{,}9856\overline{93}$ Px² pro Tick.

Dazu kommt noch die Überlegung, dass der zusätzliche Flächenfehler ja „zusätzlich“ ist im Vergleich zum Kurs des KBOs, den das TO gar nicht so nachfahren kann,[68] so dass hier auch hier davon ausgehen können, dass der maximale relative Gesamtfehler niedriger ist als der berechnete maximale Flächenfehler.

67 Ticks, in denen das TO horizontal zur RL fährt, werden hier natürlich *nicht* berücksichtigt.

68 Außer es wird ein Zeitfehler in Kauf genommen, der höher ist als der so eingesparte Flächenfehler.

5.3 Implementation: Klasse `AAFOptimalStep`

Der Basisalgorithmus wird durch die Klasse `AAFOptimalStep` implementiert.[69] Sie stellt mehrere Methoden zur Verfügung, deren Namen alle mit „optimal“ anfangen, und die sich darin unterscheiden, wie im Fall einer Gabelung zu verfahren ist.[70] Der Kern des Algorithmus ist in `optimalFromX:Y:Branch:` implementiert, die ggf. von einer der anderen Methoden aufgerufen wird. Die drei Parameter dieser Methode sind die x- und y-Koordinaten des TOs im SAM-Koordinatensystem[71] sowie der zu benutzende Zweig der RL im Fall einer Gabelung.

5.3.1 Fallunterscheidungen

Insgesamt gibt es acht verschiedene Möglichkeiten, wie sich das TO innerhalb eines Ticks bewegen sollte. Diese acht Fälle sollten nicht mit den auf S.20 aufgezählten sechs Fällen verwechselt werden. In Abbildung 20 ist für jeden dieser Fälle ein Beispiel aufgeführt, neben der RL und der Strecke der vom Algorithmus empfohlenen Bewegung ist auch gestrichelt die zugrunde liegende kontinuierliche Bewegung eingezeichnet.

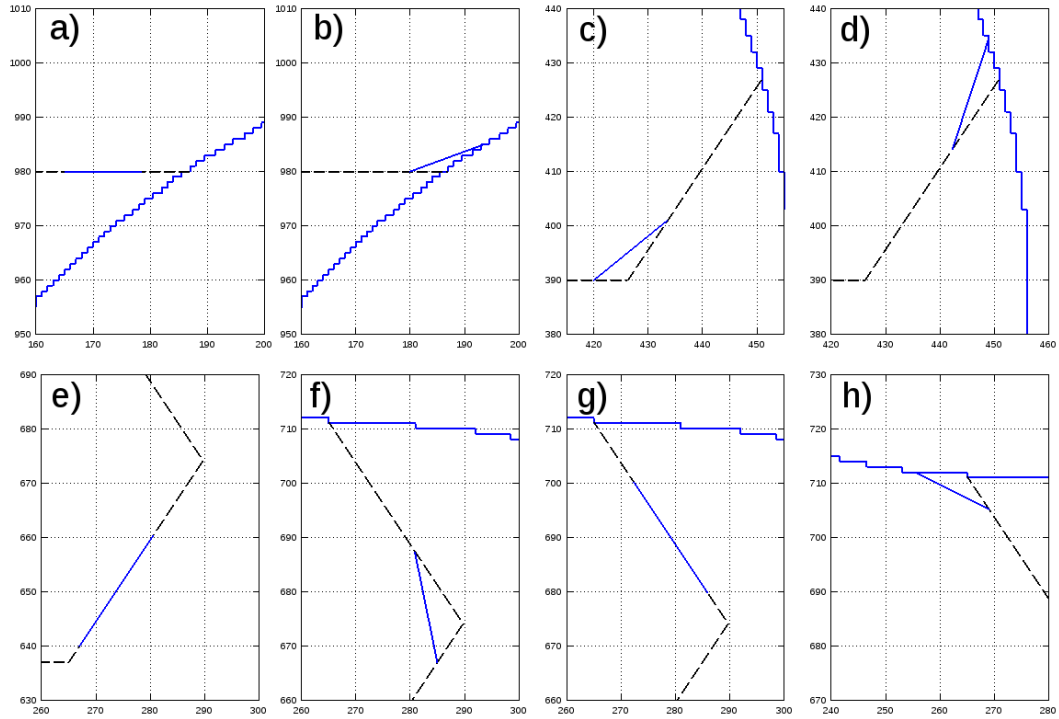

Abbildung 20: Fallunterscheidung in Bezug auf die Bewegung des TOs in einem Tick:
(a) rein horizontal (b) horizontal und RL folgen, (c) horizontal-diagonal (d) diagonal und RL folgen (e) rein diagonal (f) diagonal-invers (g) rein invers diagonal (h) invers diagonal und RL folgen

Zu beachten ist, dass diese acht Fälle von den mit den auf S.20 aufgezählten sechs Fällen unabhängig sind, die sich auf den vollständigen Weg des KBOs bis zur RL beziehen, auch wenn jedem Beispiel ein solcher Weg zugrunde liegt. So hätte für Fall (a) auch ein Beispiel genommen werden können, dem nicht der auch für Fall (b) benutzte rein horizontale Weg zugrunde liegt, sondern z.B. der in den Fällen (c) und (d) oder auch der in den Beispielen für (e) bis (g) benutzte Weg.

Abbildung 20 zeigt nur Beispiele, in denen sich das TO links von der RL befindet, dazu kommen noch die entsprechenden Fälle rechts von der RL. Diese müssen aber nicht gesondert behandelt werden, vielmehr reicht es aus, das Vorzeichen der Differenz zwischen

69 „OptimalStep“ bezieht sich auf den „optimalen Schritt“ in einer Zeiteinheit. Der Name ist insofern unglücklich gewählt, als die Bezeichnung *Step* in SAM schon für einen Versuchsschritt belegt ist, was dem Autor zu spät auffiel. Da *Squeak* beim automatischen Umbenennen Kommentare in allen betroffenen Methoden löscht, erschien es besser, diesen Namen beizubehalten.

70 Näheres siehe Kapitel 6.2, S.27.

71 Vgl. dazu [12], Kapitel 3.2.

den x-Koordinaten der Position des TOs bzw. der RL zu speichern (was in der Variablen `signum` geschieht).[72] Dann können alle Prüfbedingungen und Terme so formuliert werden, dass eine darin enthaltene Multiplikation mit `signum` ausreicht, um beide Seiten der RL abzudecken.

Wie im folgenden Abschnitt deutlich wird, fanden sich noch weitere Möglichkeiten, die Zahl der Fallunterscheidungen zu reduzieren.

5.3.2 Hilfsfunktionen

Der Basisalgorithmus enthält neben den „optimal“-Funktionen noch zwei Hilfsfunktionen: `calculateCrossingX:Y:Ymax:Grad:Branch:` und `calcReturnMx:My:`.

`calculateCrossingX:Y:Ymax:Grad:Branch:` wird benutzt, um den Knickpunkt zwischen *diagonaler* und *inverser* Fahrt zu finden, was Voraussetzung für die korrekte Berechnung des anzusteuernden Punktes im Fall diagonal-invers (f in Abbildung 20) ist. Aus Sicht des Algorithmus befindet er sich auf dem Kreuzungspunkt zwischen dem *diagonalen* Kurs und einer um die *Standardlänge* (δ_x;δ_y) verschobenen RL. Abbildung 21 zeigt dies für ein mit octave berechnetes Beispiel.

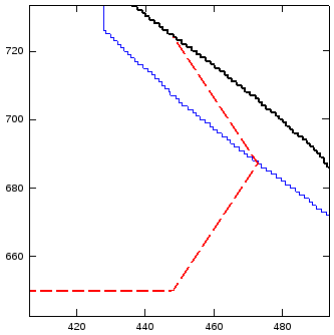

Abbildung 21: Kurs des KBO und Grenze diagonal/invers

`calcReturnMx:My:` hat primär die Aufgabe, die in der Methode `optimalFromX:Y:Branch:` ermittelte Bewegung des TOs von SAM-Koordinaten[73] (`Px`) in Joystick-Koordinaten umzurechnen (vgl. [12, Kapitel 3]). Als Schutz vor Rundungsfehlern wurde auch eine Korrektur für Werte eingebaut, die als Joystick-Koordinaten nicht möglich sind. Diese Korrektur greift nun aber auch dann, wenn die Abweichung größer ist, weshalb der Fall (e) genauso wie (d) behandelt werden kann: im rein *diagonalen* Fall (e) wird der so entstandene Fehler von `calcReturnMx:My:` korrigiert.

Eine weitere Hilfsfunktion, `getYAtX:from:to:Branch:`, wird von der schon erwähnten Klasse[74] `AAFSupportRacingLine` bereit gestellt. Sie dient in den Fällen (b) und (h) von Abbildung 20 dazu, den jeweils horizontal um v_{xmax} vom TO entfernten Zielpunkt auf der RL zu finden. Da in dem Fall, dass die RL im betrachteten Intervall (Parameter `from:to:`) keinen Schnittpunkt mit der Vertikalen der x-Koordinate (`X:`) aufweist, die untere Grenze des Suchintervalls zurückgegeben wird, ist es nicht nötig, die Fälle (a) und (b) mittels einer Fallunterscheidung separat zu behandeln.[75]

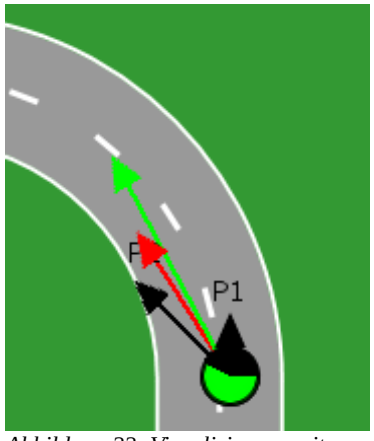

Abbildung 22: Visualisierung mit Hilfe einer frühen Version von „Pfeile am Objekt“, der hellgrüne Pfeil zeigt die berechnete optimale Richtung (verlängert) an.

5.4 Testen

Eine wirklich sinnvolle automatische Testmöglichkeit konnte nicht gefunden werden. Wie eine optimale Rückführung aussieht, ist ja nicht per se gegeben, so dass ein direkter Vergleich damit nicht so einfach möglich ist. Eine Testmethode, die sowohl die Ist- wie die Sollwerte nach diesem Algorithmus berechnet, ist natürlich auch nicht sinnvoll.

Stattdessen wurde der Algorithmus „nach Augenmaß“ getestet: mit Hilfe der jeweils aktuellen Version der von

72 Falls sich das TO auf der RL befindet, wird der weitere Verlauf der RL benutzt, um einen sinnvollen Wert für `signum` zu erhalten.

73 Genauer: die übergebenen Koordinaten enthalten die *Differenz* zwischen der Position des TOs und dem optimalen Zielpunkt für diesen Tick, wobei die Positionen jeweils in SAM-Koordinaten gemessen werden.

74 Siehe Kapitel 4.3, S.15.

75 Außerdem wurden (a,b) und (c) mit Hilfe der Standardmethode `max:` zusammengeführt.

Aydan Seid erstellten Automatik für Pfeile am Objekt[76] (siehe Abbildung 22, vorige Seite) konnte die vom Algorithmus berechnete optimale Richtung am Bildschirm graphisch dargestellt werden. Mit Hilfe des Debuggers von *Squeak* konnte dann nachgeschaut werden, wie die Arbeit des Algorithmus aussah, wenn sich das TO an einem bestimmten Punkt befand.

Auf diese Weise konnten eine ganze Reihe von Fehlern der ersten Version gefunden werden, die meisten betrafen Vorzeichen oder fehlende Klammerungen. Auch der weiter oben erwähnte Fehler, die Grenze zwischen *horizontal* und *diagonal* vor einem *inversen* Kurs falsch anzusetzen[77], wurde bei dieser Art von Testen anhand von Extremfällen deutlich. Im Lauf der Zeit stellte sich auch ein Gefühl dafür ein, wie ein korrektes Verhalten des Algorithmus aussieht. Nachdem also die gröbsten Fehler beseitigt waren, konnten dann mit dem TO gezielt alle möglichen Positionen in Bezug auf einzelne Streckenelemente angefahren und der Algorithmus auf seine Plausibilität überprüft werden.

An einigen Stellen zeigte sich dann, dass der Algorithmus nicht wie erwartet funktionierte, obwohl er korrekt implementiert war. Dies stand stets in Zusammenhang mit Stellen, an denen die RL so verläuft, dass es für ein TO optimal ist, sie zeitweise zu verlassen und eine Abkürzung zu fahren. Dies ist der Grund, warum auch Abkürzungen auf der Liste der Spezialfälle[78] stehen.

Die finale Version des Basisalgorithmus ist so gründlich geprüft worden, dass es äußerst unwahrscheinlich ist, dass auf diese Weise noch weitere Fehler der Implementation gefunden werden.

6 Spezialfall Gabelung: Die Suche nach dem besseren Zweig

6.1 Grenzfunktionen: in octave berechnet

Im Fall einer Gabelung muss ermittelt werden, ob es besser ist, zum rechten oder zum linken Zweig der Gabelung zu gehen. Dies hängt natürlich von der Position des TO ab, in der Nähe eines Zweiges ist es besser, diesen zu benutzen. Die Frage ist, wo die Grenze ist, ab der es besser ist, zum anderen Zweig zu fahren, weil dort ein kleinerer Gesamtfehler erzeugt wird.

Für einen sinnvollen Vergleich muss also für beide Zweige der Gesamtfehler für einen Kurs von der aktuellen Position des TOs bis zum Ende der Gabelung berechnet werden. Dieser Fehler lässt sich aufteilen in

- der Zeitfehler für einen horizontalen Kurs,
- Zeit- und Flächenfehler für den diagonalen und ggf. inversen Weg zur RL,
- der Fehler für den Weg entlang der RL bis zum Ende[79] der Gabelung.

Die ersten beiden Fehler können auch nicht vorhanden sein, also den Wert 0 Px^2 haben.

Da eine direkte Berechnung dieser Fehler ziemlich aufwändig ist, wurden die Grenzen zwischen den Bereichen, an denen es besser ist, zum rechten bzw. linken Zweig der Gabelung zu fahren, mit Hilfe von *octave* berechnet. Der erste Fehler ist einfach zu berechnen, sobald der Punkt bekannt ist, an dem das TO vom horizontalen Kurs abbiegt, für die anderen Fehler wurden entsprechende *octave*-Funktionen geschrieben.

Für die Berechnung des Zeit- und Flächenfehlers für den diagonalen und ggf. inversen Weg zur RL wird nicht der Kurs laut dem Basisalgorithmus zugrunde gelegt, sondern einer, der den Kurs des sich kontinuierlich bewegenden Objekts (KBO) abfährt. Dies hat den Vorteil, dass für Punkte, an denen das KBO zu beiden Zweigen zunächst horizontal

76 Herr Seid hat die Automatiken für visuelle Anzeigen implementiert. Die endgültige, voll konfigurierbare Version ist in [11], Kap. 4.3.1 beschrieben.

77 Siehe Kapitel 5.2.1, S.20.

78 Siehe Kapitel 5.1, S.18.

79 Genau genommen liegt der Punkt, bis zu dem dieser Fehler berechnet wird, hinter dem Ende der Gabelung (aber noch innerhalb der Gabelungs-Kachel), da u.U. ein Kurs erst nach diesem Ende auf die RL traf.

bis zur RL fährt, der zweite und dritte Fehler nur von der vertikalen y-Koordinate des Objekts abhängen. Im einfachsten Fall, wenn dies auf den Grenzpunkt laut diesem Algorithmus zutrifft, konnte dieser direkt aus dem zweiten und dritten Fehler berechnet werden.[80]

Zur Berechnung des Flächenfehlers wurde nur der Abstand des KBOs zur RL an den jeweiligen Knickpunkten zwischen horizontalem, diagonalem und inversem Kurs genommen, sowie in den Mittelpunkten des diagonalen bzw. inversen Kurses. Da δ_x und δ_y jeweils knapp doppelt so groß sind wie v_{xmax} und v_{ymax}, lagen diese Messpunkte also stets näher beieinander als die Positionen des TOs vor und nach einem Tick im Basisalgorithmus. Auf die Berechnung des Zeitfehlers hatte diese Vorgehensweise keinen Einfluss. Zur Fehlerabschätzung können damit die im Zusammenhang mit dem Basisalgorithmus gemachten Überlegungen[81] analog verwendet werden.

Auch für den „Restfehler“ der Fahrt wurde der Kurs des KBOs benutzt. Neben dem Zeitfehler für die Fahrt bis zum Ende der Gabelung kann er auch noch Flächenfehler erhalten, die von Abkürzungen[82] an den Stellen stammen, an denen der Verlauf der RL von vorwiegend horizontal zu vorwiegend vertikal bzw. umgekehrt wechselt. Lediglich bei der runden Kachel (Kachel *RLl*) war im linken Zweig die Krümmung der RL so gering, dass sie im „Rauschen“ der Treppenfunktion nahezu unterging,[83] so dass auf die Berechnung eines Flächenfehlers verzichtet werden konnte.

6.2 Implementation

In der Klasse `AAFOptimalStep` waren ursprünglich zwei Methoden konzipiert, nämlich `optimalFromX:Y:` für den Fall, dass dem Sender dieser Nachricht es egal ist, welcher Zweig der Gabelung benutzt wird, und `optimalForkFromX:Y:Branch:`, die als letzten Parameter den Zweig hat, der benutzt werden soll, wenn eine Gabelung vorliegt. Letztlich erwies es sich als effizienter, mit `optimalFromX:Y:Branch:` eine Methode zu schreiben, die außer `#left` und `#right` auch den Wert `#both` als Parameter für den gewünschten Zweig zulässt. Die beiden anderen Methoden wurden aus Kompatibilitätsgründen beibehalten, sie rufen ihrerseits `optimalFromX:Y:Branch:` auf.[84]

Zur Implementation des Spezialfalls Gabelung wurden drei Methoden geschrieben: `choosebranchX:Y:` wird aufgerufen, wenn `#both` als Parameter übergeben wurde, ermittelt, ob eine Gabelung vorliegt, und ruft ggf. `elrForkMid:` bzw. `rllForkMid:` auf, die Annäherungen der von *octave* berechnete Grenzfunktion für die übergebene y-Koordinate zurückgeben.

Abbildung 23 zeigt die so berechneten Grenzfunktionen (rot) und die benutzten Annäherungen. Es fällt auf, dass sie, und insbesondere die für die runde Gabelung, am Ende der Gabelung irreguläre Ausschläge nach rechts und links zeigen. Dies hängt einerseits damit zusammen, dass für die Ecken zwischen den beiden Zweigen und der vertikalen Strecke nach der Gabelung kein wirklich optimaler Kurs berechnet wird, sowie damit, dass so kurz vor dem Ende der Gabelung die Differenz der Fehler

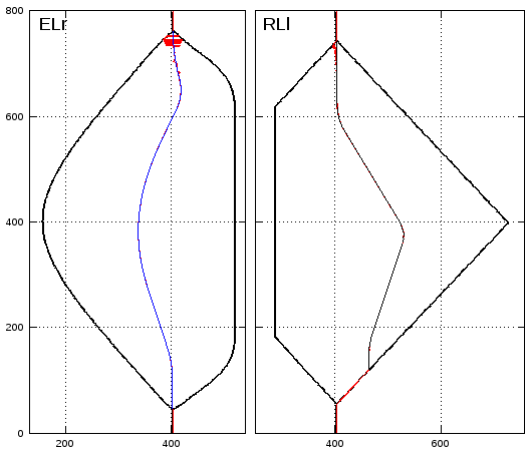

Abbildung 23: Grenzfunktion für die runde (links) und die eckige (rechts) Gabel.

80 In den übrigen Fällen wurde eine binäre Suche verwendet, um ihn auf ein tausendstel Pixel genau zu approximieren.

81 Abschnitt 5.2.3, S.20.

82 Zu Abkürzungen siehe Kap. 7.2, vgl. Abb. 27 zum Verlauf der Gabelungen.

83 Vgl. dazu auch Abb. 29, S. 30.

84 Da es der Semantik von `optimalFromX:Y:` widersprechen würde, wenn dem Sender bekannt ist, welcher Zweig in diesem Fall gewählt wird, wird dies auch nicht dokumentiert.

zwischen beiden Zweigen so gering ist, dass die Unregelmäßigkeiten der *RL*-Treppenfunktion auf das Ergebnis einen verhältnismäßig großen Einfluss haben. Deshalb wurde es für zulässig gehalten, in diesem Bereich für die Grenze eine geschätzte Funktion zu implementieren. Abbildung 24 zeigt dies für die runde Gabelung.

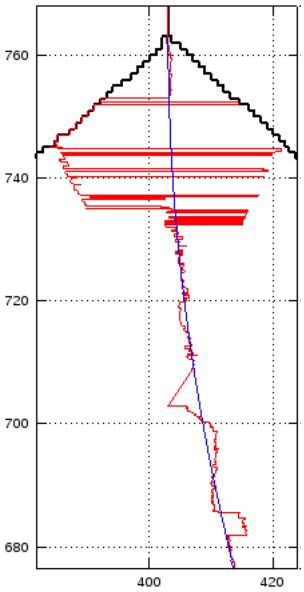

Abbildung 24: berechnete Grenze und Approximation

In den übrigen Fällen ist die Annäherung ziemlich gut, Abbildung 25 zeigt einen vergrößerten Ausschnitt aus der berechneten Grenze für die eckige Gabelung, zusammen mit der implementierten Approximation.

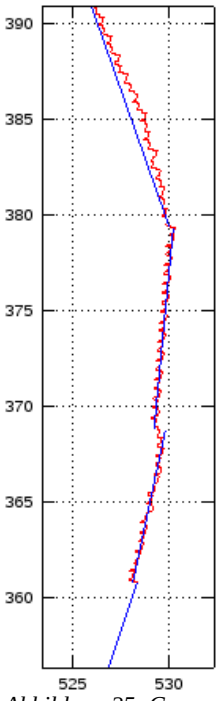

Abbildung 25: Grenzfunktion für ELr

Die Position (y-Koordinate) der Gabeln wird während des Einlesens der *Racing Line* bestimmt: immer dann, wenn eine Gabelungskachel eingelesen wird, wird in `loadRacingLineForCurrentTrack` die Anfangsposition in zwei Pools vom Typ `OrderedCollection` eingetragen, die in der *singleton*-Instanz von `SAMModelData` abgelegt werden.

Eine noch nicht genannte Aufgabe von `choosebranchX:Y:` besteht darin, die Lage des TOs relativ zur Kachel zu bestimmen. Was die x-Koordinaten angeht, so besteht zwischen SAM-Koordinaten und Kachel-Koordinaten kein Unterschied, weshalb ein Aufruf von `elrForkMid:` bzw. `rllForkMid:` mit korrekter y-Koordinate den gewünschten Wert liefert. Für die Umrechnung der y-Koordinate wird auf das `distanceDictionary` (siehe [7, Kap. 6.2.1]) zurückgegriffen. Es enthält u.a. Informationen über die Lage von Gabelungen relativ zur obersten Pixelzeile des TOs, woraus sich die Lage des Mittelpunktes des TOs relativ zur Gabelung bestimmen lässt.

Das `distanceDictionary` gibt, wenn sich das TO innerhalb einer Gabelung befindet, als Antwort auf die Nachricht `at:#nextFork`[85] ein `SAMControllerDistanceObject` zurück, dessen Methode `distance` den negativen Abstand der oberen Spitze des TOs zum Ende der jeweiligen Kachel liefert. Wenn dazu die Länge einer Kachel, also die Höhe der entsprechenden Grafik, addiert wird, so ergibt sich die y-Koordinate der obersten Pixelzeile des TOs relativ zur Kachel. Davon muss noch die halbe Höhe des TOs abgezogen werden. Beide Gabelungskacheln haben die gleiche Höhe von 800 `Px`, und da das TO kreisförmig mit einem Durchmesser von 30 `Px` ist, müssen also zum vom `SAMControllerDistanceObject` gelieferten Wert 785 `Px` addiert werden. Dieser Wert wird direkt als `forkNettoheight` gespeichert.

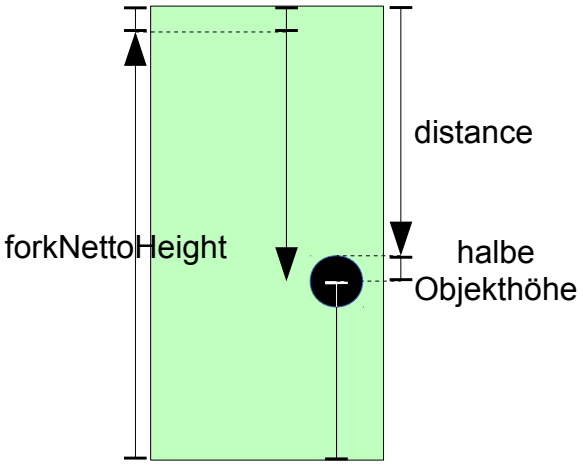

Abbildung 26: Nicht maßstäbliche Zeichnung: wird die negative `distance` zur Kachelhöhe addiert, so ergibt sich die Höhe der Spitze des TOs relativ zur Kachel. Die man die Höhe des erhält am mit TOs mit `forkNettoheight`.

85 Statt dem Abstand zur nächsten Gabelung kann auch mit `#nextElrFork` bzw. `#nextRllFork` der Abstand zur nächsten Gabelung eines bestimmten Typs abgefragt werden.

Damit sieht die Behandlung von Gabelungen folgendermaßen aus: grundsätzlich wird die Rückführung zum jeweils gewünschten Zweig gewählt. Wenn dieser nicht spezifiziert ist, wird er mit Hilfe von `choosebranchX:Y:` ausgewählt, das zunächst prüft, ob eine Gabelung vorliegt, um dann ggf. durch Aufruf von `elrForkMid:` bzw. `rllForkMid:` zu entscheiden, ob es besser ist, zum linken oder zum rechten Zweig zu gehen. Anschließend kann dann der Basisalgorithmus mit diesem Zweig als Ziel ausgeführt werden.

7 Spezialfall Abkürzung

An den Stellen, an denen die RL von *vorwiegend vertikal* zu *vorwiegend horizontal* oder umgekehrt wechselt, lohnt es sich, die Strecke zu verlassen und eine Abkürzung zu benutzen. Dies ist auch schon im Basisalgorithmus enthalten: Sobald die RL von dem Bereich, in dem der Basisalgorithmus eine *diagonale* Fahrt vorschreibt, in den Bereich kommt, in dem *invers* zu fahren ist, verlässt das KBO (und damit das TO) die RL. Dann zeigte sich jedoch, dass dies nicht ausreicht, um das Phänomen der Abkürzungen vollständig zu behandeln.

7.1 Erste Überlegungen

Betrachten wir zunächst Abbildung 27, welche die RL[86] sowie als dünnere Linie eine um *Standardlänge* (δ_x;δ_y) nach links unten verschobene Kopie zeigt. Diese ist rechts von der RL die Grenze zwischen *horizontal* und *diagonal*[87] sowie links von der RL die Grenze zwischen *diagonal* und *invers*. Wenn sich nun das TO auf der Position (0; 420) befindet, müsste es demnach diagonal fahren und damit im horizontalen Bereich bei (410,07$\overline{3}$; 14,89) ankommen (gestrichelte Linie). Das sich kontinuierlich bewegende KBO würde, sobald es den horizontalen Bereich erreicht, dort horizontal fahren, bis es wieder im diagonalen Bereich ist, und somit in etwa dem Verlauf der Grenze zwischen horizontal und diagonal folgen, was offensichtlich nicht optimal ist.

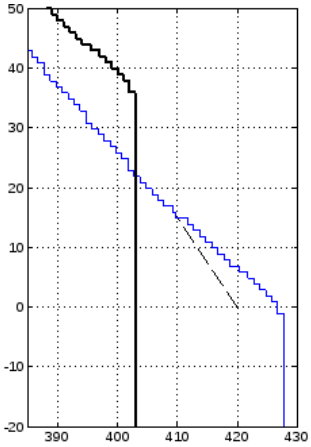

Abbildung 27: RL und Grenze horizontal/diagonal bzw. diagonal/invers, sowie ein Weg zur RL

Deshalb wurden *octave*-Tools zur Analyse von Abkürzungen geschrieben. Eins zeigte neben der RL auch die für die Grenzen relevanten Hilfslinien, Abbildung 27 und weitere Bilder sind damit erstellt worden. Da sich diese Linien bei Abkürzungen typischerweise kreuzen, können damit die Orte, an denen Abkürzungen sind, visualisiert werden. Mit einem weiteren Tool wurde dann eine maximale Abkürzung exakt lokalisiert, und schließlich gibt es ein Tool, mit dem Punkte berechnet wurden, die eine optimale Grenze der jeweiligen Bereiche, also *horizontal*/*diagonal* bzw. *diagonal*/*invers* darstellen. Diese Berechnung basierte auf folgender Forderung: wenn die Länge des (ggf. inversen) diagonalen Kurses einschließlich seiner Fortsetzung als Abkürzung auf der anderen Seite der RL zwischen beiden Seiten der RL aufgeteilt wird, muss die Differenz der beiden Längen genau eine *Standardlänge* sein.

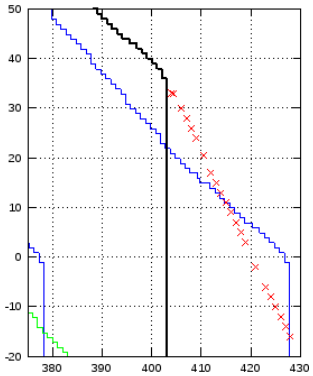

Abbildung 28: berechnete Grenzpunkte für den in Abb. 27 gezeigten Fall.

Da die RL eine Treppenfunktion ist, gibt es in der Regel mehr als einen Punkt, der als Ende der Abkürzungsfahrt angesehen werden kann, die alle bei der Berechnung berücksichtigt wurden. Deshalb ergab die Menge der so berechneten Grenzpunkte keine klare Linie, sondern ein Cluster von Punkten, mit deren Hilfe dann eine sinnvolle Grenze bestimmt wurde. Abbildung 28 zeigt die Punktwolke für den Fall von Abbildung 27. In

86 Und zwar den linken Teil der Eckigen Gabelung *Elr*. Für den Bereich vor der Kachel wird ein vertikaler Verlauf der RL eingezeichnet, aber ein anderer Verlauf hätte keine Auswirkungen auf die Grenze oberhalb von $-\delta_y$ = -37,11.

87 Dies gilt nicht überall, vgl. Abb. 16, S.20, aber z.B. im gezeigten Ausschnitt.

diesem Fall ist die Linie besonders klar erkennbar, in anderen Fällen ist sie weniger deutlich zu sehen. Abbildung 29 zeigt eine ziemlich flache Kurve der runden Gabel *RLl*, bei der offenkundig auf eine Korrektur der Grenze zwischen diagonal und invers verzichtet werden kann.

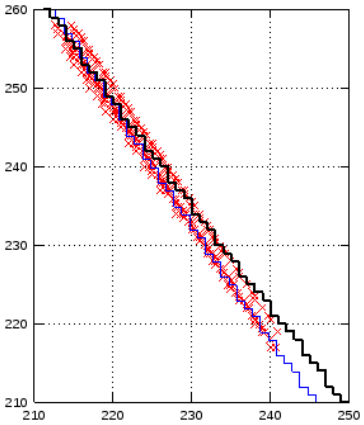

Abbildung 29: Nahezu keine Abkürzung bei RLl, linker Zweig

7.2 Typen von Abkürzungen

Bisher wurden zwei Typen von Abkürzungen erwähnt: In *Typ 1* (Abbildung 28) wechselt die RL von vorwiegend horizontal zu vorwiegend vertikal,und die Grenze *horizontal/diagonal* wird neu bestimmt. In *Typ 2* (Abbildung 29) *w*echselt die RL dagegen von vorwiegend vertikal zu vorwiegend horizontal, und hier muss die Grenze *diagonal/invers* neu bestimmt werden; die Lage der Grenze *horizontal/diagonal* ergibt sich dann daraus, dass sie genau eine *Standardlänge* von der Grenze *diagonal/ invers* entfernt ist.

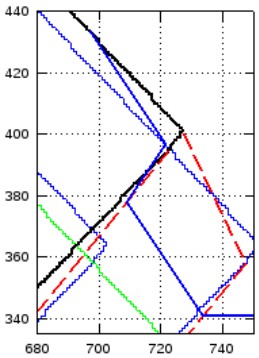

Abbildung 30: Typ3: RL, Grenzen und Kurs des TOs. Grenzen laut Basisalgorithmus sind durchgezogen, Grenzen laut Typ3-Algorithmus rot gestrichelt.

Neben diesen beiden Typen gibt es als *Typ 3* den Sonderfall in der eckigen Gabelung ELr (Abbildung 30), bei dem die Abkürzung nicht aus einer, sondern aus zwei Strecken mit *Standardlänge* besteht. Hier würde ein KBO, wenn es sich von unten rechts der RL (dicke schwarze Linie) nähert, zunächst diagonal, dann invers fahren, die RL kreuzen, womit aus dem inversen Kurs ein diagonaler wird (ohne dass ein Richtungswechsel stattfindet), um dann im letzten Teilstück sich invers von links der RL zu nähern. Die Grenze zwischen *horizontal* und *diagonal* ergibt sich aus der Forderung, den Flächenfehler zu minimieren, da ein Zeitfehler keine Rolle spielt.[88] Der Einfachheit halber wurde dabei der Verlauf der RL als gerade Strecke angenommen. Wenn das Objekt zwischen den Endpunkten der neuen Grenzen horizontal/diagonal und diagonal/invers auf die RL trifft, liegt der gleiche Fall vor wie bei *Typ 1*, so dass diese Grenze mit Hilfe von *octave* bestimmt wurde.

Schließlich stellte sich heraus, dass es noch einen weiteren Typ von Abkürzungen gibt (siehe Abbildung 31), deren Weg allerdings kürzer ist als eine *Standardlänge*, woraus folgt, dass sich die Hilfslinien, die normalerweise die Grenzen zwischen dem horizontalen, diagonalen und inversem Bereich darstellen, bei diesen Abkürzungen nicht kreuzen. Sie sind aber auch ohne spezielle Hilfsmittel leicht zu lokalisieren, da sie stets an den „Haken“ vorkommen, die von Streckenelementen wie in Abbildung 9 auf S.14 gebildet werden. Für diesen Typ wurde ein *octave*-Tool geschrieben, das Anfangs- und Endpunkt einer maximalen Abkürzung berechnet. Dabei stellte sich heraus, dass alle diese Abkürzungen kürzer sind als die maximal mögliche Bewegung des TOs in einem Tick (v_{xmax}, v_{ymax}).

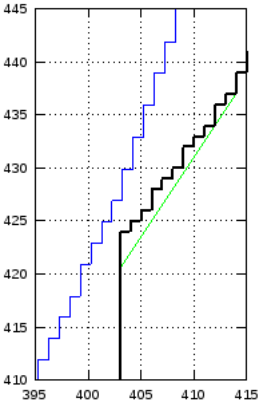

Abbildung 31: Beispiel für Typ 0 (grün) mit Racing Line (Schwarz), und Grenze horizontal/diagonal (blau).

Da die Abkürzungstypen 1 bis 3 aufsteigend nach ihrer Komplexität nummeriert worden waren, wird dieser letzte Typ als *Typ 0* bezeichnet.

7.3 Shortcut-Klassen und -Dateien

Zur Implementation der Abkürzungen wurden fünf Klassen geschrieben (siehe Abbildung 32[89]) die „abstrakte“ Klasse `AAFShortCut`, die keine eigenen Instanzen erzeu-

88 Anders ausgedrückt: für alle in Frage kommenden Fälle ist der Zeitfehler identisch.

89 Das Klassendiagramm wurde mit Hilfe des freien Illustrationstools *Dia* (Version 0.97.2) erstellt.

gen kann, sondern nur Instanzen der anderen Shortcut-Klassen, sowie `AAFShortCut0`, `AAFShortCut1`, `AAFShortCut2` und `AAFShortCut3`. Dabei ist jede Klasse von der des nächstniedrigeren Typs abgeleitet, und `AAFShortCut0` von `AAFShortCut`. Alle bieten die gleichen Schnittstellen: Informationen über die Strecke einer maximalen Abkürzung, die Ausnahme-Bereiche, in denen die Grenze zwischen horizontal und diagonal bzw. diagonal und invers anders ist als im Basisalgorithmus, den gesamten von der Abkürzung betroffenen Bereich[90] und schließlich, auf welcher Seite sich die Ausnahmebereiche befinden. Bei nicht vorhandenen Ausnahmebereichen wird als Anfangs- und Endkoordinate jeweils ein Wert außerhalb (nach bzw. vor) der Strecke angegeben bzw. die Information, dass die übergebene y-Koordinate sich nicht in diesem Bereich befindet. Damit bleibt es der sendenden Instanz überlassen ob sie den Abkürzungstyp erfragt und dann gezielt nur die notwendigen Informationen abruft[91] oder sie eine einzige Vorgehensweise für alle Abkürzungstypen benutzt.

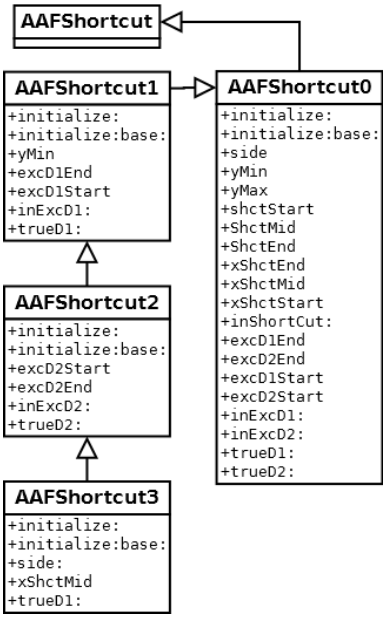

Abbildung 32: Abhängigkeiten und eigene (nicht vererbte) Instanzenmethoden der Shortcut-Klassen, AAFShortCut0 wurde als letzte eingefügt.

Um eine Instanz dieser Klassen zu erstellen, muss die Klassenmethode `new:` aufgerufen werden, die als Parameter einige Kenndaten erwartet, aus denen alle internen Variablen der Instanz berechnet werden können. Diese Kenndaten können aus einer Konfigurationsdatei eingelesen werden, die außerdem noch enthält, welcher Typ von Abkürzung vorliegt.[92] der Parameter, den `new:` erwartet, ist somit das Resultat des Einlesens einer Konfigurationszeile, also eine `Collection` von Zeichenketten.

Die Konfigurationsdateien sind im Prinzip wie andere in SAM und ATEO benutzten Konfigurationsdateien aufgebaut: jede Konfigurationszeile enthält durch Semikolons getrennte Zeichenketten und Zahlen. Der ursprüngliche Plan, alle Zeilen gleich zu formatieren, ließ sich aber nicht durchhalten, da Typ 3 (und ebenso Typ 0) mehr Daten benötigt als die Typ 1 und Typ 2. Als „Erweiterung“ zum allgemeinem Format in ATEO-Konfigurationsdateien ist es möglich, Kommentarzeilen zu erstellen, indem ein Doppelkreuz (#) an den Anfang der Zeile gestellt wird.[93]

Als Typmarkierung wurde für diese Dateien die Erweiterung *shct* gewählt (von Englisch *shortcut*[94]).

Das Einlesen der shct-Konfigurationsdateien wurde in das Einlesen der *Racing Line* integriert. Für jede Kachel wird von `loadRacingLineForCurrentTrack` die Methode `processShctFile:base:` aufgerufen, welche die shct-Datei der entsprechenden Kachel liest und ggf. für jede darin verzeichnete Abkürzung ein entsprechendes Objekt erzeugen lässt. Je nachdem zu welchem Zweig[95] es gehört und welche Seite von den Ausnahmen betroffen ist, wird es dann in einem oder zwei von vier Pools der Klasse `OrderedCollection` abgelegt.

90 Also die Vereinigung der Ausnahmebereiche mit dem Bereich der eigentlichen Abkürzung.

91 Beispielsweise ist es nur in `AAFShortCut3` nötig, die Koordinaten des Mittelpunkts der Abkürzung zu berücksichtigen.

92 Wenn die Methode `new:` von `AAFShortCut` aufgerufen wird, wird mit dieser Information bestimmt, an welche Unterklasse diese Nachricht weitergesendet werden muss.

93 Ähnlich wie in Shellskripts in unixoiden Betriebssystemen darf dabei kein Leerzeichen vor dem Doppelkreuz stehen.

94 Nachdem Recherchen im Internet ergeben hatten, dass in Frage kommende Erweiterungen mit drei Buchstaben schon für andere Dateitypen reserviert waren.

95 Bei Abkürzungen außerhalb von Gabelungen gehört die Abkürzung logisch zu beiden Zweigen!

Zunächst schien damit alles vorhanden zu sein, um die Abkürzungen in den Algorithmus zu integrieren, doch dann zeigte sich, dass die Sache komplizierter ist. Die Details dazu und die weiteren Gründe dafür, warum der Algorithmus nicht zu Ende entwickelt wurde, werden in der folgenden Diskussion ab Abschnitt 8.2 erläutert.

8 Diskussion

8.1 Zusammenfassung des Erreichten

In dieser Arbeit konnten erstens die theoretischen Grundlagen gelegt werden, ohne die kein sinnvoller Algorithmus für eine optimale Rückführung erarbeitet werden kann (Kapitel 3). Die Begriffe des lokalen Fehlern, des horizontalen und vertikalen Zeitfehlers sowie die Möglichkeit, Zeitfehler in Flächenfehler umzurechnen, werden in jedem Algorithmus benötigt, der einen optimalen Kurs berechnen soll.

Außerdem wurde mit der Implementation der *Racing Line* (RL) eine praktische Voraussetzung für die Implementation eines solchen Algorithmus geschaffen (Kapitel 4). In diesem Zusammenhang konnten auch Abweichungen der vom LFA benutzten Dokumentation der RL von der benutzten Darstellung der RL in den Streckengrafiken erkannt und korrigiert werden.

Darauf aufbauend, konnte ein *Basisalgorithmus* konzipiert und implementiert werden, der die „gewöhnlichen Fälle“ abdeckt und eine nahezu optimale Rückführung ermöglicht (Kapitel 5). Auch konnte der Spezialfall der Gabelungen implementiert und in den Basisalgorithmus integriert werden (Kapitel 6).

Auch zum Spezialfall der Abkürzungen konnten wichtige Vorarbeiten geleistet werden (Kapitel 7). Warum Aufgabe, diesen Fall in den Basisalgorithmus zu integrieren, aber nicht gelöst wurde, ist Thema des nächsten Abschnitts.

Die Behandlung des Sonderfalls „Hindernis“ ist kein Teil des erarbeiteten Algorithmus. Die dafür in Angriff genommen Arbeiten sind Teil der Diplomarbeit, die in Anschluss an diese Arbeit erstellt wird.

8.2 Probleme bei Abkürzungen

Wie schon oben am Ende von Kapitel 7.3 erwähnt, sind die Abkürzungen zwar als `AAFShortcut`-Klassen implementiert, aber nicht in den Basisalgorithmus integriert worden. Dafür gibt es mehrere Gründe.

Der erste Grund hängt mit der Entdeckung der Abkürzungen vom Typ 0 zusammen. Auf den ersten Blick scheint das ein problemloser Typ zu sein, da die Abkürzung kürzer ist als die Entfernung, die das TO in einem Tick diagonal fahren kann. Der oben[96] beschriebene Fall einer nicht optimalen Fahrt würde demnach hier so nicht auftreten. Die gemessene Länge der Abkürzung gilt jedoch nur für die Abkürzung, die ein TO nehmen sollte, wenn es sich schon auf der Strecke befindet, es gibt aber auch eine längere Abkürzung auf der anderen Seite der RL, bei der ein analoges Problem auftaucht. Allerdings liegt am Anfang und Ende der Abkürzung die Abweichung von der RL in einer Größenordnung, wie sie auch sonst als Folge der unregelmäßigen Treppenfunktion der RL vorkommt. Somit ist das Gebiet, in dem sich das TO befinden muss, damit der Fehler ins Gewicht fällt, relativ klein, und es stellt sich die Frage, ob dieser Fall berücksichtigt werden muss oder nicht.

Im Zusammenhang mit der Beschäftigung mit Typ 0 trat auch ein zweites Problem ins Bewusstsein: wenn das TO sich in einem Ausnahmebereich befindet und dann auf die RL trifft, sollte es nicht dieser folgen, sondern sie kreuzen, da ja auf der anderen Seite der Bereich einer Abkürzung ist - aber es sollte der RL folgen, wenn es sie zum zweiten mal trifft. Das bedeutet, dass es nicht ausreicht, wie ursprünglich angenommen wurde, die Berechnung der Knickpunkte mit Hilfe von `AAFShortcut`-Objekten zu korrigieren, vielmehr gibt es in Zusammenhang mit Abkürzungen den Fall, in dem ein vorläufig berechneter Kurs nicht korrigiert werden muss, wenn er die RL kreuzt, und den Fall, dass ein

96 Kap. 7.1, S.29.

vorläufig berechneter Kurs zu korrigieren ist, wenn er die RL scheinbar *nicht* kreuzt[97] (tatsächlich aber zweimal kreuzt). Die Bedingungen, unter denen diese beiden Fälle auftreten, weichen außerdem so voneinander ab, dass es nicht möglich ist, sie als zwei Zweige einer einzigen `if`-Abfrage zusammenzufassen. Die Behandlung der mit Abkürzungen zusammenhängenden Ausnahmen müsste also an verschiedenen Stellen des Basisalgorithmus geschehen - hier stellt sich die Frage, ob es am Ende effizienter ist, im Fall einer Abkürzung den optimalen Kurs direkt vom jeweiligen `AAFShortcut`-Objekt berechnen zu lassen.

Aus diesen zwei Gründen wurde die eigentliche Implementation der Abkürzungen verschoben, in der Hoffnung, dass nach einer Denkpause[98] ein erneuter Blick auf diese Probleme neue Perspektiven liefert. Dann jedoch tauchte ein weiterer, dritter Grund auf, der dazu führte, dass die Arbeit an diesem Algorithmus vollständig eingestellt wurde. Er wird im nächsten Abschnitt erläutert.

8.3 Flächenfehler: „real“ und nach LFA

Der hier dargestellte Algorithmus beruht letztlich auf einer Abwägung zwischen Zeit- und Flächenfehler. Der Flächenfehler wird grundsätzlich als die Fläche zwischen RL und dem Kurs des TOs betrachtet. Allerdings wird bei der Auswertung der Versuchsreihen im LFA-Tool nicht der tatsächliche Flächenfehler ausgemessen, denn dies würde ja darauf hinauslaufen, ihn für jede einzelne Pixelzeile der Streckengrafik zu bestimmen und diese lokalen Flächenfehler zu addieren, was sehr hohen Aufwand erfordern würde.[99] Stattdessen werden für die Berechnung des lokalen Flächenfehlers in jedem Tick lediglich die Punkte der RL in Höhe der Anfangs- und Endposition des TOs berücksichtigt. Wenn das TO die RL kreuzt, besteht die Fläche des lokalen Fehlers nicht aus zwei Dreiecken, wie sie sich ergeben, wenn die Fläche zwischen den Polygonzügen der (approximierten) RL und der Bewegung des TOs genommen wird, und wie der Autor es automatisch angenommen hatte. Vielmehr wird die Fläche des Trapez genommen, das aus den vier Messpunkten gebildet wird (siehe Abbildung 33).

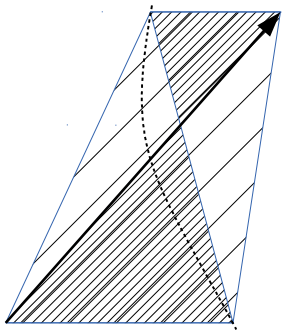

Abbildung 33: Flächenberechnung bei Kreuzung, eng schraffiert die vermutete, weit schraffiert die tatsächliche Fehlerfläche.

Die Überlegungen zu der Frage, wo bei Vorliegen einer Abkürzung die Grenzen zwischen dem *horizontalen, diagonalen* und *inversen* Bereich liegen, sind mit dieser Art der Flächenberechnung nicht kompatibel, da sie darauf zielten, die reale Fläche zwischen TO und RL zu minimieren. Das gesamte Konzept, dass den in den `AAFShortcut`-Klassen bzw. in den shct-Dateien gespeicherten Parametern zu Grunde liegt, wird hinfällig, wenn die „Definition“ des Flächenfehlers, die im Algorithmus des LFA-Tools liegt, berücksichtigt wird. Schließlich ist es kaum sinnvoll, einen Kurs zu berechnen, der zwar bezüglich des realen Zeitfehlers optimal, aber gemäß der Auswertung mit dem LFA-Tool nicht optimal ist.

Die `AAFShortcut`-Klassen bzw. die bei der Initialisierung eines *Steps* erzeugten Instanzen dieser Klassen sind allerdings nicht völlig bedeutungslos: sie können zur Anzeige der Abkürzungen benutzt werden.

Die Bedeutung des Unterschieds zwischen realen Flächenfehler und Flächenfehler laut LFA geht aber über den Fall der Kreuzungen hinaus. Immer dann, wenn die RL nicht geradlinig verläuft, kommt es zu Abweichungen. Ganz zu Beginn der Konzeption des Algorithmus wurde abgeschätzt, dass sich diese Abweichungen in Grenzen halten, weil es

97 Normalerweise muss ein Kurs, der die Strecke kreuzt, so korrigiert werden, dass das TO der Strecke folgt, statt diagonal mit maximaler Geschwindigkeit zu fahren.

98 Also der Beschäftigung mit anderen Aspekten des Algorithmus.

99 Im Fall der Strecke *hauptabschnitt_lang* wären dies rund 117 200 Werte, nämlich 119 610 Pixelzeilen abzüglich der Zeilen am Anfang (vor der Startposition des TOs) und Ende (nach dem Ziel), in denen das TO nicht gefahren ist und somit keinen Flächenfehler erzeugt hat.

sich nur dann lohnt, *diagonal* zur Strecke zu fahren, wenn diese vorwiegend vertikal verläuft. Der größte Fehler würde demnach an den Stellen auftreten, an denen die RL einen Knick enthält, also von exakt vertikal zu ungefähr diagonal mit einer Steigung von 45° oder umgekehrt wechselt,[100] für diesen Fall ergab sich idealisiert ein maximal möglicher Fehler von $46{,}6033\overline{7}\ \mathrm{Px}^2$.[101] Bei der weiteren Entwicklung des Algorithmus wurde nicht beachtet, dass in ihm auch *invers* diagonal zu einer vorwiegend horizontal verlaufenden RL gefahren wird und damit die ursprüngliche Abschätzung obsolet geworden ist.

Von den in Fußnote 101 erwähnten Knicks abgesehen, sind die Streckenelemente stets so zusammengesetzt, dass die idealisierte RL stetig verläuft, also entweder gradlinig (vertikal oder diagonal im Winkel von 45°) oder entlang von Viertelkreisen (90°-Segmenten eines Kreisbogens mit Radius von 150 Px). Der Unterschied zwischen dem „realen" Flächenfehler und der nach dem LFA berechneten Fläche ist bei der idealisierten RL also gleich der Fläche zwischen dem jeweiligen Kreisbogen und der Sehne, die das LFA als Annäherung der RL benutzt. Die Formel für die Fläche eines Kreisabschnitts in einem Kreis mit Radius *r* über einer Sehne der Länge *s* lautet:

$$F_{Sehne} = r^2 * \arcsin\frac{s}{2*r} - \frac{s*\sqrt{4*r^2 - s^2}}{4} \qquad \textit{(Formel 8)}$$

Die maximale Höhe (Differenz der y-Koordinaten) der Sehne ist natürlich v_{ymax}, die maximale Breite (Differenz der x-Koordinaten) v_{xmax}, und damit ist die Sehne maximal $\sqrt{13/9}*20{,}48\,\mathrm{Px} \approx 24{,}614\,\mathrm{Px}$ lang. Die Fläche des Kreisabschnitts beträgt damit

$$\begin{aligned} F_{RLvert} &= (150\,\mathrm{Px})^2 * \arcsin\frac{24{,}614\,\mathrm{Px}}{2*150\,\mathrm{Px}} \\ &\quad - \frac{24{,}614\,\mathrm{Px} * \sqrt{4*(150\,\mathrm{Px})^2 - (24{,}614\,\mathrm{Px})^2}}{4} \\ &\approx 8{,}301\,\mathrm{Px}. \end{aligned}$$

Wenn aber die *inverse* Annäherung an die RL mit berücksichtigt wird, ergibt sich die maximale Breite einer Sehne[102] als

$$\sqrt{(150\,\mathrm{Px})^2 - (150\,\mathrm{Px} - 20{,}48\,\mathrm{Px})^2} \approx 75{,}661\,\mathrm{Px},$$

und eingesetzt in Formel 8 ergibt sich:

$$\begin{aligned} F_{RLallg} &= (150\,\mathrm{Px})^2 * \arcsin\frac{75{,}661\,\mathrm{Px}}{2*150\,\mathrm{Px}} \\ &\quad - \frac{5{,}661\,\mathrm{Px} * \sqrt{4*(150\,\mathrm{Px})^2 - (75{,}661\,\mathrm{Px})^2}}{4} \\ &\approx 245{,}38\,\mathrm{Px}^2. \end{aligned} \qquad \textit{(Formel 9)}$$

Auf den ersten Blick scheint dies im Widerspruch zu stehen zu dem Ergebnis von Hampel [3, Kap. 4.2], wonach der Unterschied in der Berechnung des Flächenfehlers nicht ins Gewicht fällt. Allerdings hat ein Halbkreis mit einem Radius von Radius 150 Px eine Fläche von $\pi*(150\,\mathrm{Px})^2/2 \approx 35.342{,}9\,\mathrm{Px}^2$, und die oben berechnete Fläche beträgt gerade ca. 0,69 % davon. Da es etwas unwahrscheinlich ist, dass bei einer der in [3] beschriebenen Testfahrten dieser Maximalwert erreicht wurde, und so hohe Fehler nur punktuell auftreten, ist die Angabe, dass nur weniger als 1 % Abweichung ermittelt wurde, kein Widerspruch zu Ergebnis in Formel 9.

100 Dies ist bei den Gabelungen der Fall, sowie beim abrupten Übergang zwischen einem vertikalen Abschnitt und einem Viertelkreis, dessen Tangente an der Knickstelle eine Neigung von 45° besitzt, siehe Abbildung 9, S. 14 für zwei Beispiele..

101 In den meisten Fällen liegt der tatsächliche maximale Fehler darunter, nur in der Kachel RLl (runde Gabelung) kann er geringfügig darüber liegen.

102 Wenn das TO mit maximaler vertikaler Geschwindigkeit genau die Höhe erreicht, in der oberste Punkt eines Viertelkreises (und damit der niedrigste Punkt des nächsten Viertelkreises) liegt.

Eine Abweichung von ca. 250 Px² ist allerdings nicht hinnehmbar,[103] zumal diese *lokal* fast ein Viertel der Fläche zwischen vertikaler „Mittellinie“ und RL ausmacht. Hier kann der Unterschied zwischen „realem“ Flächenfehler und dem Flächenfehler nach LFA nicht ignoriert werden. Dies macht eine völlige Neukonzeption des Algorithmus notwendig, die Thema der an diese Arbeit anschließenden Diplomarbeit sein wird.

103 Angesichts der unregelmäßigen Form der RL sind noch höhere als in Ergebnis 9 möglich.

Anhänge

A Termini und Abkürzungen

ATEO: Arbeitsteilung Entwickler-Operateur

diagonal: bezieht sich stets auf eine Bewegung mit sowohl horizontal wie vertikal maximaler Geschwindigkeit. Im Fall einer *invers* diagonalen Bewegung wird in dieser Arbeit das Wort „diagonal“ oft weggelassen.

Flächenfehler: Fehler, der dadurch entsteht, dass das *tracking object* (TO) von den MWB nicht auf der *racing line* (RL) gehalten wird, gemessen als Größe der Fläche zwischen RL und Weg des TOs.

Gesamtfehler: Summe aus Zeitfehler und Flächenfehler

horizontaler Zeitfehler: ein *lokaler* Zeitfehler über eine horizontale Distanz

invers: in Kontext von einer Bewegung des TOs bzw. eines ähnlichen Objekts bezieht sich „invers“ stets auf eine *diagonale* Bewegung, deren horizontaler Anteil von der RL wegführt.

KBO (*kontinuierlich bewegtes Objekt*)*:* ein hypothetisches Objekt, dass sich nicht wie das *tracking object* diskret in einzelnen Ticks bewegt, sondern kontinuierlich, eine Hilfskonstruktion für den Basisalgorithmus.

Kurs:

LFA: Logfile-Auswertung,

lokaler Zeitfehler: Zeitfehler, der daher kommt, dass das TO innerhalb eines oder mehrerer Ticks eine Distanz nicht fährt, entweder im Vergleich zu einer Fahrt mit maximaler Geschwindigkeit (vgl. *Zeitfehler*) oder im Vergleich zu einer Fahrt, die an einer anderen Stelle endet (relativer Zeitfehler). An Stellen, an denen der optimale Kurs des TOs vorwiegend horizontal verläuft, kann dies auch eine horizontale Distanz sein.

MWB: Mikroweltenbewohner (*microworld inhabitant, MWI*), eine Versuchsperson, die in der ATEO/SAM-“Mikrowelt“ das TO steuert

RL: racing line, die Mittellinie der Fahrbahn, auf der die MWB das TO steuern sollen. Fahrfehler sind Abweichungen von dieser Mittellinie, gemessen als Flächenfehler, der Fläche zwischen der RL und dem tatsächlichen Weg des TOs.

SAM: Socially Augmented Microworld, die Mikrowelt

Tick: atomare Zeiteinheit in SAM

TO: tracking object, das von den MWB zu steuernde Objekt

Treppenfunktion der RL: gemeint ist die Abbildung y = RL(x) von y-Koordinaten (vertikale Achse) auf die x-Koordinaten der RL, die eine Treppenfunktion darstellt.

Standardlänge: [Länge einer] diagonale[n] Strecke ($\delta_x;\delta_y$), die für den Basisalgorithmus grundlegend ist.

vertikaler Zeitfehler: ein *lokaler* Zeitfehler über eine vertikale Distanz.

vorwiegend horizontal/vertikal: ein Kurs des TOs mit maximaler horizontaler (v_{xmax}) bzw. vertikaler (v_{ymax}) Geschwindigkeit, i.d.R. Mit nicht maximaler Geschwindigkeit in der jeweils anderen Dimension. Wird sowohl horizontal wie vertikal mit maximaler Geschwindigkeit gefahren, so wird der Ausdruck *diagonal*[er Kurs] benutzt.

Zeitfehler: Fehler, der dadurch entsteht, dass die MWB das TO vertikal nicht so schnell wie möglich bewegen; gemessen in zusätzlichen Ticks im Vergleich zu einer Fahrt mit durchgehend maximaler vertikaler Geschwindigkeit v_{ymax}. Siehe auch *lokaler* Zeitfehler.

B Oft benutzte Werte

δ_x Umrechnungsfaktor eines „vertikalen Zeitfehlers“ (als Strecke) zu einem Flächenfehler, zugleich horizontaler Teil des (diagonalen) Standardabstands zwischen den Grenzen eines Bereichs, in dem das TO diagonal fahren sollte.

δ_y Umrechnungsfaktor eines „horizontalen Zeitfehlers“ (als Strecke) zu einem Flächenfehler, zugleich vertikaler Teil des (diagonalen) Standardabstands zwischen den Grenzen eines Bereichs, in dem das TO diagonal fahren sollte.

v_{xmax} maximale horizontale Geschwindigkeit $13{,}65\overline{3}$ `Px/T` bzw. die maximal in einem Tick zurückgelegte Distanz. In octave `vxmax`, in *Squeak* `vmaxX`.

v_{ymax} maximale vertikale Geschwindigkeit (20,48 `Px`) bzw. die maximal in einem Tick zurückgelegte Distanz. In octave `vymax`, in *Squeak* `vmaxY`.

Literaturverzeichnis

A Papierformat

[1] CRINNER, CORDULA: *Design von Assistenz: Einfluss verschiedener Determinanten auf Assistenzkonzepte von Entwicklern*. Dissertation, Institut für Psychologie der HU Berlin, 2008.

[2] FUHRMANN, ESTHER: *Entwicklung eines GUI für die Konfiguration der Software-Komponente zur Systemprozessüberwachung und -kontrolle in einer psychologischen Versuchsumgebung*. Diplomarbeit, Institut für Informatik der HU Berlin, 2010.

[3] HAMPEL, TOBIAS: *Konzeption und Implementation eines Analysetools zur Auswertung von Logfiles einer komplexen, dynamischen Versuchsumgebung*. Studienarbeit, Institut für Informatik der HU Berlin, 2012.

[4] HAMPEL, TOBIAS: *Weiterentwicklung und Verbesserung einer Analysesoftware für dynamische Versuchsumgebungen*. Diplomarbeit, Institut für Informatik der HU Berlin, 2012.

[5] HASSELMANN, MICHAEL: *Erweiterung einer Softwarekomponente für Automatiken zur Systemprozessüberwachung und -führung als Bestandteil einer psychologischen Versuchsumgebung.* Diplomarbeit, Institut für Informatik der HU Berlin, in Vorbereitung.

[6] KAIN, SASKIA: *Entwickler in komplexen Mensch-Maschine-Systemen: Analyse des Einflusses von Entwicklerressourcen auf den Entwicklungsprozess und das -ergebnis*. Dissertation in Vorbereitung.

[7] KOSJAR, NIKOLAI: *Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur (ATEO).* Diplomarbeit, Institut für Informatik der HU Berlin, 2012.

[8] LEONHARD, CHRISTIAN: *Fenster zum Prozess: Weiterentwicklung eines Operateursarbeitsplatzes im Projekt Arbeitsteilung Entwickler Operateur (ATEO).* Diplomarbeit, Institut für Informatik der HU Berlin, 2012.

[9] NACHTWEI, J. & VON BERNSTORFF, C.: *Between keyhole and clutter effect - A multi-level evaluation of interface extensions using the ATEO master display (AMD).* Manuscript submitted to Human Factors.

[10] NICOLAS NIESTROJ: *Analyse des Verhaltens von Entwicklern komplexer Mensch-Maschine-Systemen*. Dissertation in Vorbereitung.

[11] SEID, AYDAN, *Entwicklung und Konfiguration von visuellen Anzeigen für die Objektsteuerung im Projekt ArbeitsTeilung Entwickler-Operateur (ATEO).* Diplomarbeit, Institut für Informatik der HU Berlin, 2012.

[12] WICKERT, ANDREAS: *Fehler und deren Vermeidung bei der Programmierung und Konfiguration von Automatiken,* Studienarbeit, Institut für Informatik der HU Berlin, 2012.

[13] WICKERT, ANDREAS: *Klassifizierung und Entwicklung von Automatiken für Eingriffe in der Socially Augmented Microworld (SAM)*. Diplomarbeit, Institut für Informatik der HU Berlin, 2012.

B Weblinks

[14] ARBEITSGRUPPE INGENIEURSPSYCHOLOGIE: *Tools*. http://www.psychologie.hu-berlin.de/prof/ingpsy/forschung/tools (abgerufen am 21.01.2014).

[15] BALCK, ANDREW P. ET.AL.: *Squeak by example.* http://www.iam.unibe.ch/~scg/SBE/SBE.pdf (Version of 2009-09-29, zuletzt abgerufen am 21.01.2013).

[16] EATON, JOHN W.: *About GNU Octave*. http://www.gnu.org/software/octave/about.html (abgerufen am 22.01.2014).
[17] FREE SOFTWARE FOUNDATION: *GNU Octave, free software Matlab replacement*. http://www.fsf.org/campaigns/priority-projects/priority-projects/highpriorityprojects#Octave (abgerufen am 22.01.2014).
[18] INGALLS, DAN ET.AL.: *Back to the Future. The Story of Squeak, A Practical Smalltalk Written in Itself*. ftp://ftp.create.ucsb.edu/pub/Smalltalk/Squeak/docs/OOPSLA.Squeak.html (abgerufen am 21.01.2014).
[19] KAY, ALAN: *The Early History Of Smalltalk*. http://www.smalltalk.org/downloads/papers/SmalltalkHistoryHOPL.pdf (abgerufen am 15.01.2014).
[20] SHAFRANOVICH, Y.: *RFC 4180*. http://tools.ietf.org/html/rfc4180 (abgerufen am 31.10.2013).
[21] SHARP,ALEC: *Smalltalk by Example*. (PDF-Conversion by Lukas Renggli) http://stephane.ducasse.free.fr/FreeBooks/ByExample/SmalltalkByExampleNewRelease.pdf (abgerufen am 17.01.2014).
[22] WIKIPEDIA: *CSV (Dateiformat)*. http://de.wikipedia.org/wiki/CSV_(Dateiformat) (abgerufen am 30.10.2013).
[23] WIKIPEDIA: *Squeak*. http://de.wikipedia.org/wiki/Squeak (abgerufen am 21.01.2014).
[24] ZENTRUM MENSCH-MASCHINE-SYSTEME: *Forschungsschwerpunkt 8*. http://www.prometei.de/de/forschungsschwerpunkte/fsp-8.html (abgerufen am 14.1.2014).
[25] ZENTRUM MENSCH-MASCHINE-SYSTEME: *Graduiertenkolleg prometei*. http://www.prometei.de (abgerufen am 9.1.2014).