Dynamische Szenerien in der Fahrsimulation

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität Würzburg

vorgelegt von
Armin Kaußner

aus
Eichstätt

Würzburg, 2003

Eingereicht am: 04.07.2003
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. Hartmut Noltemeier
2. Gutachter: Prof. Dr. Jürgen Wolff von Gudenberg
Tag der mündlichen Prüfung: 03.12.2003

Inhaltsverzeichnis

1. Einleitung

1.1. Fahrsimulation in der Forschung

Moderne Fahrzeuge werden immer komplexer. Über die eigentliche Funktion als Fortbewegungsmittel hinaus bieten sie zusätzliche technische Unterstützung in den Bereichen Sicherheit, Komfort und Fahrvergnügen. So sollen z.B. Assistenzsysteme den Fahrer von der Fahraufgabe entlasten. In der Fliegerei wurden bereits viele Erfahrungen mit solchen automatisierenden Systemen und deren Auswirkung auf die Leistung eines Operators gesammelt. Es hat sich gezeigt, dass neben den positiven Effekten der Entlastung auch negative, aufmerksamkeitsmindernde Konsequenzen zu erwarten sind. Inwieweit diese Erfahrung auf das Fahren übertragbar sind, ist ein aktueller Forschungsgegenstand. Infotainment Systeme, die Telekommunikation, Navigationshilfe und Internetzugang vereinen, werden vom Fahrer gerne in Anspruch genommen. Damit der Fahrer nicht von der eigentlichen Fahraufgabe abgelenkt wird, ist eine benutzergerechte und ergonomische Gestaltung dieser Systeme wichtig. Für Assistenz- und Infotainment Systeme hat sich bewährt, den späteren Nutzer schon sehr früh in die Entwicklung miteinzubeziehen.

Über diese technischen Aspekte der Fahrzeugentwicklung hinaus gewinnen aber auch andere Themen, wie z.B. Auswirkungen von Medikamenten oder psychotropen Substanzen im Straßenverkehr und die Fahreignungsdiagnose beeinträchtigter Personenkreise, immer mehr Aufmerksamkeit und Bedeutung.

Bei der Bearbeitung solcher Fragestellungen setzen sich zunehmend Fahrsimulatoren als methodisches Werkzeug der verkehrswissenschaftlichen Forschung aufgrund einiger entscheidender Vorteile durch (vgl. [16]):

- Komponenten lassen sich an sehr früher Stelle ihres Entwicklungsprozesses in einem Verkehrsumfeld testen. Dadurch werden Entwicklungszyklen verkürzt und Kosten für teure Fahrzeugprototypen eingespart.

- Die Testfahrt in einem Simulator ist für den Fahrer ungefährlicher als in der Realität. Auch stellt der Fahrer kein Risiko für andere Verkehrsteilnehmer dar.
- Die Verkehrssituationen, in denen eine Fragestellung untersucht werden soll, können in der Simulation besser kontrolliert werden.
- Da sowohl das Fahrzeug als auch die Umwelt simuliert wird, stehen zahlreiche Messgrößen zur Verfügung, die zur Analyse einer Fahrt aufgezeichnet werden können. In realen Fahrzeugen ist eine solch umfassende Datenerhebung aufgrund aufwändiger Sensorik häufig problematisch.

Beispiele zum Thema "rapid prototyping" mit Fahrsimulatoren finden sich in [20], [26], [27] und [33]. Fahrtauglichkeitsuntersuchungen bei beeinträchtigten bzw. unter Drogeneinfluss stehenden Personen wurden z.B. in [23], [25] und [28] durchgeführt.

Grundlage eines Versuchs in einem Fahrsimulator bilden auf die jeweilige Fragestellung zugeschnittene Verkehrssituationen, in denen ein Fahrer bestimmte Aufgaben ausführen muss. Diese Szenarien spielen sich an bestimmten Stellen des simulierten Straßennetzwerks ab. Das Straßennetzwerk ist in einem Softwaremodul namens "Datenbasis" abgelegt. Die Datenbasis verwaltet zwei miteinander korrelierte Informationen: Zum einen ist die grafische Repräsentation festgelegt, aus der die Bilder generiert werden, die der Fahrer z.B. über ein Projektionssystem angezeigt bekommt. Hierzu werden die Straße, die umgebende Landschaft sowie die Objekte, die sich auf der Straße (Pfosten, Verkehrsschilder etc.) und in der Landschaft (Bäume, Häuser etc.) befinden, in Polygone unterteilt und mit Texturen gefüllt. Zum anderen ist in der Datenbasis die logische Struktur des Straßennetzwerks gespeichert. Dazu zählt beispielsweise der Verlauf von Fahrspuren, deren Breite und Richtung. Die logische Repräsentation des Straßennetzwerks hat eine zentrale Bedeutung innerhalb der Simulation: Verkehrssituationen werden in ihrem Kontext definiert, zahlreiche andere Softwaremodule (z.B. die physikalische Simulation des Fahrzeugs, die Simulation anderer Verkehrsteilnehmer und Assistenzsysteme) werden von ihr mit Informationen versorgt.

Die übliche Architektur von Fahrsimulator-Datenbasen repräsentiert einen Ausschnitt aus der realen oder einer fiktiven Welt. Solche Datenbasen sind global geometrisch konsistent, d.h. ihr Straßennetzwerk kann im Ganzen als maßstabsgetreue Karte gezeichnet werden. Sie haben jedoch einige Nachteile:

- Aufgrund der Datenmengen ist die Länge des befahrbaren Straßennetzes sehr beschränkt. Selbst in umfangreichen Datenbasen können dadurch z.B. Autobahnfahrten bei hohen Geschwindigkeiten nur wenige Minuten dauern. Fragestellungen, die sich mit den Problemen der Fahrer bei sehr langen Autofahrten beschäftigen, sind damit also nur unzulänglich zu behandeln.

- Verkehrssituationen beziehen sich immer auf bestimmte Stellen im Straßennetzwerk. Wenn für einen Versuch eine feste Abfolge von Situationen wichtig ist, muss der Fahrer genau instruiert werden, welche Route er durch die Datenbasis zu fahren hat. Biegt er an einer Kreuzung versehentlich falsch ab, dann muss der Versuch in der Regel abgebrochen werden.

- Die Abfolge der Verkehrssituationen hängt lediglich von der Route ab, die der Fahrer wählt. Situationen haben aber oft Vorbedingungen an den Zustand des Fahrers (z.B. den Grad seiner Aufmerksamkeit) oder an den Zustand bestimmter Softwaremodule (z.B. aktuelle Geschwindigkeit oder Abstand zu einem vorausfahrenden Fahrzeug). Solche Abhängigkeiten können bei der Reihenfolge der Situationen nicht berücksichtigt werden.

In dieser Arbeit wird ein neuartiges Konzept der Modellierung des Straßennetzwerks in Datenbasen vorgestellt. Das Straßennetzwerk wird topologisch repräsentiert und ist nur lokal geometrisch konsistent. Eine maßstabsgetreue Karte kann nur in einem engen Bereich um den Fahrer, seinem Sichtbarkeitsbereich, angefertigt werden. Dieses Konzept beseitigt die genannten Einschränkungen. Es erlaubt Straßennetzwerke mit nahezu beliebiger Ausdehnung, die sich während der Simulation außerhalb des Sichtbarkeitsbereichs, also unmerklich für den Fahrer, verändern. Durch diese Veränderungen können Fehler, die der Fahrer bei der Wahl seiner Route macht, abgefangen werden. Zudem können die Veränderungen von Eingaben des Versuchsleiters oder von den Messwerten, die in der Simulation erhoben werden, ausgelöst werden[1]. Das Konzept wurde am Fahrsimulator des IZVW[2] implementiert und in mehreren Forschungsprojekten erfolgreich eingesetzt.

[1] Trotz der dynamischen Repräsentation des Straßennetzwerks wird in dieser Arbeit der im Bereich der Simulation übliche Begriff "Datenbasis" für das entsprechende Softwaremodul beibehalten.

[2] Interdisziplinäres Zentrum für Verkehrswissenschaften an der Universität Würzburg

1.2. Gliederung der Arbeit

Die Anforderungen an die neue Form der Datenbasis sind aus Forschungsfragen entstanden, die am IZVW bearbeitet werden. Diese Probleme stammen aus den eingangs erwähnten Themengebieten und werden an einem Fahrsimulator untersucht, der im nächsten Abschnitt kurz vorgestellt wird.

Einen Überblick über bisherige Forschungsarbeiten zur Repräsentation von Straßennetzwerken in Fahrsimulatoren gibt Abschnitt 1.4.

In Abschnitt 1.5 dieser Einleitung wird ein Untersuchungsgegenstand eines aktuellen Forschungsprojekts beschrieben. Einige in der Arbeit angeführte Beispiele beziehen sich repräsentativ auf Aspekte dieser Untersuchung und verdeutlichen, welche Vorteile die vorgestellte Repräsentation des Straßennetzwerks bringt.

Die restlichen Kapitel folgen der Vorgehensweise, die bei der Bearbeitung eines Problems durch Simulation üblich ist: Zunächst wird die Aufgabenstellung formuliert, danach das Problem formalisiert. Nach der Modellierung des Problems kann die Simulation durchgeführt werden. Die Analyse der Ergebnisse der Simulation bringt Erkenntnisse über die Leistungsfähigkeit des Modells und über weitere zu unternehmende Schritte.

In der **Aufgabenstellung** (Kapitel 2) werden die wesentlichen **Anforderungen**, die das Modell des Straßennetzwerks zu erfüllen hat, gesammelt.
Im Kapitel **Formale Beschreibung** (Kapitel 3) wird die **formale Sprache** vorgestellt, die entwickelt wurde, um Datenbasen zu beschreiben. Die detaillierte Darstellung der Sprache soll zum einen demonstrieren, wie Anwender Streckennetze und darauf aufbauend Situationen definieren können, die auf ihre Fragestellungen zugeschnitten sind. Zum anderen werden die genauen Anforderungen und Grenzen für das in Kapitel 4 entwickelte Modell vorgegeben. Alles, was durch die Sprache beschrieben werden kann, muss vom Modell abgedeckt werden. Aspekte der Modellierung, die darüber hinaus gehen, können - mangels Beschreibungsmöglichkeit - vom Anwender auch nicht eingesetzt werden.
Mit Methoden der objektorientierten **Modellierung** wird in Kapitel 4 das Modell des Straßennetzwerks aufgebaut. Dies beinhaltet die **topologische Repräsentation** auf mehreren Detailebenen durch Graphen sowie die **geometrische Repräsentation** von Spurverläufen und Höhenprofilen von Straßen

und Verkehrsknoten.
Das Kapitel **Simulation** (Kapitel 5) beschreibt den Vorgang der **geometrischen Instanzierung**, die aus dem Modell während der Simulation im Sichtbarkeitsbereich des Fahrers ein geometrisch konsistentes Straßennetzwerk erzeugt. Außerdem wird die **Schnittstelle** vorgestellt, mit der andere Softwaremodule der Simulation auf die Informationen der Datenbasis zugreifen können.
Die in dieser Arbeit erreichten neuen **Ergebnisse und Anknüpfungspunkte** für weitere Entwicklungen werden im abschließenden Kapitel 6 herausgestellt.

1.3. Fahrsimulation am IZVW

Der grundlegende Aufbau des IZVW Farhsimulators besteht aus den Komponenten, die in Abbildung 1.1 dargestellt sind. Grau hinterlegte Kästen sind Hardwarekomponenten, weiße Kästen repräsentieren Softwaremodule.

Mockup: Der Fahrer sitzt im sog. Mockup. Beim IZVW-Fahrsimulator ist dies ein BMW 520i, der aus Platz- und Gewichtsgründen hinter der B-Säule abgetrennt ist (Bild 1.2).

Das Mockup ist mit einer Elektronik ausgestattet, die die digitale Erfassung aller Bedieneingaben des Fahrers (Gas, Bremse, Lenkung etc.) sowie die Ansteuerung aller Anzeigeinstrumente (Tachometer, Drehzahlmesser, Signallämpchen etc) erlaubt. Ferner ist an der Lenksäule ein Motor befestigt, mit dem Lenkkräfte simuliert werden können.

Mockup-Schnittstelle: Diese Software übernimmt die Ansteuerung der Elektronik des Mockups.

Physikalisches Fahrzeugmodell: Die wesentlichen Komponenten eines 5er BMWs, wie z.B. Antriebsstrang mit Motor, Lenkung und Reifen werden durch eine Mehrkörper-Simulation nachgebildet (vgl. etwa [17], [38]). Als Eingaben dienen in jedem Simulationsschritt die erfassten Daten aus der Mockup-Schnittstelle, wie z.B. Stellung von Gas-, Bremspedal und aktueller Lenkwinkel, sowie einige Informationen aus der Datenbasis, wie z.B. die Steigung der Straße. Hieraus werden Größen wie Position, Geschwindigkeit und Beschleunigung des Fahrzeugs berechnet. Aufgrund der zahlreichen, schwer zu bestimmenden Parameter, die in das Mehrkörper-Modell des Fahrzeugs eingehen (z.B. Massenträgheiten, Motorkennfelder, Reifeneigenschaften) wird für dieses Modul eine kommerziell erhältliche

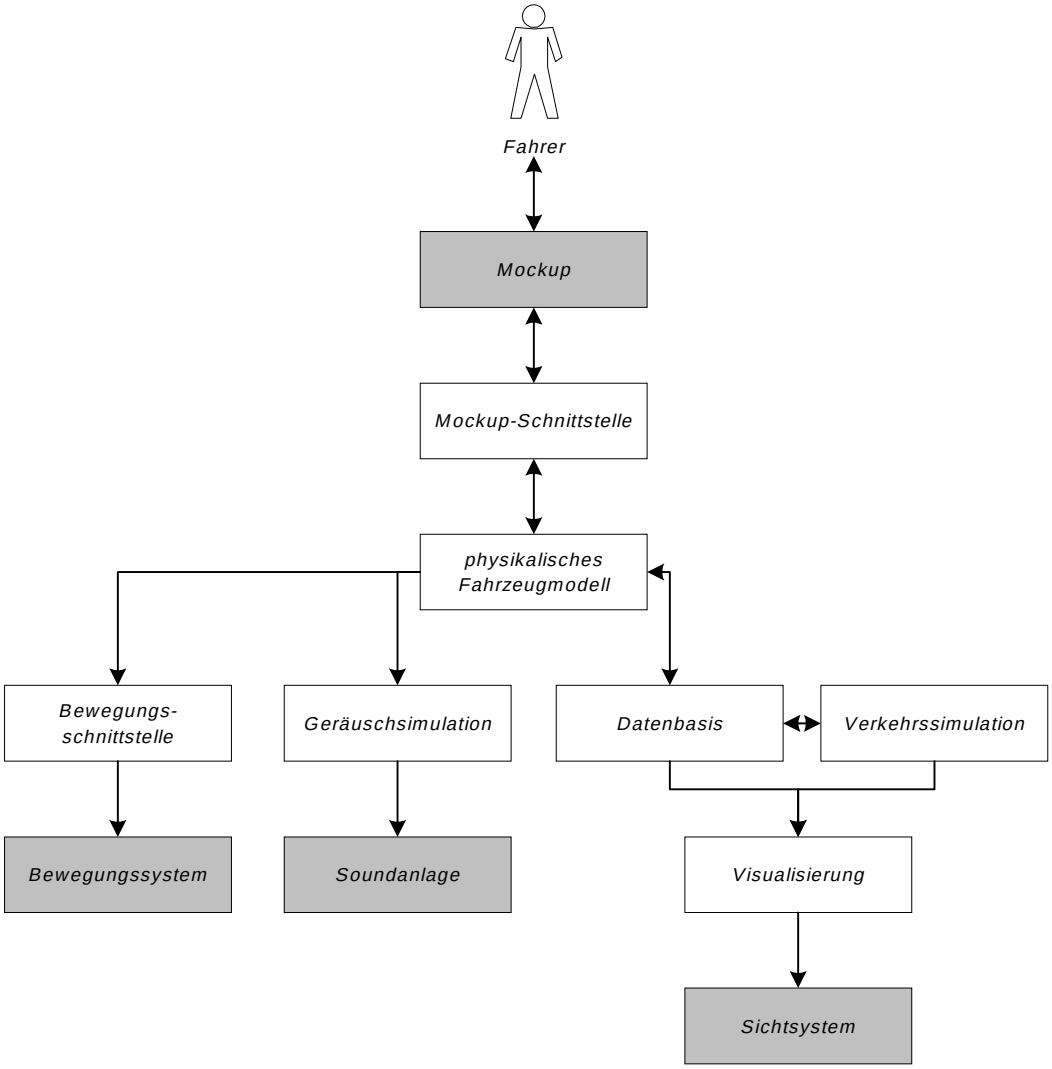

Abbildung 1.1.: Aufbau des IZVW Fahrsimulators.

Abbildung 1.2.: Mockup

Software eingesetzt, deren Kern bei einem Automobilhersteller entwickelt und validiert wurde.

Datenbasis: Die Aufgaben der Datenbasis wurden bereits eingangs kurz erläutert. Mit diesem Softwaremodul beschäftigt sich diese Arbeit.

Visualisierung: Aus den Informationen der Datenbasis und der Verkehrssimulation erzeugt das Modul "Visualisierung" die grafische Darstellung der Straße, ihrer Umgebung und anderer Verkehrsteilnehmer aus der Sicht des Fahrers. Damit während der Fahrt ein flüssiger Bewegungseindruck entsteht, werden diese Bilder mindestens mit einer Frequenz von 60 Hz aktualisiert. Abbildung 1.3 zeigt als Beispiel für die Ausgabe der Visualisierung die Annäherung an eine T-Kreuzung.

Das Visualisierungsmodul ist in [24] und in [29] beschrieben.

Sichtsystem: Die von der Visualisierung generierten Bilder werden dem Fahrer über ein Projektionssystem dargeboten, das ein Sichtfeld von 180 Grad abdeckt (die sphärisch gekrümmte Leinwand ist in Abbildung 1.2 zu erkennen). Zusätzlich sind in den drei Rückspiegeln des Mockups LCD-Displays eingebaut, die die Sicht nach hinten wiedergeben.

Geräuschsimulation: Über die Geräuschsimulation werden der Klang des Motors bei unterschiedlicher Last und Drehzahl sowie Wind- und Rollgeräusche nachgebildet.

Abbildung 1.3.: Annäherung an eine T-Kreuzung

Soundanlage: Die Ausgaben der Geräuschsimulation werden über die Bordlautsprecher des Mockups eingespielt.

Verkehrssimulation: Dieses Modul übernimmt die Simulation anderer Verkehrsteilnehmer. Näheres zur Verkehrssimulation ist [21] zu entnehmen.

Bewegungssystem und -schnittstelle: Das Mockup und das Sichtsystem befinden sich in einer abgeschlossenen, ca. 4 t schweren Kabine, die von 6 beweglichen und drei passiven Aktuatoren getragen wird. Der Aufbau ist der eines Parallel-Manipulators vom Typ Stewart-Plattform (vgl. [40]). Damit lassen sich in beschränktem Umfang Beschleunigungsreize, wie sie beispielsweise beim Bremsen, Kurvenfahren oder durch Unebenheiten der Straße auftreten, simulieren (Abb. 1.4).

Für kurzzeitige Beschleunigungsimpulse entlang der Fahrzeugachsen führen die Aktuatoren eine Translation der Kabine in die entsprechende Richtung aus. Langanhaltende Beschleunigungen werden unter Ausnutzung der Schwerkraft durch ein Kippen der Kabine simuliert. Zur Simulation von Winkelbeschleunigungen wird die Kabine (innerhalb enger Grenzen) rotiert. Die Bewegungsschnittstelle bekommt die aktuellen Beschleunigungswerte aus der physikalischen Fahrzeugsimulation und setzt sie in entsprechende Stellkommandos für die Aktuatoren um. Einen Überblick über die hierbei verwendeten Verfahren und deren Grenzen gibt [31].

Zu dieser Grundkonfiguration kommen, je nach Fragestellung, diverse zusätzliche Module. Hierzu zählen beispielsweise Assistenzsysteme, Displays im

Abbildung 1.4.: Bewegungssystem. Die Kabine ist leicht nach links gekippt, was für den Fahrer die Beschleunigungen beim Durchfahren einer Rechtskurve simuliert.

Mockup und Schnittstellen zu physiologischen Messwertaufnehmern.

Bei den zugrundeliegenden Rechnersystemen von Fahrsimulatoren setzen sich mehr und mehr PCs durch (vgl. z.B. [1], [7], [19]). Vor allem die schnelle Entwicklung im Unterhaltungssektor, z.B. im Bereich der Grafikhardware, beschleunigt diesen Trend. Statt teurer Großrechner läuft die Simulationssoftware verteilt in einem Netz aus mehreren kostengünstigen Standardrechnern. Im IZVW Fahrsimulator werden 12 PCs verwendet. Während die Beschaffung, Erweiterung und Wartung der Hardware dadurch vereinfacht wird, erhöht sich die Komplexität der Steuersoftware des Simulators: Softwaremodule, die auf verschiedenen Computern laufen, müssen zeitlich synchronisiert werden, Daten müssen zwischen den Modulen der einzelnen Rechner in Echtzeit ausgetauscht werden.

Für den IZVW Fahrsimulator wurde deswegen eine Steuersoftware implementiert, die es dem Entwickler erleichtert, neue Softwaremodule zu integrieren, und die den Anwender bei der Konfiguration und Bedienung des Simulators für eine spezielle Fragestellung unterstützt. Details zu dieser Software sind [22] zu entnehmen.

1.4. Grundlegende Arbeiten

Die Arbeit baut auf Konzepte auf, die sich bei Datenbasen für Fahrsimulatoren bewährt haben. Die zugrundeliegenden Arbeiten einiger Forschungsgruppen werden in diesem Abschnitt kurz vorgestellt.

Die zentrale Aufgabe der Datenbasis ist, Informationen für verschiedene andere Softwaremodule der Fahrsimulation zur Verfügung zu stellen. Diese Module können in zwei Gruppen eingeteilt werden.

Die **erste Gruppe** umfasst Module, die auf die **logische Struktur des Straßennetzwerks** zugreifen. Beispiele für solche Module sind die Verkehrssimulation, die Fahrdynamiksimulation und Assistenzsysteme. Grundlage einer logischen Repräsentation bildet ein Modell der Fahrspuren. Hierzu ist der in [3] vorgestellte Ansatz besonders geeignet: Der Verlauf von Spuren wird analytisch, d.h. durch mathematische Funktionen in einem kurvilinearen Koordinatensystem repräsentiert. Es gibt vier Typen von Spurverläufen: Geradenstücke, Kreisbögen, Übergangsbögen und Spurverläufe in Verkehrsknoten. Da diese Einteilung auch beim "realen" Straßenbau vorgenommen wird, können die umfangreichen Regelwerke zur Anlage von Straßen, wie sie z.B. in [36] und [32] zu finden sind, in der Simulation berücksichtigt werden. Das analytische Modell hat ferner den Vorteil, dass Daten wie die Krümmung, Bahntangentenwinkel und Höhenlage von Spuren effizient und genau berechnet werden können. Um sicherzustellen, dass während der Simulation bestimmte Ereignisse (z.B. Verhaltensänderungen der simulierten Verkehrsteilnehmer) reproduzierbar eintreten, werden in den Fahrspuren für den Fahrer unsichtbare Markierungen gespeichert. Die Ereignisse werden ausgelöst, wenn diese sog. Strecken-Events überfahren werden (vgl. [11], [2]).
Das Straßennetzwerk, das sich aus einzelnen Spuren zusammen setzt, wird üblicherweise als Graph modelliert. Es wird unterschieden zwischen Verkehrsknoten und Straßen. In [10], [18], [41] sind Straßen Kanten, denen jeweils eine Menge von Spuren zugeordnet sind. Verkehrsknoten bilden die Ecken des Graphen. Die Struktur findet sich auch in den gängigen Formaten von Straßenverkehrsdaten für die Navigationssysteme realer Fahrzeuge (vgl. [43]). In der vorliegenden Arbeit wird diese Repräsentation leicht abgewandelt: Sowohl Straßen als auch Verkehrsknoten sind Ecken des Graphen, die Kanten stellen mögliche Verbindungen dar, aus denen erst während der Simulation eine einzige ausgewählt wird.

In [13] werden - wie in dieser Arbeit - Straßen durch den Verlauf ihrer exakten Mitte sowie Querschnitts-Informationen modelliert. Im Querschnitt ist u.a. die Anzahl der Spuren, deren Versatz zur Mitte der Straße sowie ihre Breite und Fahrtrichtung gespeichert. Verkehrsknoten werden häufig detailliert als Gebiete modelliert, in denen mehrere Straßen aufeinander treffen. Neben rein geometrischen Daten wie den Spurverläufen werden auch Informationen zur Navigation durch den Verkehrsknoten gespeichert, die beispielsweise von den Fahrzeugen der Verkehrssimulation genutzt werden (vgl. [5], [34]).

Die **zweite Gruppe** von Softwaremodulen, die Informationen aus der Datenbasis beziehen, visualisiert das Straßennetzwerk und die es umgebende Landschaft auf spezieller Grafikhardware. Diese **grafische Repräsentation** ist in einer für die Grafikhardware effizienten Form gespeichert. In der Regel handelt es sich dabei um eine Menge von Polygonen, die mit bestimmten Texturen überzogen werden. Auch können diesen Polygonen verschiedene Material-Eigenschaften zugeordnet werden, die das Erscheinungsbild bei Beleuchtung durch simulierte Lichtquellen beeinflussen. In herkömmlichen Datenbasen, bei denen sich Straßennetzwerk und Landschaft während der Simulation nicht ändern, kann die Zerlegung in Polygone in einem Vorverarbeitungsschritt erfolgen (vgl. z.B. [39]).
Eine Abstraktion dieser Informationen bilden sog. Scene-Graph Repräsentationen, über die der Programmierer komplexe grafische Szenen aufbauen und manipulieren kann. Einen Überblick hierzu gibt [12]. In [4] und [42] werden Verfahren vorgestellt, wie sich die logische Repräsentation - ebenfalls in einem Vorverarbeitungsschritt - aus der grafischen ableiten lässt. Der umgekehrte Weg ist in [10] dargestellt: Bei der Vorverarbeitung wird aus der logischen Repräsentation des Straßennetzwerks die grafische berechnet. Durch das in dieser Arbeit vorgestellte Konzept von Straßennetzwerken, die sich während der Simulation ändern, können die grafischen Daten nur beschränkt vorverarbeitet werden. Die Zerlegung der Szenerie in Polygone muss zur Laufzeit in jedem Schritt der Simulation erfolgen. Als Grundlage hierfür dient die detaillierte logische Repräsentation des Straßennetzwerks. Die Vorgehensweise wird in [24] und [29] beschrieben.

1.5. Exemplarischer Untersuchungsgegenstand

Beim Fahren mit Assistenzsystemen muss der Fahrer nicht nur den Überblick über das jeweilige System sondern auch über die Fahraufgabe und die Verkehrsumwelt behalten. Ob dies immer möglich ist, ist ein Untersuchungsgegenstand des Projekts EMPHASIS[3] am IZVW (vgl. [9]). Die Prüfung, ob die Komplexität einer Aufgabe vom Fahrer erfasst wird, erfolgt üblicherweise über Abfragemethoden, wie z.B. Freezing. Das Simulatorbild wird kurz angehalten und der Fahrer hat über die wesentlichen Details der eben erlebten Fahraufgabe zu berichten. Das Problem dieser Erhebungsmethode ist, dass die Aufgabe für den Fahrer nach kurzer Zeit durchschaubar wird. Er wird sich im weiteren Verlauf des Versuchs nicht mehr auf die Fahraufgabe konzentrieren, sondern sich mehr und mehr die Merkmale einprägen, die beim Freezing abgefragt werden. Es besteht somit die Gefahr, nicht mehr Situationsbewusstsein zu erfassen, sondern einen Wahrnehmungs- und Gedächtnistest durchzuführen.

Die in dieser Arbeit vorgestellte Datenbasis-Architektur erlaubt es, diese Fragestellung auf einem anderen Weg zu untersuchen. In einem Fahrparcours werden Situationen dargeboten, bei denen der Fahrer aufgrund richtigen Verhaltens Situationsbewusstsein beweisen kann. Er wird regelrecht in Fallen gelockt, in denen er entscheiden muss, ob die Nutzung eines bestimmten Assistenzsystems angemessen ist oder nicht.

Der Parcours besteht insgesamt aus 28 Fahraufgaben, die in einer bestimmten Reihenfolge zu absolvieren sind. Die folgenden Beispielsituationen sind diesem Parcours entnommen. Sie sind zur Untersuchung von Auswirkungen des ACC (Adaptive Cruise Control) auf das Fahrverhalten konzipiert. Es handelt sich dabei um einen erweiterten Tempomat, der durch Gas- und Bremseingriffe automatisch Abstand zu einem Führungsfahrzeug hält. Die Reichweite des Sensors, der die vorausfahrenden Fahrzeuge detektiert, beträgt ca. 150 m. In beiden Situationen wird geprüft, ob sich der Fahrer durch das Verhalten des Führungsfahrzeugs beeinflussen lässt. Unter Umständen wird er sich durch das ACC vom Führungsfahrzeug "mitziehen lassen", ohne zu prüfen, ob ein anderes Verhalten angemessener wäre.

[3] EMPHASIS: Effort-Management und Performance-Handling in sicherheitsrelevanten Situationen.

Eine der Fahraufgaben des Parcours ist die sog. "**Navigationsaufgabe**". Auf einem Display in der Mittelkonsole des Mockups geben Pfeile die Richtung an, in die der Fahrer fahren soll. Kurz vor einer Kreuzung beginnen die Pfeile in schneller Folge zu wechseln. Die Instruktion lautet, dass der letzte Pfeil, bevor das Display wieder schwarz wird, die richtige Richtung anzeigt. Ein Führungsfahrzeug wählt dabei manchmal die falsche Richtung. Das Straßennetzwerk ist so gehalten, dass, egal in welche Richtung der Testfahrer fährt, er immer zur nächsten Situation weitergeleitet wird.

Bei der Fahraufgabe "**Scharfe Kurve**" handelt es sich um eine sehr enge Kurve, von der der Fahrer aus einem Trainingsdurchgang weiß, dass sie nur mit ca. 80 km/h fehlerfrei durchfahren werden kann. Ein Führungsfahrzeug gibt die Kurve mit 100 km/h vor. Folgt der Testfahrer vertrauensvoll ohne das ACC abzuschalten, gerät er in Schwierigkeiten. Im Gegensatz zu den Simulatorfahrzeugen kann er im Gefahrenbereich mit dieser hohen Geschwindigkeit die Spur nicht mehr halten. Die scharfe Kurve wird erst dann dargeboten, wenn sicher gestellt ist, dass sich das Führungsfahrzeug im Sensorbereich des ACC befindet.

Im Verlauf dieser Arbeit wird an mehreren Stellen auf diese Beispiel-Situationen "Navigationsaufgabe" und "Scharfe Kurve" Bezug genommen. Auch die restlichen 26 Fahraufgaben des Parcours machen von den in dieser Arbeit vorgestellten Konzepten gebrauch (vgl. [9]). Besonders deutlich lassen sich die Vorteile von dynamischen Straßennetzwerken jedoch anhand der Navigationsaufgabe und der scharfen Kurve demonstrieren.

2. Aufgabenstellung

2.1. Generierung von reproduzierbaren Situationen

Um bei Untersuchungen die Ergebnisse mehrerer Fahrer oder Wiederholungen eines Fahrers vergleichen zu können, müssen die in der Datenbasis abgelegten Situationen reproduzierbar sein. Die Generierung von reproduzierbaren Situationen soll über zwei Mechanismen unterstützt werden.

(1) Zum einen sollen **Ereignisse in bestimmten Streckenabschnitten an genau spezifizierten Orten eintreten**. Dabei wird unter Ereignis die Veränderung von Parametern der an der Simulation beteiligten Softwaremodule verstanden. Ereignisse werden ausgelöst, wenn der Testfahrer oder ein anderes Objekt, das sich auf dem Straßennetzwerk bewegt, sog. "Strecken-Events" überfährt.

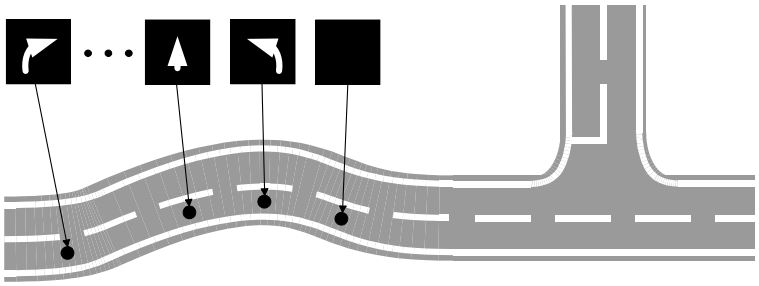

Abbildung 2.1.: Wenn die Strecken-Events (schwarze Punkte) überfahren werden, wechselt die Anzeige im Navigationssystem

Dieser Mechanismus wird beispielsweise in der **Navigationsaufgabe** ein-

gesetzt[1]. Wann das Navigationssystem welchen Richtungspfeil anzeigt, wird durch Strecken-Events definiert, die auf dem Weg zur Kreuzung liegen. Wenn der Fahrer eines dieser Events überfährt, wird die entsprechende Meldung im Navigationssystem ausgegeben (Bild 2.1). Dadurch ist sichergestellt, dass die Fahrer bei jedem Versuch dieselbe Abfolge von Richtungshinweisen mit jeweils demselben Abstand zur Kreuzung erhalten. Insbesondere ist die letzte Anzeige (mit der korrekten Richtungsangabe) bei allen Versuchswiederholungen identisch.

Auch das Softwaremodul, das andere Verkehrsteilnehmer simuliert, kann den Mechanismus der Strecken-Events nutzen, um reproduzierbar spezielle Aktionen einzelner Fahrzeuge auszulösen[2]. Beispielsweise könnte ein Strecken-Event einem Verkehrsteilnehmer beim Überfahren den Befehl geben, bei der nächsten Gelegenheit rechts abzubiegen oder plötzlich scharf zu bremsen.

(2) Der zweite Mechanismus zur Generierung reproduzierbarer Situationen soll das **Auftreten von Streckenabschnitten und der darin definierten Situationen von Bedingungen abhängig** machen. Solche Bedingungen werden an den Zustand des Fahrers und/oder an einzelne Softwaremodule der Fahrsimulation gestellt. Um dies zu ermöglichen muss das Streckennetz während der Simulation verändert werden. Wichtig ist dabei, dass der Testfahrer die Veränderungen nicht bemerkt.

Diese Methode wird z.B. in der Situation **Scharfe Kurve** verwendet[3]. Der Testfahrer soll einem Fahrzeug folgen, das zu schnell durch eine scharfe Kurve fährt. Das Führungsfahrzeug wird von der Verkehrssimulation in großer Entfernung auf der Strecke, die zu der Kurve führt, erzeugt, damit der Testfahrer das plötzliche Erscheinen nicht bemerkt. Erst wenn er das vorausfahrende Fahrzeug eingeholt hat, taucht die scharfe Kurve auf. Die Bedingung für das Auftreten der scharfen Kurve ist also, dass der Abstand des Testfahrers zum vorausfahrenden Fahrzeug einen bestimmten Wert unterschreitet.

[1] vgl. S. 19
[2] Die Steuerung der simulierten Verkehrsteilnehmer wird ausführlicher auf Seite 188 behandelt.
[3] vgl. S. 19

2.2. Kontrolle des Versuchsablaufs

Zur Realitätsnähe der Fahrsimulation gehört, dass der Fahrer das Gefühl hat, seine Route frei wählen zu können. Da die Abfolge von Situationen auf einer Abfolge von Streckenabschnitten beruht, darf der Versuchsablauf durch die Wahlmöglichkeiten des Testfahrers nicht gestört werden. Hierzu soll die Datenbasis Mechanismen bieten, die es für den Anwender vereinfachen, das **Streckennetz für die wahrscheinlichsten Eventualitäten auszulegen**.

Das Problem wird durch die **Navigationsaufgabe** verdeutlicht[4]. Während sich der Testfahrer einer T-Kreuzung nähert, zeigt das Navigationssystem eine Reihe widersprüchlicher Richtungshinweise an. Der Testfahrer hat die Instruktion, der letzten Richtungsangabe vor der Kreuzung zu folgen. Gem. Abbildung 2.1 muss er also links abbiegen. Auf der anschließenden Strecke gelangt er zur nächsten Situation. Für den Fall, dass er fälschlicherweise geradeaus weiter fährt, gibt es zwei Varianten der Situation: In der ersten Variante soll er unmerklich, d. h. ohne eine weitere Kreuzung zu durchfahren, zu der ursprünglichen T-Kreuzung zurück gelangen und die Navigationsaufgabe wiederholen. Die zweite Variante führt ihn - ebenfalls unmerklich - *zur selben* Folgesituation, zu der er bei korrektem Durchfahren der Kreuzung gelangt wäre.

Auch dieser Mechanismus erfordert (wie das im letzten Abschnitt beschriebene bedingte Auftreten von Streckenabschnitten) ein Streckennetz, das während der Simulation verändern kann.

2.3. Einfacher Entwurf von Straßennetzwerken

Der Versuchsleiter entwirft die auf eine bestimmte Fragestellung zugeschnittene Datenbasis selbst. Dabei wird er in mehrerer Hinsicht unterstützt:

(1) Der Versuchsleiter benötigt eine **effiziente und intuitive Möglichkeit, Situationen und dazugehörige Streckennetze zu beschreiben**. Die Beschreibung soll dabei gleichzeitig zur Dokumentation von Experimenten verwendet werden können. Diese Forderungen werden durch eine geeignete Skriptsprache erfüllt.

[4] vgl. S. 19

(2) Die Sriptsprache unterstützt sowohl die **Planung als auch Organisation und Auswertung von Versuchen**. Hierzu bietet sie Möglichkeiten zur situationsbezogenen Modularisierung des Streckennetzes. Die in der Simulation aufgezeichneten Messwerte sollen bei der Auswertung mit gängiger Statistik-Software leicht den Situations-Modulen zugeordnet werden können[5]. Dies erlaubt eine situationsbezogene Auswertung der Daten, wie folgende Beispiele zeigen.

Bei der Auswertung der **Navigationsaufgabe**[6] interessiert sich der Versuchsleiter für die gewählte Route des Testfahrers und den Zeitpunkt, ab dem er gebremst hat. In der Fahraufgabe **Scharfe Kurve**[7] wird die Geschwindigkeit untersucht, mit der der Testfahrer die Kurve durchfährt. Ferner gibt die Güte der Spurhaltung Aufschluss darüber, ob die Geschwindigkeit adäquat gewählt wurde.

Zusätzlich zur angesprochenen Planung, Organisation und Auswertung von Versuchen soll die Modularisierung die Wiederverwendbarkeit von Situationen in unterschiedlichen Versuchen ermöglichen.

(3) Der Anwender soll dabei unterstützt werden, **Lage- und Höhenpläne konform mit den in Deutschland gültigen Richtlinien zum Straßenbau** (RAS-L, vgl. [32], [36]) zu entwerfen. Dabei gibt er gewöhnlich nur die wichtigsten Parameter der Linienführung vor (z.B. Längen von Geraden und Kurven, Kurvenradien, Steigungen). Die restlichen Parameter (z.B. Längen von Übergangsbögen in Kurven) werden automatisch und RAS-L konform gewählt. Es soll aber auch die Möglichkeit geben, die Linienführung detailliert und mit allen Parametern zu spezifizieren. Wenn dabei die RAS-L verletzt werden, soll der Anwender gewarnt werden.

[5] So werden auch größere Datenmengen überschaubar. Bei einer üblichen Messfrequenz von 100 Hz fallen typischerweise pro Stunde Simulatorfahrt 300 MB an Daten an.

[6] Die Navigationsaufgabe ist auf Seite 19 erklärt.

[7] Die Situation "Scharfe Kurve" ist auf Seite 19 erklärt.

2.4. Bereitstellung von Informationen

Die Datenbasis muss **detaillierte Informationen über das Straßennetzwerk zur Verfügung stellen**. Neben der bereits erwähnten Aufzeichnung von Messwerten werden diese Information während der Simulation von verschiedenen Softwaremodulen benötigt, wie die folgenden Beispiele verdeutlichen.

Der Sensor eines Assistenzsystems zur Querführung ist ein Beispiel für ein solches Softwaremodul[8]: In Abhängigkeit von der prädizierten Spurabweichung in einem vorausliegenden Punkt auf der Straße erzeugt das Assistenzsystem Drehmomente am Lenkrad. Diese Momente geben die Richtung vor, in die der Fahrer den Lenkwinkel korrigieren muss, um die Spur zu halten. Die Eingriffe des Systems sind umso stärker, je größer die prädizierte Spurabweichung ist. Die zur Vorhersage der Spurabweichung benötigten Daten erhält das System von einem Sensor. Der Sensor erfragt aus der Datenbasis die aktuelle Spurabweichung des Testfahrers, den Winkel, den sein Fahrzeug mit der Bahntangente einnimmt, sowie die Krümmung der momentanen Fahrspur in einem Punkt, der einen bestimmten Abstand zum Fahrzeug hat. Ferner meldet der Sensor dem System, wenn sich diese Daten nicht eindeutig ermitteln lassen, wie dies z.B. im Innenbereich einer Kreuzung der Fall ist.

Die Verkehrssimulation nutzt die Datenbasis zur Steuerung der Fahrzeuge[9]. Neben grundlegenden Informationen zur Bewegung entlang der einzelnen Spuren können Fahrzeuge auch Navigationsinformationen aus der Datenbasis erhalten. Die Fahrbahnen in einem Verkehrsknoten (wie z.B. Kreuzungen) enthalten Attribute namens "heading", die angeben, ob die Fahrbahn nach links, rechts oder geradeaus führt. Angenommen, ein Fahrzeug des automatischen Verkehrs bekommt in einer bestimmten Situation den Auftrag, bei der nächsten Gelegenheit rechts abzubiegen. Sobald sich der simulierte Verkehrsteilnehmer auf einer Fahrbahn mit entsprechendem "heading" befindet, kann der Abbiegevorgang beginnen.

[8] Die Funktionsweise des Systems wird genauer auf Seite 183 beschrieben.
[9] Die Steuerung der simulierten Verkehrsteilnehmer wird ausführlicher auf Seite 188 behandelt.

2.5. Erweiterbarkeit

Die Skriptsprache und die Repräsentation des Straßennetzwerks müssen so aufgebaut werden, dass sie sich für künftige Fragestellungen leicht **um zusätzliche Modellierungsaspekte erweitern** lassen.

3. Formale Beschreibung

3.1. Notation

Die Sprache, mit der der Anwender Datenbasen definiert, wird anhand kontextfreier Grammatiken spezifiziert. Die Grammatiken enthalten Regeln, mit denen syntaktisch korrekte Beschreibungen von Datenbasen formuliert werden können. Folgendes Beispiel soll dies demonstrieren:

Grammatik 3.1 Reelle Zahl

<Reelle Zahl> →
    $(0,1)$ *<Vorzeichen>*
    <Vorkomma-Anteil>
    $(0,1)$ *<Nachkomma-Anteil>*
<Vorzeichen> →
    + |
    -
<Vorkomma-Anteil> →
    $(1 \ldots \star)$ *<Ziffer>*
<Nachkomma-Anteil> →
    . $(1 \ldots \star)$ *<Ziffer>*
<Ziffer> →
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Grammatik 3.1 definiert die Schreibweise reeller Zahlen, wie z.B. $+123.4$, -53.8 oder 2002. Sie besteht aus einer Reihe von Ersetzungsregeln der Form

<...> → ...

Bezeichner in spitzen Klammern sind Nicht-Terminalsymbole und werden gemäß den Regeln, auf deren linker Seite sie erscheinen, ersetzt. Auf der rechten

Seite stehen entweder wieder Nicht-Terminalsymbole, für die wiederum eine eigene Ersetzungsregel existiert, oder Terminalsymbole. Die Terminalsymbole sind Worte über einer Menge von Zeichen[1].

Nicht-Terminalsymbolen auf der rechten Seite einer Regel sind gelegentlich Quantifizierer vorangestellt, die angeben, wie oft das Symbol ersetzt wird. So besagt die erste Regel in obiger Grammatik beispielsweise, dass eine reelle Zahl aus keinem oder einem Vorzeichen, einem Vorkomma-Anteil und aus keinem oder einem Nachkomma-Anteil besteht. Mögliche Quantifizierer sind:

- kein vorangestellter Quantifizierer: genau einmalige Ersetzung
- $(0, 1)$: keine oder einmalige Ersetzung
- $(1 \ldots \star)$: ein- oder mehrmalige Ersetzung
- $(,)$: ein- oder mehrmalige Ersetzung. Die einzelnen Ersetzungen werden durch Kommas getrennt.
- $(\star)$: keine oder mehrmalige Ersetzung

Für die Ersetzungen auf der rechten Seite einer Regel kann es Alternativen geben, die durch das Zeichen | getrennt sind. So kann in Grammatik 3.1 das Nicht-Terminalsymbol *<Vorzeichen>* durch die Symbole + oder − ersetzt werden. Auch die letzte Regel der Grammatik nutzt diese Schreibweise: Eine Ziffer ist eine 0 oder eine 1 oder ... oder eine 9.

Die Spezifikation der Beschreibungssprache wird in mehrere Teil-Grammatiken aufgeteilt, um beschreibenden Text einfügen zu können. Wenn eine Teil-Grammatik dabei ein Nicht-Terminalsymbol verwendet, dessen Ersetzungsregel in einer anderen Teil-Grammatik definiert ist, wird darauf verwiesen. Folgende Grammatik für Längenangaben demonstriert die dabei verwendete Schreibweise:

[1] Es handelt sich dabei um die Zeichen, die der Anwender bei der Beschreibung tatsächlich eingibt.

Grammatik 3.2 Längenangaben

<Länge> →
    <Reelle Zahl (→ 3.1, S.27)*>*
    <Längeneinheit>
<Längeneinheit> →
    km | m | dm | cm | mm

Eine Längenangabe besteht aus einer reellen Zahl und einer Längeneinheit. Im Nicht-Terminalsymbol *<Reelle Zahl>* wird auf die Grammatik für reelle Zahlen verwiesen.

3.2. Karten und Module

Die folgenden Abschnitte erläutern die Sprache in einer "top-down" Reihenfolge: Jeder Versuch findet in einer sog. **Karte** statt (dieser Abschnitt). Karten bestehen aus einer bestimmten **Abfolge von Situations-Modulen** (3.3). Diese wiederum enthalten **Streckennetze**, die sich aus **einzelnen Strecken** (3.4) zusammen setzen.

Eine Beschreibung einer Datenbasis enthält eine oder mehrere Karten. Eine Karte hat einen Namen und repräsentiert die Datenbasis für einen Versuch. Beim Start der Simulation öffnet der Versuchsleiter zunächst eine Datei mit der Datenbasis-Beschreibung und selektiert dann eine darin enthaltene Karte.

Jede Karte besteht aus Modulen. Ein **Modul kapselt inhaltlich abgeschlossene Elemente eines Versuchs**. Beispielsweise werden einzelne Situationen durch Module repräsentiert. Die Module einer Karte werden miteinander verknüpft. Dadurch wird der Ablauf eines Versuchs festgelegt.

Es kann vorkommen, dass zwei Karten einer Beschreibung dieselben Module verwenden. Eine Karte könnte beispielsweise zum Training bestimmter Situationen benutzt werden, während der eigentliche Versuch in einer anderen Karte enthalten ist. Deswegen sind die Module einer Karte Instanzen von sog. Modul-Schablonen, die außerhalb der Karten einer Beschreibung definiert werden. Eine Modul-Schablone kann in einer Karte mehrfach instanziert werden,

was einer Wiederholung einer Situation an verschiedenen Stellen des Ablaufs entspricht. Der Mechanismus ähnelt dem in höheren Programmiersprachen: Die Schablonen entsprechen Datentypen, von denen (u.U. mehrfach) Instanzen erzeugt werden.

Eine Datenbasis-Beschreibung hat also folgenden grundlegenden Aufbau:

Grammatik 3.3 Grundlegender Aufbau einer Beschreibung

```
<Datenbasis-Beschreibung> →
    (1...⋆)<Querschnittsprofil (→ 3.21, S.58)>
    (1...⋆)<Modul-Schablone>
    (1...⋆)<Karte>
<Modul-Schablone> →
    ModulSchablone <Name Modul-Schablone>
    {
        <Beschreibung Modul-Schablone (→ 3.6, S.35)>
    }
<Karte> →
    Karte <Name Karte>
    {
        (1...⋆)<Modul-Instanz (→ 3.4, S.30)>
        (0,1)<Modul-Ablauf (→ 3.5, S.31)>
        <Aufsetzpunkte (→ 3.22, S.63)>
    }
```

Auf die Querschnittsprofile wird im Abschnitt 3.4.3 eingegangen, auf die Aufsetzpunkte im Abschnitt 3.4.5.

Eine Instanz einer Modul-Schablone bekommt einen Namen, mit dem sie innerhalb einer Karte adressiert wird.

Grammatik 3.4 Instanzierung von Modul-Schablonen in einer Karte

```
<Modul-Instanz> →
    <Name Modul-Schablone> <Name Modul-Instanz> ;
```

Wenn mehrere Instanzen von Modul-Schablonen in einer Karte erzeugt wurden, dann muss ihre Abfolge spezifiziert werden. Hierzu werden bereits in einer

Modul-Schablone Strecken, die aus dem Modul herausführen, als Anschlussstellen für andere Module festgelegt. Eine solche Anschlussstelle heißt "Port" eines Moduls. Ein Port mit Nummer i einer Modul-Instanz `mod` wird mit `mod.Port<i>` bezeichnet.

Grammatik 3.5 Festlegen des Modul-Ablaufs einer Karte

<Modul-Ablauf> →
    `Ablauf =`
    `{`
        (,)*<Modul-Verknüpfung>*
    `};`
<Modul-Verknüpfung> →
    <Name Modul-Instanz>.<Port> <-> <Name Modul-Instanz>.<Port>
<Port> →
    `Port`*<Nummer>*

Am Beispiel des eingangs dargestellten Parcours lässt sich der geschilderte Aufbau von Datenbasis- Beschreibungen verdeutlichen[2]. Er besteht aus mehreren Fahraufgaben, in die der Testfahrer während des Versuchs gebracht wird. Aus Gründen der Übersichtlichkeit wird der Parcours in diesem Beispiel auf drei Situationen reduziert: Die Einfahrt in den Parcours, die Navigationsaufgabe[3] und die scharfe Kurve[4]. In verschiedenen Trainingsdurchgängen, die der Testfahrer vor dem eigentlichen Versuch absolviert, werden ihm die Situationen vorgestellt. Die Beschreibung enthält daher drei Karten namens `TrainingNavigation`, `TrainingScharfeKurve` und `Versuch`. Wiederum aus Gründen der Übersichtlichkeit sind in der folgenden Beschreibung nur die Karten `TrainingNavigation` und `Versuch` aufgeführt. Die an späterer Stelle vorgestellte Beschreibung der Modul-Schablonen wird durch "..." angedeutet.

[2] Der Situationsbewusstsein-Parcours wird im Abschnitt "Exemplarischer Untersuchungsgegenstand" (S. 18) vorgestellt.
[3] Die Navigationsaufgabe ist auf Seite 19 erklärt.
[4] Die Situation "Scharfe Kurve" ist auf Seite 19 erklärt.

```
ModulSchablone Einfahrt
{ ... }
ModulSchablone Navigation
{ ... }
ModulSchablone ScharfeKurve
{ ... }
Karte TrainingNavigation
{
    Einfahrt einf;
    Navigation nav;
    Ablauf =
    {
        einf.Port2 <-> nav.Port1
    };
    ...
}
Karte Versuch
{
    Einfahrt einf;
    Navigation nav1;
    Navigation nav2;
    ScharfeKurve kurve1;
    ScharfeKurve kurve2;
    Ablauf =
    {
        einf.Port2 <-> nav1.Port1,
        nav1.Port2 <-> kurve1.Port1,
        kurve1.Port2 <-> nav2.Port1,
        nav2.Port2 <-> kurve2.Port1
    };
    ...
}
```

Abbildung 3.2 zeigt den Aufbau der Karte "TrainingNavigation". In Abbildung 3.1 ist die Karte "Versuch" dargestellt.

Durch die Kapselung von inhaltlich abgeschlossenen Teilen einer Datenbasis in den Modulen wird die in Abschnitt 2.3 unter (2) geforderte situationsbezogene Modularisierung gewährleistet. Die dort ebenfalls verlangte Wiederverwend-

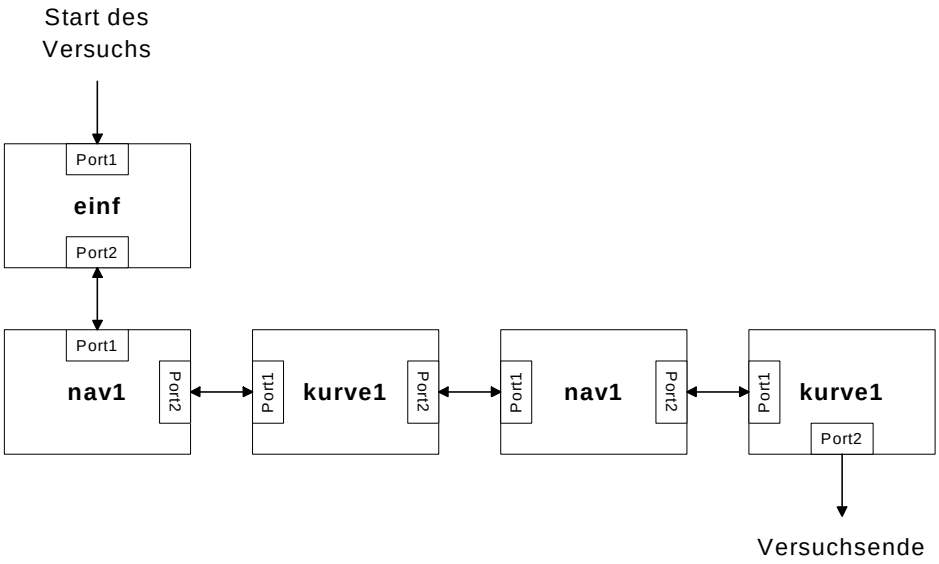

Abbildung 3.1.: Aufbau der Karte `Versuch`

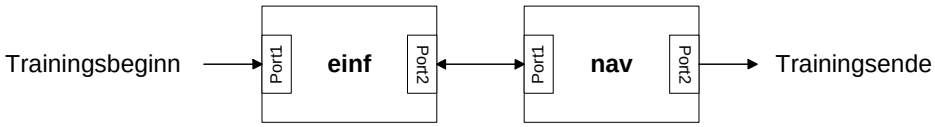

Abbildung 3.2.: Aufbau der Karte `TrainingNavigation`

barkeit der Module wird in dreierlei Hinsicht erfüllt: In einer Karte können mehrere Instanzen derselben Modul-Schablone erzeugt werden. Darüber hinaus können in verschiedenen Karten dieselben Modul-Schablonen verwendet werden. Weil Modul-Schablonen unabhängig von den Karten definiert werden, in denen sie eingesetzt werden, sind sie auch leicht in andere Skript-Dateien einfügbar[5].

Die Aufteilung in Modul-Schablonen und Karten unterstützen die Planung und Organisation von Versuchen.

3.3. Module

3.3.1. Strecken-Schablonen und Instanzen

Durch Modul-Schablonen werden inhaltlich abgeschlossene Teile eines Versuchs repräsentiert. Grundlage hierfür ist ein in der Modul-Schablone definiertes Streckennetz, das sich aus einzelnen Strecken zusammensetzt. Die Strecken werden auch hier zunächst als Schablonen beschrieben, von denen Instanzen erzeugt werden. Mit diesen Instanzen wird das Streckennetz festgelegt. Wie bei den Modul-Schablonen und -Instanzen im vorhergehenden Abschnitt bringt dies den Vorteil, dass einzelne Strecken in einem Modul mehrfach verwendet werden können. Um zu gewährleisten, dass die Beschreibung einer Modul-Schablone in sich abgeschlossen und damit leicht wiederverwendbar ist, müssen die Schablonen für die Strecken *innerhalb* der Modul-Schablone spezifiziert werden. Eine bestimmte Strecken-Schablone kann also nicht in mehreren Modul-Schablonen verwendet werden.

[5] Es ist möglich, Modul-Schablonen in kleine Teilskripten auszulagern. Die Teilskripten werden, ähnlich wie bei Programmiersprachen, durch einen "include"-Befehl von der Hauptdatei aus eingebunden.

Grammatik 3.6 Beschreibung einer Modul-Schablone

<Beschreibung Modul-Schablone> →
id = *<Integer>*;
(1...⋆)*<Strecken-Schablone* (→ 3.7, S.36)*>*
(1...⋆)*<Strecken-Instanz* (→ 3.8, S.36)*>*
(0,1)*<Streckennetz* (→ 3.9, S.37)*>*
<Port-Verfeinerungen (→ 3.11, S.44)*>*
(0,1)*<Events von Strecken-Instanzen* (→ A.4, S.209)*>*

Der Teil *<Events von Strecken-Instanzen* (→ A.4, S.209)*>* der Grammatik wird in Abschnitt 3.4.4 behandelt.

Bei der Beschreibung vergibt der Anwender für die Modul-Schablone eine Kennung (id). Zusammen mit einem Zähler für die Instanzen eines Moduls können damit Abschnitte eines Versuchs eindeutig identifiziert werden. Während eines Versuchs wird die Identifikation der Modul-Instanz, in der sich der Testfahrer gerade befindet, zusammen mit den Messwerten protokolliert. Zur Erleichterung der späteren Auswertung des Versuchs kann die Protokoll-Datei anhand dieser Identifikation in die einzelnen Modul-Schablonen oder -Instanzen aufgeteilt werden (vgl. Anforderung (2) in 2.3).

Es gibt **zwei Typen von Strecken** und entsprechend auch zwei Typen von Strecken-Schablonen:

Straße: Straßen mit einheitlichem Querschnittsprofil ohne Abzweigungen. Die Strecken-Schablonen für diesen Typ heißen `STRSchablone`.

Verkehrsknoten: Kreuzung, Kreisverkehre und Übergänge zwischen Straßen mit unterschiedlichem Querschnittsprofil (z.B. Auffahrten auf Autobahnen). Schablonen dieses Typs heißen `VKSchablone`.

Grammatik 3.7 Strecken-Schablonen

```
<Strecken-Schablone> →
    <Schablone Straße> |
    <Schablone Verkehrsknoten>
<Schablone Straße> →
    STRSchablone <Name>
    {
        <Beschreibung Straße (→ 3.12, S.45)>
    }
<Schablone Verkehrsknoten> →
    VKSchablone <Name>
    {
        <Beschreibung Verkehrsknoten (→ 3.15, S.50)>
    }
```

Die Instanzierung einer Strecken-Schablone hat folgende Form:

Grammatik 3.8 Instanzierung einer Strecken-Schablone

```
<Strecken-Instanz> →
    <Name Strecken-Schablone> <Name Strecken-Instanz> ;
```

3.3.2. Streckennetz

Die **Verknüpfung von Strecken**[6] zu einem Streckennetz erfolgt unabhängig vom Lage- und Höhenplan. Betrachtet man also Lage- und Höhenplan von Strecken als deren geometrische Repräsentation, dann wird das Streckennetz **auf einer topologischen Ebene** beschrieben.

Jede Strecke hat Ports, über die sie mit anderen Strecken verbunden werden kann. Straßen haben zwei Ports (`Port1` und `Port2`), die dem Anfang bzw. dem Ende der Straße zugeordnet sind. Verkehrsknoten sind rechteckige Gebiete, die auf jeder Seite (oben, unten, rechts, links) beliebig viele Ports haben können. Mit `Port<i>` wird der i-te Port eines Verkehrsknotens bezeichnet.

[6] Mit Strecken sind im folgenden Strecken-Instanzen gemeint

Grammatik 3.9 Streckennetz eines Moduls

```
<Streckennetz> →
    Streckennetz =
    {
        (,)<Strecken-Verknüpfung>
    };
<Strecken-Verknüpfung> →
    <einfache Strecken-Verknüpfung> |
    <bedingte Strecken-Verknüpfung (→ 3.10, S.42)>
<einfache Strecken-Verknüpfung> →
    <Name Strecken-Inst.>.<Port> <-> <Name Strecken-Inst.>.<Port>
```

Einfache Verknüpfungen

Bereits mit Verknüpfungen der Form

<einfache Strecken-Verknüpfung>

können zwei Typen von Streckennetzen aufgebaut werden, die in global geometrisch konsistenten Datenbasen nicht zu realisieren sind.

Strecken können zu Schleifen verknüpft werden, obwohl sie geometrisch, d.h. aufgrund ihres Lage- und Höhenplans, keine Schleifen bilden. Solche Schleifen werden als **topologische Schleifen** bezeichnet. Die einfachste Form besteht aus zwei Strecken, im Beispiel `s1` und `s2` genannt, deren Lageplan jeweils eine Gerade ist. Die Geraden sind folgendermaßen verknüpft:

```
Streckennetz =
{
    s1.Port2 <-> s2.Port1,
    s2.Port2 <-> s1.Port1
};
```

Das Netzwerk ist in Abb. 3.3 dargestellt. Bei der **Auflösung der topologischen Schleife** während der Simulation wird ausgenützt, dass ein Fahrer eine eingeschränkte Sichtweite hat.
Befindet er sich (von `s1` kommend) auf `s2`, dann ist an einem bestimmten Punkt `s1` nicht mehr im Rückspiegel erkennbar. Die Strecke `s1` kann von da an

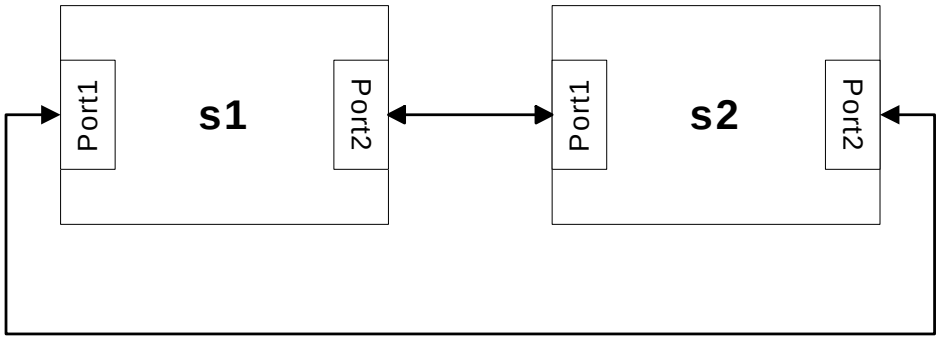

Abbildung 3.3.: Einfachste Form einer topologischen Schleife

für ihn unbemerkt an das Ende von `s2` angeschlossen werden. Dies geschieht, sobald der Port `s2.Port2` für den Fahrer sichtbar wird.

Der skizzierte Algorithmus funktioniert nur unter einer wichtigen Bedingung: Die Gerade `s2` muss länger sein, als der Fahrer sehen kann. Nur dann ist gewährleistet, dass `s1` aus der Sichtbarkeit des Fahrers herausfällt. Damit der Anwender leicht sehen kann, ob er topologische Schleifen aus Strecken konstruieren darf, in deren Lageplan auch Kurven vorkommen, wird eine **spezielle Definition des Sichtbarkeitsbereichs** herangezogen: Eine Ausdehnung $S_{p,max}$ des Sichtbarkeitsbereichs bedeutet, dass der Fahrer von seiner aktuellen Position aus $S_{p,max}$ Meter nach vorne und nach hinten *entlang der Fahrbahn* sehen kann.

Der zweite, global geometrisch inkonsistente Typ von Streckennetzwerken, der mittels einfachen Strecken-Verknüpfungen aufgebaut werden kann, heißt **topologischer Verkehrsknoten**. Ein solcher Knoten liegt vor, wenn ein Port einer Strecke mit mehreren Ports anderer Strecken verknüpft wird.

Ein topologischer Verkehrsknoten wird beispielsweise in der **Navigationsaufgabe** verwendet[7]. Der Fahrer muss aus einer Reihe von uneindeutigen Navigationshinweisen den letzten befolgen und an der T-Kreuzung links abbiegen. Wenn er fälschlicherweise gerade durch die Kreuzung fährt, soll er unmerklich wieder an die Kreuzung zurück geführt werden und die Aufgabe wiederholen. Das Streckennetz hierzu wird wie folgt aufgebaut (vgl. Abb.

[7] Die Navigationsaufgabe ist auf Seite 19) erklärt.

3.4): Der Fahrer kommt auf einer Straße s0 aus einer Situation, die der Navigationsaufgabe vorausging. Es folgt eine Straße s1, auf der der Fahrer zu der T-Kreuzung k gelangt. Er erhält die uneindeutigen Navigations-Hinweise auf s1. Wenn der Fahrer den letzten Hinweis richtig erkennt, biegt er in k links ab und gelangt auf die Straße s2, die ihn zur nächsten Situation führt. Wenn er gerade weiterfährt, kommt er auf die Straße s3 und von der aus wieder auf s1.

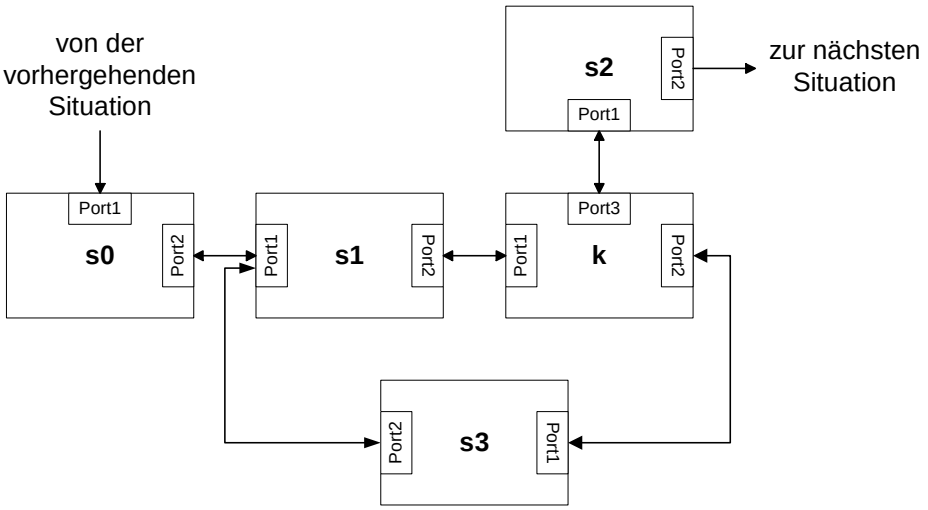

Abbildung 3.4.: Topologischer Verkehrsknoten an s1.Port1

Abbildung 3.4 zeigt das Streckennetz, das durch folgenden Skript-Ausschnitt definiert wird:

```
Streckennetz =
{
    s0.Port2 <-> s1.Port1,
    s1.Port2 <-> k.Port1,
    k.Port2 <-> s3.Port1,
    k.Port3 <-> s2.Port1,
    s3.Port2 <-> s1.Port1
};
```

Der topologische Verkehrsknoten befindet sich in Port1 der Straße s1. Zwei Ports, s0.Port2 und s3.Port2, sind an ihn angeschlossen, obwohl sich rein geometrisch keine Kreuzung dort befindet.

Wie das Beispiel der Navigationsaufgabe zeigt, unterstützen topologische Verkehrsknoten die in Abschnitt 2.2 geforderte Kontrolle des Versuchsablaufs. Zur **Auflösung von topologischen Verkehrsknoten** während der Simulation muss für den Port, der mehrfach verbunden ist, eine eindeutige Verknüpfung selektiert werden. Hierzu wird, sobald der Port in den Sichtbarkeitsbereich des Fahrers gelangt, immer diejenige Verknüpfung gewählt, die den Port mit der Strecke verbindet, auf der sich der Fahrer gerade befindet. Im Beispiel wäre dies, von `s0` kommend, die Verknüpfung

```
s0.Port2 <-> s1.Port1
```

Nähert sich der Fahrer dem topologischen Verkehrsknoten auf `s3`, dann wird die Verknüpfung

```
s3.Port2 <-> s1.Port1
```

ausgewählt. Wenn die Strecke `s1` von `k` aus befahren wird und der Port `s1.Port1` sichtbar wird, dann wird diejenige Verknüpfung selektiert, die gültig war, als der Fahrer zuletzt von `s0` oder `s3` gekommen ist. War dies noch nie der Fall, dann wird die Verknüpfung gewählt, die bei der Beschreibung des Ablaufs zuerst angegeben wurde.

Auch die Auflösung dieses Verknüpfungs-Typs stellt eine Bedingung an das Straßennetzwerk: Der topologische Verkehrsknoten darf zu einem bestimmten Zeitpunkt der Simulation immer nur von einer zu ihm hinführenden Strecke (in obigem Beispiel die Strecken `s0` oder `s3`) aus sichtbar sein.

Wie das Beispiel der Navigationsaufgabe zeigt, kann durch topologische Verkehrsknoten der Versuchsablauf kontrolliert werden, obwohl der Fahrer seine Route frei wählen kann (vgl. Anforderung 2.2, S. 23).

Bedingte Verknüpfungen

Wie im Abschnitt 2.1 unter (2) gefordert, muss es zur Generierung von reproduzierbaren Situationen möglich sein, dass sich das **Straßennetzwerk während der Simulation abhängig vom Zustand des Fahrers oder dem einzelner Softwaremodule der Fahrsimulation ändert**. Als Beispiel wurde die Situation "scharfe Kurve" aufgeführt, bei der der Fahrer erst ein vorausfahrendes Fahrzeug erreichen muss, bevor er diesem durch eine enge Kurve folgt.

Ein Softwaremodul hat einen bestimmten Zustand, wenn eine an die Werte seiner Ein- und Ausgänge[8] gestellte Bedingung zutrifft.

Bedingte Verknüpfungen werden beispielsweise in der Fahraufgabe "Scharfe Kurve" eingesetzt[9]. Die enge Kurve soll erst auftauchen, wenn der Fahrer höchstens 50 Meter vom Führungsfahrzeug entfernt ist. Der Abstand des Fahrers von den umliegenden Fahrzeugen wird im Modul "`traffic`", der Verkehrssimulation, berechnet. Dort ist der Abstand zum nächsten vorausfahrenden Fahrzeug, das sich auf der selben Spur wie der Fahrer befindet, über den Ausgang `traffic.DistSameAheadNext` zugänglich. Die enge Kurve soll also im Zustand `traffic.DistSameAheadNext <= 50` der Verkehrssimulation auftreten. Die im Beispiel benutzte Notation

<Softwaremodul Name>.<Ein-/Ausgang>

für die Parameter von Softwaremodulen wird im Folgenden beibehalten.

Der Zustand des Fahrers wird von Softwaremodulen gemessen oder errechnet, die am Fahrer erhobene physiologische Maße verarbeiten. Im IZVW-Fahrsimulator stehen hierzu u.a. Puls, EMG und Lidschlussverhalten zur Verfügung. Der Zustand des Fahrers beeinflusst folglich den Zustand dieser Softwaremodule, der wiederum, wie oben dargestellt, als Bedingung an Parameter beschrieben werden kann.

Der Streckenverlauf kann auch vom Versuchsleiter während der Simulation interaktiv beeinflusst werden: Jede Eingabe, die er beispielsweise mittels einer grafischen Benutzeroberfläche macht, wirkt sich auf die Ausgänge und damit den Zustand spezieller Softwaremodule aus.

Ein Streckennetz, das sich während der Simulation auf die oben beschriebene Art verändert, wird in Form von bedingten Verknüpfungen festgelegt:

[8] Ein- und Ausgänge werden zusammenfassend auch als Parameter des Softwaremoduls bezeichnet.

[9] Die Situation "Scharfe Kurve" ist auf Seite 19 erklärt.

Grammatik 3.10 Bedingte Verknüpfungen

```
<bedingte Strecken-Verknüpfung> →
    <Name Strecken-Inst.>.<Port> <->
        (
            (,)<Alternative>,
            <Default Alternative>
        )
<Alternative> →
    <Bedingung> : <Name Strecken-Inst.>.<Port>
<Default Alternative> →
    default : <Name Strecken-Inst.>.<Port>
```

Die rechte Seite einer bedingten Verknüpfung besteht aus einer Menge. Die Menge enthält die alternativen Anschluss-Strecken. Jedes Element der Menge besteht aus einer Bedingung und der Strecke, die angeschlossen wird, wenn die Bedingung erfüllt ist. Bei den Bedingungen handelt es sich um prädikatenlogische Ausdrücke über die Werte von Ein- und Ausgängen von Softwaremodulen. Die von den Bedingungen gebildete Fallunterscheidung muss disjunkt und endlich sein. Zwingend ist die Angabe einer sog. Default-Alternative. Sie wird selektiert, wenn keine der formulierten Bedingungen zutrifft. Zusammen mit der Default-Alternative ist die Fallunterscheidung vollständig.

Die Fahraufgabe "scharfe Kurve" illustriert die Beschreibung von bedingten Verknüpfungen[10]. Das Streckennetz für diese Fahraufgabe setzt sich aus drei Strecken zusammen: Die Straßen `s1` und `s2` bilden eine topologische Schleife, solange der Fahrer weiter als 50 m vom Führungsfahrzeug entfernt ist. Das Führungsfahrzeug wird beim ersten Befahren von `s1` in genügend großer Entfernung vom Testfahrer aufgesetzt, so dass er sein plötzliches Erscheinen nicht bemerkt. Ist der gewünschte Abstand erreicht, gelangt der Fahrer auf die Straße `kurve`, die die scharfe Kurve enthält.

```
Streckennetz =
{
    s1.Port2 <-> s2.Port1,
    s2.Port2 <->(
                    traffic.DistSameAheadNext <= 50 : kurve.Port1,
```

[10] Die Situation "Scharfe Kurve" ist auf Seite 19 erklärt.

```
                    default : s1.Port1
                )
};
```

Wenn die Bedingung von Alternative 1 nicht zutrifft, wird über die Default-Alternative die Schleife erneut durchfahren. Das Streckennetz ist in Abbildung 3.5 dargestellt.

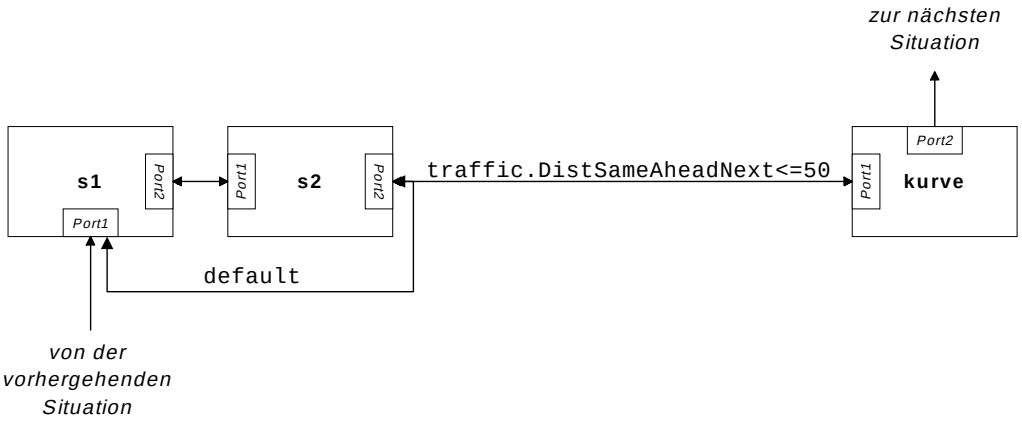

Abbildung 3.5.: Bedingte Verknüpfung in der Situation "scharfe Kurve"

Während der Simulation erfolgt die Auswertung der Bedingungen und die Selektion der Anschluss-Strecke, sobald `s2.Port2` in die Sichtbarkeit des Fahrers gerät.

Voraussetzung für die Verwendung von bedingten Verknüpfungen ist der Zugriff auf die Parameter von Softwaremodulen. Da Simulatoren häufig aus mehreren Rechnern bestehen (wie z.B. der IZVW-Simulator), wird dies dadurch erschwert, dass Parameter von den Computern übertragen werden müssen, auf denen die entsprechenden Module laufen. In der Steuersoftware der IZVW-Simulation kann von jedem Rechner aus auf die Parameter aller Softwaremodule zugegriffen werden (vgl. [22]).

Die in diesem und im vorausgehenden Abschnitt angedeuteten Verfahren zur Erzeugung von Streckennetzen, die innerhalb des Sichtbarkeitsbereich des Fahrers geometrisch konsistent sind, werden in Abschnitt 5.1 detailliert beschrieben.

3.3.3. Ports

In Abschnitt 3.2 wurde bereits gezeigt, wie Modul-Instanzen über ihre Ports zu einem Ablauf verknüpft werden können. Innerhalb einer Karte bilden die Modul-Instanzen also einen Graphen. Bei der Verknüpfung zweier Modul-Instanzen müssen auch ihre Streckennetze verknüpft werden. Das so entstehende zusammengesetzte Streckennetz bildet innerhalb der Karte ebenfalls einen Graphen, der als eine **Verfeinerung des Graphen der Modul-Instanzen** betrachtet werden kann[11]. Diese Verfeinerung wird definiert, indem jedem Port eines Moduls ein Port einer Strecken-Instanz zugewiesen wird:

Grammatik 3.11 Verfeinerung von Modul-Ports

```
<Port-Verfeinerungen> →
    PortVerfeinerungen =
    {
        (,)<Port-Verfeinerung>
    };
<Port-Verfeinerung> →
    <Port> <-> <Name Strecken-Inst.>.<Port>
```

Ein als Verfeinerung verwendeter Strecken-Port muss "frei" sein, d.h. er darf nicht mit anderen Strecken-Ports des Moduls verbunden sein.

Da die Verknüpfung von Modul-Instanzen im Grunde eine Verknüpfung von Strecken darstellt, können topologische Schleifen und Verkehrsknoten auch bei einem Ablauf von Modul-Instanzen eingesetzt werden. Beispielsweise könnte das Streckennetz eines ganzen Parcours, bestehend aus den Streckennetzen mehrerer Situations-Module, eine Schleife bilden. Durch topologische Verkehrsknoten kann der Ablauf eines Parcours, abhängig von der Route des Testfahrers in einem bestimmten Modul, in unterschiedliche Teile aufgespaltet werden. Alle Teile des Ablaufs werden mittels eines topologischen Verkehrsknotens in einem Port *eines* Moduls wieder zusammengeführt.

[11] Diese Sichtweise bildet die Grundlage der in Kapitel 4 vorgestellten Modellierung.

3.4. Strecken

Strecken sind **Instanzen von Strecken-Schablonen**, die wiederum innerhalb einer Modul-Schablone definiert werden. Es gibt zwei Arten von Strecken (und damit auch Strecken-Schablonen): Straßen und Verkehrsknoten. Dieser Abschnitt stellt die Beschreibung von Lage- und Höhenplan sowie Querschnittsprofilen von Strecken-Schablonen vor.

3.4.1. Straßen

Die Beschreibung einer Straße besteht aus folgenden Teilen[12]:

Grammatik 3.12 Beschreibung einer Straße

<Beschreibung Straße> →
    `id =` *<Integer>*;
    `Querschnittsprofil =` *<Name Querschnittsprofil>*;
    <Beschreibung Lageplan (→ 3.13, S.47)*>*
    <Beschreibung Höhenplan>
    $(0,1)$ *<Events Schablone Straße* (→ A.1, S.207)*>*

Wie die Modul-Schablonen werden Schablonen von Straßen vom Anwender mit einer Kennung (`id`) versehen. Wenn der Testfahrer eine Strecken-Instanz befährt, dann wird die Kennung der zu dieser Instanz gehörenden Schablone sowie ein Instanzzähler mit protokolliert. Dadurch können bei der späteren Auswertung eines Versuchs in den aufgezeichneten Daten Teile des Streckennetzes leicht identifiziert werden (vgl. Anforderung (2) in Abschnitt 2.3).

Bei Straßen ändert sich das Querschnittsprofil innerhalb einer Schablone nicht. Es wird daher für die ganze Schablone mit der Zuweisung `Querschnittsprofil = ...` festgelegt. Die Beschreibung von Querschnittsprofilen wird in Abschnitt 3.4.3 erläutert. Dort werden die definierten Querschnittsprofile mit einem Namen versehen, über den sie in der Strecken-Schablone adressiert werden.

[12] Auf den Teil *<Events Schablone Straße* (→ A.1, S.207)*>* der Grammatik wird in Abschnitt 3.4.4 eingegangen.

Lageplan

Der Lageplan legt den **Verlauf der Spuren in der Ebene** fest. Aufgrund des konstanten Querschnitts verlaufen bei einer Straße alle Spuren parallel zueinander. Es reicht daher, den Verlauf der Straßenmitte zu definieren. Hieraus und aus den im Querschnitt definierten Spurbreiten kann der Verlauf der einzelnen Spuren abgeleitet werden.

Die Elemente des Lageplans sind in den RAS-L ([36]) vorgegeben:

- Geraden: Abschnitte mit einer gerade verlaufenden Fahrbahn. Eine Gerade ist durch ihre Länge definiert und hat die Krümmung 0.
- Kreisbögen: Abschnitte, die mit einem konstanten Lenkradeinschlag durchfahren werden können. Kreisbögen werden über ihre Länge und ihren Radius beschrieben. Dabei entspricht ein positiver Radius einer Krümmung nach rechts, ein negativer Radius einer Linkskrümmung. Die Größe der Krümmung ist der inverse Radius. Die Krümmung eines Kreisbogens ist also konstant.
- Übergangsbögen: Wäre im Lageplan ein Kreisbogen direkt an ein Geradenstück angeschlossen, dann würde sich die Krümmung sprunghaft von 0 (Geradenstück) auf eine Krümmung $\neq 0$ im Kreisbogen ändern. Für eine korrekte Spurhaltung müsste in diesem Fall der Fahrer den Lenkwinkel sprunghaft ändern, was in der Realität nicht möglich ist. Deshalb schreiben die RAS-L zwischen Streckenstücken mit unterschiedlicher konstanter Krümmung sog. Übergangsbögen vor. Diese haben die Aufgabe, die unterschiedlichen Krümmungen langsam ineinander zu überführen. Übergangsbögen werden über ihre Länge sowie die Krümmung am Anfang und am Ende parametrisiert.

Für die Wahl der oben genannten Parameter geben die RAS-L aus Sicherheitsgründen genaue Richtlinien. Die in Abschnitt 2.3 unter (3) formulierte Anforderung verlangt, dass der Anwender den Lageplan auf zwei Arten beschreiben kann:

1. Die detaillierte Angabe aller Parameter der Elemente des Lageplans
2. Die automatische und RAS-L konforme Berechnung einiger dieser Parameter

Der Lageplan wird als eine Folge von Geraden und Kurven spezifiziert. Eine Kurve fasst dabei drei Elemente zusammen: Sie beginnt mit einem Übergangsbogen, der von der Krümmung 0 zu der eines Kreisbogens überführt. Ein abschließender Übergangsbogen blendet die Krümmung des Kreisbogens wieder auf die Krümmung 0 aus.

Grammatik 3.13 Lageplan einer Straße

```
<Beschreibung Lageplan> →
    Lageplan =
    {
        (,)<Lageplan-Element>
    };
<Lageplan-Element> →
    ( <Gerade> ) |
    ( <Kurve> )
<Gerade> →
    Gerade, <Länge>
<Kurve> →
    Kurve, <Länge>, <Radius>, <Anteil ÜB1>, <Anteil ÜB2> |
    Kurve, <Länge>, <Radius>
```

Für die Beschreibung einer Kurve gibt es zwei Alternativen: Bei der ersten spezifiziert der Anwender die gesamte Länge der Kurve, den Radius des Kreisbogens sowie die Längen-Anteile der beiden Übergangsbögen an der Gesamtlänge der Kurve. Bei dieser Art der Definition hat der Anwender die Möglichkeit, die RAS-L absichtlich zu verletzen. Er kann beispielsweise eine Kurve ohne Übergangsbögen konstruieren, indem der die beiden Längenanteile auf 0 setzt. Bei der Umwandlung der Beschreibung in ein Modell der Datenbasis wird er bei solchen Übertretungen der RAS-L lediglich gewarnt. Bei der zweiten Alternative der Kurvendefinition entfallen die Längenanteile der Übergangsbögen. Sie werden automatisch und konform mit den RAS-L berechnet (vgl. Abschnitt 4.4.4). In beiden Fällen kann es vorkommen, dass der Anwender andere Richtlinien der RAS-L verletzt, z.B. solche, die die Radien zweier aufeinanderfolgender Kurven betreffen. Auch in diesen Fällen erhält er bei der Generierung des Modells Warnungen. Die Ausgabe von Warnungen wird in Abschnitt 4.4.4 erläutert.

Am Beispiel **Scharfe Kurve** soll die Beschreibung von Lageplänen ver-

anschaulicht werden[13]. Das auf Seite 41 vorgestellte Streckennetz dieser Situation besteht aus drei Straßen: Zwei Straßen, s1 und s2, auf denen der Fahrer fährt, bis er einen bestimmten Abstand zu einem Führungsfahrzeug erreicht, und eine Straße kurve, die die eigentliche scharfe Kurve enthält. Da s1 und s2 eine topologische Schleife bilden, solange sich der Fahrer an das Führungsfahrzeug annähert, muss s2, wie in Abschnitt 3.3.2 dargestellt, länger sein, als der Fahrer überblicken kann. Als Größe des Sichtbarkeitsbereichs[14] wird im Folgenden 2000 m angenommen. In der Schablone, von der die Strecke s1 instanziert wird, ist der Lageplan wie folgt definiert:

```
Lageplan =
{
    (Gerade, 500 m);
};
```

Der Lageplan der Strecke s2, ebenfalls beschrieben in der zugehörigen Schablone, sieht so aus:

```
Lageplan =
{
    (Kurve, 400 m, 1000 m),
    (Kurve, 800 m, -1500 m),
    (Gerade, 200 m),
    (Kurve, 500 m, 2000 m),
    (Gerade, 100 m)
};
```

Bei den Kurven sind keine Längen für die Übergangsbögen angegeben. Sie werden automatisch berechnet. Der Lageplan in der Schablone für kurve besteht aus einer 500 m langen Kurve mit sehr geringem Radius:

```
Lageplan =
{
    (Kurve, 500 m, -300 m)
};
```

Auch hier werden die Längen der Übergangsbögen automatisch berechnet.

[13] Die Fahraufgabe "Scharfe Kurve" ist auf Seite 19 erklärt.

[14] Hier ist der modifizierte Sichtbarkeitsbereich gemeint, d.h. die Größe des Sichtbarkeitsbereichs wird entlang der Straße gemessen.

Höhenplan

Im Höhenplan wird **jedem Punkt der Straßenmitte eine Höhe zugeordnet**. Gemäß RAS-L besteht er aus einer Abfolge von zwei Element-Typen:

- Abschnitte mit konstanter Steigung q: Diese Abschnitte sind entweder flach ($q = 0$), oder repräsentieren Steigungen ($q > 0$) bzw. Gefälle ($q < 0$). Sie werden über ihre Länge und den Wert q definiert.
- Ausrundungen: Zwischen zwei aufeinanderfolgenden Abschnitten mit unterschiedlicher konstanter Steigung q_1, q_2 wird durch Ausrundungen überblendet. Diese gibt es in Form von Kuppen ($q_2 < 0 < q_1$), Wannen ($q_1 < 0 < q_2$), unechten Kuppen ($q_1 > q_2$) und unechten Wannen ($q_1 < q_2$). Die Ausrundungen sind Parabel-Kurven, deren Form über den sog. Ausrundungshalbmesser festgelegt wird. Für diesen Parameter gibt es, abhängig vom Straßentyp, in den RAS-L genaue Richtlinien[15].

Gemäß Abschnitt 2.3 (3) hat der Anwender auch bei der Beschreibung des Höhenplans zwei Möglichkeiten:

1. Für jedes Element werden sämtliche Parameter angegeben.
2. Es werden nur die Elemente mit konstanter Steigung festgelegt, die Ausrundungen werden automatisch und RAS-L konform berechnet.

Der Struktur der Beschreibung folgt der des Lageplans:

Grammatik 3.14 Höhenplan einer Straße

```
<Beschreibung Höhenplan> →
    Höhenplan =
    {
        (,)<Höhenplan-Element>
    };
<Höhenplan-Element> →
   ( <Länge>, <Steigung in Prozent>, <Ausrundungshalbmesser> ) |
   ( <Länge>, <Steigung in Prozent> )
```

Der Höhenplan besteht aus einer Abfolge von Elementen, bei denen jedes aus einer Ausrundung am Anfang und einer anschließenden konstanten Steigung

[15] Hierauf wird im Abschnitt 4.4.3 näher eingegangen.

besteht. Als Parameter werden pro Element die Länge, die Steigung, und der Ausrundungshalbmesser angegeben. Der letzte Parameter ist optional: Wird er weggelassen, dann wird der Ausrundungshalbmesser automatisch berechnet[16]. Die Gesamtlänge des Höhenplans muss der des Lageplans entsprechen. Damit sich die Steigungen aneinanderschließender Strecken nicht aufsummieren, ist im ersten und im letzten Element des Höhenplans nur die Steigung 0 zulässig. Dadurch wird beim ersten Element keine Ausrundung benötigt, der Ausrundungshalbmesser muss deswegen mit 0 angegeben werden. Bei Verletzung dieser Bedingungen, der RAS-L, sowie der Gleichheit von Lageplan-Länge und Höhenplan-Länge wird der Anwender gewarnt.

3.4.2. Verkehrsknoten

Verkehrsknoten repräsentieren **komplexe Teile eines Straßennetzwerks** (z.B. Kreuzungen und Kreisverkehre) sowie Übergänge zwischen Straßen mit unterschiedlichem Querschnittsprofil (z.B. Autobahnauffahrten). Sie sind als rechteckige Gebiete modelliert. Die Spuren von Verkehrsknoten müssen nicht, wie bei Straßen, parallel verlaufen. In ihrem Verlauf kann sich die Breite sowie die relative Höhe zu den anderen Spuren verändern. Dadurch wird die Beschreibung deutlich aufwändiger als die von Straßen. Der Anwender wird aber an mehreren Stellen unterstützt: Die Beschreibungssprache bietet ihm die Möglichkeit, die Spuren des Verkehrsknotens nach logischen Gesichtspunkten zu gruppieren. Ihr exakter Verlauf kann aus sehr wenigen Informationen automatisch bestimmt werden.

Grammatik 3.15 Beschreibung eines Verkehrsknotens

<Beschreibung Verkehrsknoten> →
    id = *<Integer>*;
    Größe = (*<Breite>*, *<Länge>*);
    $(1 \ldots \star)$*<Verkehrsknoten Port* (→ 3.16, S.51)*>*
    <Spur-Informationen (→ 3.17, S.53)*>*
    <Spur-Verknüpfungen (→ 3.20, S.57)*>*
    $(0,1)$*<Events Schablone Verkehrsknoten* (→ A.3, S.208)*>*

Der Teil *<Events Schablone Verkehrsknoten* (→ A.3, S.208)*>* der Grammatik

[16] Dies entspricht dem Weglassen der Längenanteile von Übergangsbögen beim Lageplan.

wird in Abschnitt 3.4.4 behandelt. Wie bei den Modul-Schablonen und Schablonen von Straßen dient auch hier die Zuweisung `id = ...` der Kennzeichnung der Schablone eines Verkehrsknotens. Die Ausmaße des zugrundeliegenden Rechtecks werden in `Größe = ...` festgelegt.

Ports

Die Anschlüsse (auch hier "Ports" genannt), über die ein Verkehrsknoten mit Straßen verknüpft werden kann, befinden sich auf den Seiten `links`, `rechts`, `unten` und `oben` des Rechtecks. Ihnen wird ein Querschnittsprofil zugeordnet und sie können entlang der Seite, auf der sie liegen, verschoben werden:

Grammatik 3.16 Port eines Verkehrsknotens

```
<Verkehrsknoten Port> →
    <Port>.Seite = <Name Seite>;
    <Port>.Querschnittsprofil = <Name Querschnittsprofil>;
    <Port>.Position = <Verschiebung zur Seitennmitte>;
<Port> →
    Port<Nummer>
<Name Seite> →
    oben |
    unten |
    links |
    rechts
```

Mit

`Port<Nummer>`

ist gemeint, dass die Ports von `1` beginnend durchnumeriert werden. Für Ports auf den Seiten `oben` und `unten` entspricht eine positive Position einer Verschiebung von der Seitenmitte aus nach rechts. Auf den Seiten `rechts` und `links` sind Ports mit positiver Position gegenüber der Seitenmitte nach oben verschoben. Die Spuren eines Querschnitts werden, von außerhalb des Verkehrsknotens auf den Port blickend, von links nach rechts durchnumeriert.

Die Ports einer einfachen T-Kreuzung sind in Abbildung 3.6 eingezeichnet.

Spur-Informationen

Die Spuren im Inneren des Rechtecks, das dem Verkehrsknoten zugrunde liegt, werden in sog. **Spurzellen** gruppiert. Sie teilen die Spuren eines Verkehrsknotens in **logische Bereiche** ein. Beispielsweise wird die T-Kreuzung in Abbildung 3.6 in vier Zellen aufgeteilt: Zelle 1, 2 und 4 enthalten Spuren, die von den Ports an den Rändern des Rechtecks zur eigentlichen Kreuzung heranführen. Die Spuren des inneren Bereichs der Kreuzung sind in Zelle 3 gruppiert. Die Spurzellen sind in der Abbildung schematisch als Rechtecke eingezeichnet. Sie besitzen keine Form im geometrischen Sinne, da sie lediglich Gruppen von Spuren repräsentieren.

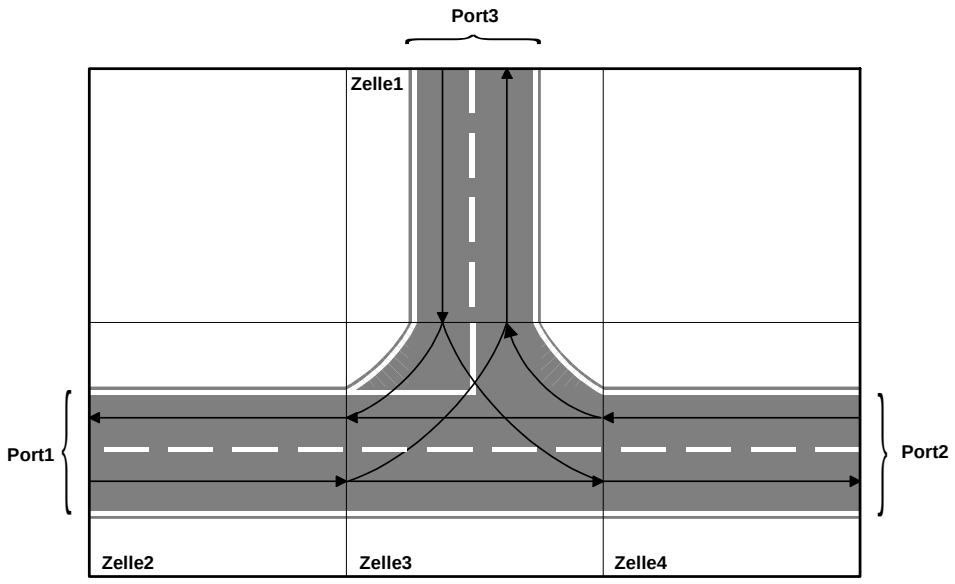

Abbildung 3.6.: T-Kreuzung

Zur Definition einer Spur werden folgende Informationen benötigt:

- Verlauf: Eine Spur kann ein Geradenstück oder ein Bogen zwischen zwei Geradenstücken sein. Da auf diese Art keine Spurzellen übergreifende Kreise (wie sie bei Kreisverkehren vorkommen) beschrieben werden können, gibt es zusätzlich die Möglichkeit, Verläufe außerhalb von Spurzellen zu definieren. Den Spuren in den Zellen werden Segmente davon zugewiesen. Solche übergeordneten Spurverläufe werden im folgenden Super-Spuren genannt.

- Breite: Die Breite von Spuren eines Verkehrsknotens kann sich in ihrem Verlauf ändern. Nur dadurch können Verkehrsknoten als Übergänge zwischen Straßen mit unterschiedlichem Querschnittsprofil dienen.
- Relative Höhe: Verkehrsknoten sind flach. Für Beginn und Ende einzelner Spuren können jedoch Höhen relativ zur Ebene, in der das Rechteck des Verkehrsknotens liegt, angegeben werden.
- Richtungsinformationen: Mit dieser Information navigiert der automatische Verkehr durch Knotenpunkte.

Nach der Definition von übergeordneten Spurverläufen erfolgt die Beschreibung der einzelnen Spuren innerhalb von Spurzellen:

Grammatik 3.17 Spur-Informationen von Verkehrsknoten

```
<Spur-Informationen> →
    (⋆)<Super-Spur>
    (1...⋆)<Spurzelle (→ 3.18, S.54)>
<Super-Spur> →
    <Kreis> |
    <...>
<Kreis> →
    Kreis <Name Super-Spur>
    {
        Radius = <Radius>;
        Mittelpunkt = (<x>, <y>);
    }
```

Als Super-Spuren sind bisher nur Kreise implementiert. Die Beschreibung von Verkehrsknoten kann jedoch an der durch "<...>" gekennzeichneten Stelle in obiger Grammatik leicht um weitere Typen von Super-Spuren (wie z.B. Spline-Kurven) erweitert werden. Hierauf wird auch in Abschnitt 4.5 eingegangen.

Spurzellen erhalten einen Namen und kapseln einzelne Spuren, für die ebenfalls ein Name vergeben wird. Der Name von Spuren muss nur innerhalb ihrer Spurzelle eindeutig sein. Alle Spurtypen (Gerade, Bogen und Segment einer Super-Spur) haben Parameter wie Breite, Höhe etc. Zusätzlich dazu muss

bei den Segmenten der Ausschnitt der Super-Spur spezifiziert werden. Zu diesem Zweck wird in einem Tupel `Abschnitt` die Super-Spur über ihren Namen (wie in Grammatik 3.17 vergeben) ausgewählt. Die beiden Tupel-Elemente

<von>, <bis>

geben die Strecke auf der Super-Spur an, die durch das Segment repräsentiert wird.

Grammatik 3.18 Spurzellen von Verkehrsknoten

<Spurzelle> →
    SpurZelle *<Name Spurzelle>*
    {
        $(1 \ldots \star)$ *<Spur>*
    }
<Spur> →
    Gerade *<Name Spur>*
    {
        <Spur-Parameter (→ 3.19, S.55)*>*;
    } |
    Bogen *<Name Spur>*
    {
        <Spur-Parameter (→ 3.19, S.55)*>*;
    } |
    Segment *<Name Spur>*
    {
        Abschnitt = (*<Name Super-Spur>*, *<von>*, *<bis>*);
        <Spur-Parameter (→ 3.19, S.55)*>*;
    }

Aus obiger Grammatik ist zu entnehmen, dass jede Spur einen durch geschweifte Klammern begrenzten Block hat, in dem ihre Parameter aufgeführt werden. Durch diese Syntax lässt sich die Beschreibung einer Spur leicht um weitere Angaben innerhalb ihres Parameter-Blocks erweitern. Folgende Spur- Parameter müssen jedoch mindestens angegeben werden:

Grammatik 3.19 Spur-Parameter

```
<Spur-Parameter> →
    Beginn = <Spur-Beginn>;
    (0,1)<Spurbreite Beginn>
    (0,1)<Spurhöhe Beginn>
    Richtung = { (,)<Richtung> };
<Spur-Beginn> →
    (<x>, <y>) |
    <Port (→ 3.16, S.51)>.Spur<Spurindex>
<Spurbreite Beginn> →
    BreiteBeginn = <positive reelle Zahl>;
<Spurhöhe Beginn> →
    HöheBeginn = <positive reelle Zahl>;
<Richtung> →
    geradeaus |
    links |
    rechts
```

Unter der Voraussetzung, dass eine Spur eines Verkehrsknotens mit ihrem Ende stets an den Beginn einer anderen Spur oder an einen Port anschließt[17], werden durch die Definition aller Punkte, an denen Spuren beginnen, automatisch auch alle Endpunkte festgelegt. Geht eine Spur nicht von einem Port aus, dann wird ihr Beginn als Punkt in einem Koordinatensystem festgelegt, dessen Ursprung die Mitte des zugrundeliegenden Rechtecks des Verkehrsknotens ist (s. Abb. 4.26, S. 118). Dem Beginn von Spuren, die von einem Port ausgehen, wird ein Spurindex im Querschnitt[18] des Ports zugeordnet. Aus den Spurbreiten des Querschnitts und der Position des Ports können die Koordinaten des Spurbeginns berechnet werden.

Mit zwei weiteren Voraussetzungen reichen Spurbeginn und -ende bereits aus, den Verlauf von Geraden und Bögen automatisch zu bestimmen:

1. Spuren beginnen oder enden in einem Port immer senkrecht zur jeweiligen Seite.
2. Zwischen zwei Bögen verläuft stets eine Gerade oder ein Segment einer Superspur.

[17] Die Verknüpfung von Spuren wird im nächsten Abschnitt behandelt.
[18] Die Definition von Querschnitten wird in Abschnitt 3.4.3 erläutert.

Sind diese Bedingungen erfüllt, dann können zunächst die Gleichungen der Geraden aus ihren Beginn- und End-Punkten aufgestellt werden. Die Gleichungen der Segmente von Superspuren sind zu diesem Zeitpunkt bereits bekannt. Damit stehen die Tangentenwinkel in allen Beginn- und End-Punkten von Spuren fest. Die Gleichungen der Bögen erhält man durch Lösen von Interpolationsproblemen, bei denen zwei Punkte einer Kurve und die Tangentenwinkel in diesen Punkten vorgegeben sind. Der Algorithmus zur Bestimmung der Spurverläufe in Verkehrsknoten wird in Abschnitt 4.5.5 dargestellt.

Auch Höhe und Breite einer Spur müssen nur in ihrem Beginn festgelegt werden. Wenn eine Spur von einem Port ausgeht, dann entfällt die Angabe dieser Information. In diesem Fall werden die Werte aus dem Querschnitt des Ports übernommen. Die Höhe (bzw. Breite) am Beginn einer Spur wird in die Höhe (bzw. Breite) am Ende der Spur überführt.

Die unter `Richtung = ...` angegebene Information wird vom automatischen Verkehr zur Navigation durch einen Verkehrsknoten genutzt (vgl. Beispiel auf Seite 188). Die hierbei möglichen Richtungen `geradeaus`, `links` und `rechts` werden vom Beginn der Spur aus gesehen festgelegt.

Spur-Verknüpfungen

Die Spuren einer Zelle werden mit ihrem Beginn bzw. ihrem Ende mit den Spuren anderer Zellen oder mit Spuren der Querschnitte von Ports des Verkehrsknotens verknüpft:

Grammatik 3.20 Verknüpfung von Spuren

```
<Spur-Verknüpfungen> →
        SpurVerknüpfungen =
        {
            (,)<Spur-Verknüpfung>
        };
<Spur-Verknüpfung> →
        <Name Spurzelle>.<Name Spur>.<Spur-Anschluss> <->
                <Name Spurzelle>.<Name Spur>.<Spur-Anschluss>
<Spur-Anschluss> →
        Beginn |
        Ende
```

Jeder dabei entstehende "Weg" durch den Verkehrsknoten muss über mindestens eine Spur mit einem Port verknüpft sein.

Ein Beispiel für die formale Beschreibung der T-Kreuzung aus der Navigationsaufgabe findet sich in Anhang A.1.

3.4.3. Querschnittsprofile

Jeder Schablone einer Straße ist ein Querschnittsprofil zugeordnet (vgl. Grammatik 3.12). Schablonen von Verkehrsknoten speichern in jedem Port ein Querschnittsprofil. Es können nur Strecken verknüpft werden, die bzgl. des Querschnittsprofils gleich sind.

Grammatik 3.21 Querschnittsprofil

```
<Querschnittsprofil> →
    Querschnittsprofil <Name Querschnittsprofil>
    {
        Kategorie = <Bezeichner>;
        ve = <Entwurfsgeschwindigkeit>;
        Spuren =
        (
            (,)<Spurquerschnitt>
        );
    }
<Spurquerschnitt> →
    (<Breite>, <rel. Höhe>, <Fahrtrichtung>)
<Fahrtrichtung> →
    mit | gegen
```

Der Parameter `Kategorie` gibt die Kategoriengruppe an, der die Straße mit diesem Querschnittsprofil zuzuordnen ist. Diese Unterteilung ist durch die RAS vorgegeben. Die Hauptkategorie A, das sind anbaufreie Straßen außerhalb bebauter Gebiete mit maßgebender Verbindungsfunktion, ist in Tabelle 3.1 aufgelistet. Der Tabelle sind auch die entsprechenden Entwurfsgeschwindigkeiten `ve` zu entnehmen.
Neben der in der Tabelle aufgeführten Hauptkategorie A gibt es noch folgende:

Kategoriengruppe B: Anbaufreie Straßen im Vorfeld und innerhalb bebauter Gebiete mit maßgebender Verbindungsfunktion.

Kategoriengruppe C: Angebaute Straßen innerhalb bebauter Gebiete mit maßgebender Verbindungsfunktion.

Kategoriengruppe D: Angebaute Straßen innerhalb bebauter Gebiete mit maßgebender Erschließungsfunktion.

Kategoriengruppe E: Angebaute Straßen innerhalb bebauter Gebiete mit maßgebender Aufenthaltsfunktion.

Die Elemente des Tupels `Spuren` sind wiederum Tupel. Jedes dieser enthaltenen Tupel beschreibt den Querschnitt einer Spur. Die Reihenfolge legt dabei die

Tabelle 3.1.: Hauptkategorie A gem. RAS

Kategorie	**Verkehrsart**	**Höchstgeschw.**	**Anz. Fahrbahnen**	**ve**
AI Fernstraße	Kfz Kfz	keine 100	2 1	120, 100 100, 90
AII überregionale. oder regionale Straße	Kfz Allg.	keine 100	2 1	100, 90 90, 80
AIII zwischen- gemeindliche Straße	Kfz Allg.	100 100	2 1	80, 70 80, 70, 60
AIV flächener- schließende Straße	Allg.	100	1	70, 60
AV untergeordn. Straße	Allg.	100	1	50

Spur-Indizes fest[19] : Die Spuren werden von links nach rechts aufgeführt und, beginnend mit Index 0, ebenso gezählt.

Ein Tupel, das eine einzelne Spur beschreibt, besteht aus folgenden Elementen:

- `Breite`: Die Breite der Spur in Meter.
- `rel. Höhe`: Die relative Höhe der Spur in Meter.
- `Fahrtrichtung`

Für die einzelnen RAS Straßenkategorien gibt es in der RAS-Q (vgl. [37]) Regelquerschnitte, denen der Anwender diese Parameter entnehmen kann.

Sowohl die Strecken in der **Navigationsaufgabe**[20] als auch die in der Fahraufgabe **Scharfe Kurve**[21] sind Bundesstraßen mit je einer Fahrbahn pro Fahrtrichtung. Das Querschnittsprofil dieses Straßentyps ist so definiert:

```
Querschnittsprofil Bundesstraße
 {
     Kategorie = AI;
     ve = 100.0;
     Spuren =
     (
         (3.5, 0.0, gegen),
         (3.5, 0.0, mit)
     );
 }
```

Bei Bundesstraßen handelt es sich um die Kategorie AI aus Tabelle 3.1. Es gibt eine Fahrbahn mit zwei Spuren, eine für jede Richtung. Hieraus ergibt sich die Entwurfsgeschwindigkeit `ve`.

[19] Die Spur-Indizes werden z.B. bei der Beschreibung der Verkehrsknoten verwendet (vgl. Grammatik 3.19).
[20] Die Navigationsaufgabe ist auf Seite 19 erklärt.
[21] Die Situation "Scharfe Kurve" ist auf Seite 19 erklärt.

3.4.4. Strecken-Events

In Abschnitt 2.1 wurde unter (1) beschrieben, wie sich reproduzierbare Situationen durch die Platzierung sog. Strecken-Events erzeugen lassen. Bei Strecken-Events handelt es sich um für den Testfahrer **unsichtbare Markierungen auf den Spuren einer Strecke, durch deren Überfahren Veränderungen in Softwaremodulen der Simulation ausgelöst werden**.

In der Beschreibung einer Datenbasis können die Strecken-Events sowohl in der Schablone einer Strecke (Straße oder Verkehrsknoten) als auch in einer Strecken-Instanz definiert werden. Dadurch ist es möglich, für alle Instanzen einer Strecken-Schablone gemeinsame Strecken-Events festzulegen, aber auch jeder Instanz eigene Events zuzuweisen. Insgesamt wird so die Wiederverwendbarkeit von Strecken-Schablonen verbessert.

In **Schablonen von Straßen** ist jedes Event als Tupel definiert. Die Tupel setzen sich zusammen aus einer Positionierung des Events und Argumenten, die beschreiben, welches Softwaremodul auf welche Weise auf das Event reagieren soll. Ein Event wird auf einer bestimmten Spur, adressiert durch ihren Index gemäß dem Querschnittsprofil, platziert. Auf dieser Spur befindet es sich auf einem Punkt, der vom Beginn aus eine bestimmte Distanz entlang der Spur liegt. Diese Distanz kann entweder direkt in Meter oder als Anteil der gesamten Spurlänge in Prozent angegeben werden. In den Argumenten eines Events ist als erstes festgelegt, auf welches Softwaremodul der Simulation es sich bezieht. Danach folgt eine Liste beliebiger Parameter. Die Parameter sind dabei reelle Zahlen oder Bezeichner, also alphanumerische Zeichenketten. Wird ein Strecken-Event überfahren, dann kann das betroffene Softwaremodule diese Parameterliste von der Datenbasis abfragen und interpretieren. Dieser Mechanismus wird in Abschnitt 5.2.3 erläutert. Die Grammatiken zur Beschreibung von Events in Schablonen von Straßen finden sich im Anhang unter A.1 (S. 207) und A.2 (S. 208).

Bei Events in **Schablonen von Verkehrsknoten** wird zur Positionierung anstatt eines Spurindex der Name einer Spur angegeben. Die Grammatik hierzu ist im Anhang unter A.3 (S. 208) wiedergegeben.

Events von **Strecken-Instanzen** werden, wie in Grammatik 3.6 (S. 35) definiert, in den Modul-Schablonen beschrieben. Auch hier ist jedes Event

ein Tupel. Zusätzlich zu den Events in Strecken-Schablonen hat es als erstes Element den Namen einer Strecken-Instanz. Die restlichen Elemente werden wie bei den Events in den entsprechenden Schablonen spezifiziert. Grammatik A.4 (im Anhang, S. 209) zeigt die Syntax im Detail.

Die Beschreibung von Strecken-Events kann am Beispiel der **Navigationsaufgabe** verdeutlicht werden[22]. Die wechselnden Richtungsinformationen, die das Navigationssystem anzeigt, während sich der Fahrer auf der Strecke `s1` der Kreuzung nähert, werden als Strecken-Events in der zu `s1` gehörenden Schablone `Annäherung` definiert:

```
STRSchablone Annäherung
{
    id = 1234;
    Querschnittsprofil = Bundesstraße;
    Lageplan =
    {
        (Kurve, 300 m, 1000 m),
        (Gerade, 200 m)
    };
    ...
    Events =
    {
        (1, 250 m, navi, links),
        (1, 300 m, navi, rechts),
        (1, 350 m, navi, geradeaus),
        (1, 400 m, navi, rechts),
        (1, 450 m, navi, links),
        (1, 500 m, navi, leer)
    };
};
```

Die Straße hat den Querschnitt einer Bundesstraße, wie er im Beispiel auf Seite 60 angegeben ist. Der Lageplan hat eine Länge von 500 m. Ab Streckenmeter 250 liegen die Events im Abstand von 50 Metern auf Spur 1, also auf der rechten Spur vom Beginn der Strecke aus gesehen. Sie betreffen ein Softwaremodul namens `navi`, von dem das Navigationssystem simuliert wird. Nach jedem

[22] Die Navigationsaufgabe ist auf Seite 19 erklärt. Das Streckennetz für diese Situation ist auf Seite 38 beschrieben.

Simulationsschritt erfragt dieses Modul alle seit dem letzten Schritt überfahrenen Events für `navi`. Als Ergebnis dieser Anfrage bekommt das Modul, je nach überfahrenem Event, die Parameter `links`, `rechts`, `geradeaus` oder `leer` zurück geliefert. Es zeigt daraufhin, zusammen mit einem akustischen Hinweis, den entsprechenden Richtungspfeil oder löscht das Display.

3.4.5. Aufsetzpunkte

Beim Start der Simulation wird der Testfahrer an eine bestimmte Stelle des Straßennetzwerks positioniert. So eine Stelle heißt "Aufsetzpunkt". Bei der Beschreibung einer Karte kann der Anwender mehrere solcher Aufsetzpunkte definieren. Dabei vergibt er Namen, über die er bei jedem Start der Simulation einen Aufsetzpunkt auswählt. Die Beschreibung der Aufsetzpunkte hat folgende Form:

Grammatik 3.22 Aufsetzpunkte

```
<Aufsetzpunkte> →
    Aufsetzpunkte =
    {
        (,)<Aufsetzpunkt>
    };
<Aufsetzpunkt> →
    (
        <Name Aufsetzpunkt>,<Name Modul-Inst.>,
        <Name Straßen-Inst.>, <Spurindex>
    )
```

Der Fahrer kann folglich auf den Beginn einer bestimmten Spur einer Straße aufgesetzt werden.

4. Modellierung

4.1. Notation

Das Modell ist nach den Methoden der objektorientierten Modellierung aufgebaut. Die in dieser Arbeit verwendete Notation wird in diesem Abschnitt vorgestellt und an einem Beispiel erläutert. An verschiedenen Stellen dieses Kapitels werden Zusammenhänge im Modell anhand von UML-Diagrammen verdeutlicht (vgl. [6]). Die dabei verwendeten Diagrammelemente werden im folgenden ebenfalls eingeführt.

Ein Modell besteht aus einer Menge von Objekten. Diese Objekte sind in Typen, auch Klassen genannt, eingeteilt. In einem Modell für Fahrzeuge könnte es beispielsweise die Typen $\boldsymbol{PKW}$ und $\boldsymbol{Motorroller}$ geben. Ein Objekt stellt dann die konkrete Ausprägung eines bestimmten Typs dar.

Abbildung 4.1.: Objekttypen in UML.

So sind die Objekte $VWGolf$ und $BMW520i$ Objekte des Typs $\boldsymbol{PKW}$. Als Schreibweise hierfür wird $VWGolf \triangleright \boldsymbol{PKW}$ bzw. $BMW520i \triangleright \boldsymbol{PKW}$ verwendet. In der UML werden Typen wie in Abb. 4.1 dargestellt.

Die Eigenschaften von Objekten werden durch Datenelemente festgelegt. Die Datenelemente werden in der Definition der Typen aufgezählt, aber erst in den Objekten mit Werten belegt. So haben z.B. Objekte des Typs $\boldsymbol{PKW}$ eine Zahl n_T, die die Anzahl der Türen des PKWs angibt. Auf die Elemente eines bestimmten Objekts wird über die aus der Programmiersprache C++

bekannte Punkt-Notation zugegriffen. Damit können die Elemente auch mit Werten belegt werden, z.B.:

$$\begin{aligned} VWGolf.n_T &= 2 \\ BMW520i.n_T &= 4 \end{aligned}$$

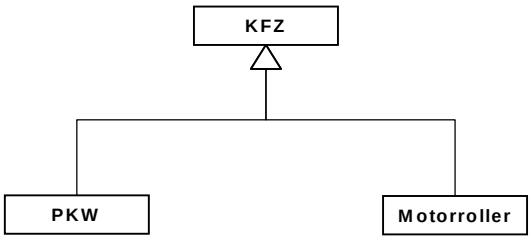

Abbildung 4.2.: Ableitung in UML.

Die unterschiedlichen Objekttypen können in einer Hierarchie organisiert werden. Im Beispiel können die Typen ***PKW*** und ***Motorroller*** zu einem Typ ***KFZ*** abstrahiert werden. Man sagt, die Typen ***PKW*** und ***Motorroller*** sind vom Typ ***KFZ*** abgeleitet und ***KFZ*** ist der Grundtyp von ***PKW*** und von ***Motorroller***. Abb. 4.2 zeigt, wie dieser Sachverhalt in der UML dargestellt wird.

Datenelemente, die in Grundtypen aufgezählt werden, sind auch in den Objekten der abgeleiteten Typen enthalten. Abgeleitete Typen "erben" also die Datenelemente des Grundtyps. Im Beispiel kann dadurch zum Ausdruck gebracht werden, dass alle Kraftfahrzeuge Räder enthalten: Objekte des Typs ***KFZ*** bekommen hierzu eine Zahl n_R und eine Menge $R = \{r_1, \ldots, r_{n_R}\}$ von Rädern. Für Objekte $o \triangleright$ ***PKW*** gilt: $o.n_R = 4$, bei Objekten $o \triangleright$ ***Motorroller*** ist $o.n_R = 2$.

Objekte können als Datenelemente andere Objekte enthalten. So sind die Elemente der Menge $o.R$ von Objekten $o \triangleright$ ***KFZ*** beispielsweise Objekte eines Typs ***Rad***. Man sagt, der Typ ***KFZ*** aggregiert Objekte des Typs ***Rad***. Aggregationen werden in der UML wie in Abb. 4.3 notiert.
Für jede Seite der Aggregation wird dabei eine Kardinalität angegeben: Ein einzelnes Rad ist Element genau eines Kraftfahrzeugs, ein Kraftfahrzeug enthält mindestens zwei Räder. Übliche Kardinalitäten sind:

- n : genau n

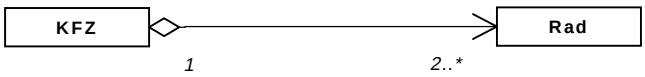

Abbildung 4.3.: Aggregation in UML.

- $\star$: null oder beliebig viele
- $n \ldots \star$: mindestens n
- n_1, n_2: entweder n_1 oder n_2

Der Pfeil auf der rechten Seite der Aggregation in Abb. 4.3 symbolisiert, dass Objekte des Typs ***KFZ*** auf die Elemente von Objekten des Typs ***Rad*** zugreifen können. Umgekehrt ist das nicht möglich, deswegen ist auf der linken Seite kein Pfeil eingezeichnet. Gelegentlich werden die Seiten einer Aggregation mit einem Namen versehen. Dieser Name beschreibt die sog. "Rolle", die ein aggregiertes Objekt im aggregierenden Objekt spielt.

Operationen, die Objekte auf ihren Datenelementen ausführen, werden in den Typen als sog. "Methoden" definiert. Eine solche Methode von Kraftfahrzeugen könnte beispielsweise $StarteMotor()$ sein. Gelegentlich wird die ebenfalls aus der Sprache C++ stammende Schreibweise $\boldsymbol{KFZ} :: StarteMotor()$ verwendet, um zu kennzeichnen, für welchen Objekttyp eine Methode definiert ist. Wie Funktionen und Prozeduren bei Programmiersprachen können Methoden Argumente und Rückgabewerte haben. Der Aufruf einer Methode erfolgt wieder durch die Punkt-Notation "von einem Objekt aus": $o.StarteMotor()$ mit $o \triangleright \boldsymbol{KFZ}$.

Zwar kann bei allen Kraftfahrzeugen der Motor gestartet werden, je nach Typ muss dies aber auf unterschiedliche Weise passieren. Die Implementation von $StarteMotor()$ kann daher für jeden von ***KFZ*** abgeleiteten Typ getrennt angegeben werden. Im Typ ***KFZ*** wird also lediglich die Schnittstelle der Methode $StarteMotor()$ definiert. Solche Methoden heißen "rein virtuell". Je nach Objekttyp von o wird durch $o.StarteMotor()$ die entsprechende Implementation ausgeführt. Für $o_1 \triangleright \boldsymbol{PKW}$ wird in $StarteMotor()$ beispielsweise die Zündung betätigt, für $o_2 \triangleright \boldsymbol{Motorroller}$ das Anlasser-Pedal. Betrachtet man jedoch o_1 und o_2 als Objekte des Grundtyps ***KFZ***, dann führt der Aufruf von $StarteMotor()$ immer zum selben Ergebnis: der Motor läuft. Durch eine Imple-

mentation einer Methode in einem Grundtyp kann ein "Standard-Verhalten" angegeben werden, das verwendet wird, wenn ein abgeleiteter Typ keine eigene Implementation zu Verfügung stellt. Solche Methoden heißen dann nur noch "virtuell", nicht mehr "rein virtuell".

4.2. Topologische Struktur des Modells

4.2.1. Überblick

Die fünf Schichten, aus denen das Modell aufgebaut wird, ergeben sich direkt aus den Informationen, die in der formalen Beschreibung angegeben werden:

Karte: Die oberste Schicht ist die Karte. Sie besteht aus einem einzelnen Objekt vom Typ ***Map***.

Module: Module kapseln inhaltlich abgeschlossene Situationen. Wie in Abschnitt 3.2 dargestellt, wird der Ablauf von Situationen eines Versuchs durch die Verknüpfung von Modulen festgelegt. Deswegen bilden die Module einer Datenbasis einen Graphen MODULES.

Strecken: Es gibt Straßen und Verkehrsknoten (vgl. 3.3.1). Ihre Verknüpfung bildet das Straßennetzwerk einer Datenbasis (vgl. 3.3.2). Im Modell wird eine Straße als ***Course***, ein Verkehrsknoten als ***Area*** bezeichnet. Das Straßennetzwerk, das aus Objekten dieser beiden Typen besteht, ist als Graph RN (Road Network) modelliert.

Spurzellen: In Abschnitt 3.4.2 werden Spurzellen eingeführt, um einzelne Spuren von Verkehrsknoten zu gruppieren. Im Kontext der Modellierung wird eine Spurzelle durch Objekte repräsentiert, die vom Typ ***LaneCell*** abgeleitet sind. Es erweist sich als sinnvoll, auch Spuren von Objekten des Typs ***Course*** in Spurzellen zu gruppieren und alle Spurzellen der Datenbasis zu verknüpfen. Man erhält dadurch einen Graphen LANECELLS, der eine verfeinerte Darstellung des Graphen RN darstellt.

Spuren: Einzelne Fahrspuren von Strecken werden durch Objekte des Typs ***Lane*** oder davon abgeleiteter Typen repräsentiert. Auch sie werden in einem Graphen LANES verknüpft, der damit eine Verfeinerung des Graphen LANECELLS ist.

Die Struktur kann am Beispiel des Situationsbewusstsein-Parcours verdeutlicht werden [1].

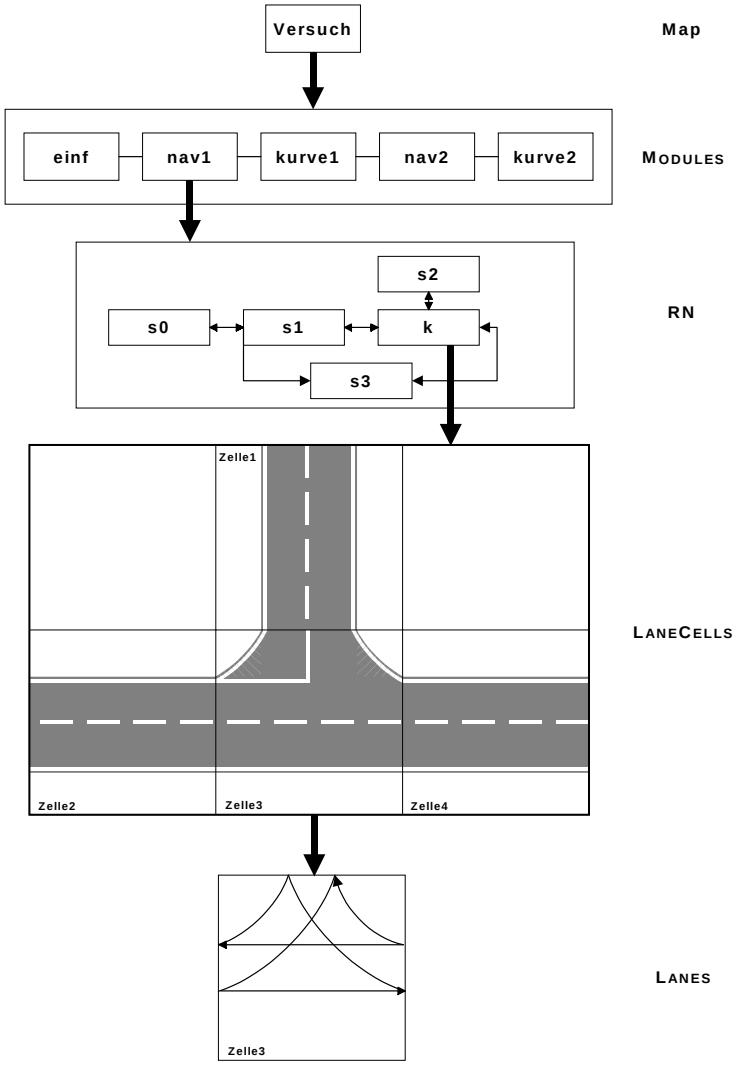

Abbildung 4.4.: Struktur des Modells am Beispiel des vereinfachten Situationsbewusstsein-Parcours

Abbildung 4.4 zeigt für diesen Fall die einzelnen Schichten des Modells. Das in der Abbildung dargestellte Streckennetz für die Navigationsaufgabe ist dem Beispiel auf Seite 38 entnommen.

[1] Der Situationsbewusstsein-Parcours wird in Abschnitt 1.5 (S. 18) vorgestellt. Die Beschreibung einer der Übersicht wegen vereinfachten Version des Parcours ist auf Seite 31 vorgestellt.

4.2.2. Knoten, Ports und Verknüpfungen

Die Knoten[2] der Graphen MODULES, RN, LANECELLS und LANES sind von einem gemeinsamen Grundtyp ***Node*** abgeleitet. Da die geometrische Instanzierung (s. Abschnitt 5.1) auf jeder Modellebene ähnlich abläuft, können die entsprechenden Verfahren in diesem Grundtyp als virtuelle Methoden implementiert werden. Er ist wie folgt definiert:

Definition 4.1 (***Node***)
Ein Knoten $v \triangleright$ ***Node*** besteht aus folgenden Elementen:

1. Einen Eintrag $v.parent \triangleright$ ***Node***. Dies ist der Knoten, zu dem v auf der nächst höheren Ebene gehört.

2. Einer Menge $v.ChildNodes = \{c_1, \ldots, c_{n_c}\}$ von Kindern des Knotens. Dabei ist $c_i \triangleright$ ***Node***. Mit $v.c_i$ wird das i-te Element von $v.ChildNodes$ bezeichnet.

3. Einem Tupel $v.Ports = (p_1, \ldots, p_{n_p})$ von Ports des Knoten. Dabei ist $p_i \triangleright v.$***Port***. Mit $v.p_i$ wird das i-te Element von $v.Ports$ bezeichnet. •

In der Menge $v.ChildNodes$ werden die Knoten gespeichert, die v auf der nächst tieferen Ebene repräsentieren. Ist beispielsweise v ein Knoten des Graphen LANECELLS, so sind seine Kinder Knoten des Graphen LANES, denn jede Spurzelle besteht aus einzelnen Spuren.

Die Knoten eines Graphen werden **über sog. Ports verbunden**. Wie in obiger Definition beschrieben, enthält jeder Knoten ein Tupel von Ports. Die folgende Definition zeigt den Grundtyp von Ports, von dem in den nächsten Abschnitten die Ports der verschiedenen Ebenen abgeleitet werden.

Definition 4.2 (***Port***)
Ein Objekt $p \triangleright$ ***Port*** besteht aus folgenden Elementen:

1. Einer Menge $p.Edges = \{v_1.p_{i_1}, \ldots, v_{n_e}.p_{i_{n_e}}\}$ von Kanten, die vom Port p ausgehen. Es ist $v_j \triangleright Node$ und $p_{i_j} \in v_j.Ports$. Mit $p.v_j$ wird der Knoten des j-ten Elements $v_j.p_{i_j}$ von $p.Edges$ bezeichnet.

2. Einer Generalisierung $p.Generalization = v.p$ des Ports, wobei $v \triangleright Node$ und $p \triangleright Port$ ist.

[2] Die Knoten der Graphen sind nicht mit Verkehrsknoten zu verwechseln.

3. Einer Menge $p.Refinements = \{v_1.p_{i_1}, \ldots, v_{n_r}.p_{i_{n_r}}\}$ von Verfeinerungen des Ports. Dabei ist $v_i \triangleright Node$. •

Die Anzahl der Elemente in $v.Ports$ eines Knotens v wird von der Ebene festgelegt, in der sich der Knoten befindet. Näheres hierzu findet sich in den Definitionen der abgeleiteten Knoten.

Die **Verbindung zwischen den Ebenen** wird durch $p.Refinements$ bzw. $p.Generalization$ eines Ports $p \in v.Ports$ hergestellt. Wenn es sich bei v beispielsweise um einen Knoten des Graphen MODULES handelt, dann enthält die Menge $p.Refinements$ eines jeden Ports genau die durch Grammatik 3.11 (S. 44) beschriebenen Ports von Strecken. Umgekehrt ist die Generalisierung des Ports p' von Strecke v' der Modul-Port p:

$$v'.p'.Generalization = v.p$$

Während für einen Port p eines Knotens der Ebenen MODULES, RN und LANECELLS immer $p.Refinements \neq \emptyset$ ist, kann $p.Generalization = \emptyset$ sein. Dies ist z.B. bei den Ports der Strecken s1, s3 und k aus Abbildung 4.4 der Fall.

Definition 4.3 (Verknüpfung von Knoten)
Für die Verknüpfung zweier Knoten $v_1, v_2 \triangleright \boldsymbol{Node}$, $v_1 \neq v_2$ über die Ports $v_1.p_i$ und $v_2.p_j$ muss gelten:

1. $v_1.p_i \in v_2.p_j.Edges$ und $v_2.p_j \in v_1.p_i.Edges$
2. Wenn $v_1.p_i.Generalization = a.p_k$ und $v_2.p_j.Generalization = b.p_l$, dann sind auch a und b über die Ports p_k und p_l verknüpft.
3. Wenn
$$c.p_m \in v_1.p_i.Refinements, c.Generalization = v_1.p_i$$
und
$$d.p_n \in v_2.p_j.Refinements, d.Generalization = v_2.p_j$$
dann sind auch c und d über die Ports p_m und p_n verknüpft. •

Aufbauend auf die Typen ***Node*** und ***Port*** können jetzt die Knoten-Typen für die Graphen der einzelnen Ebenen des Modells definiert werden. Dabei kann jedem Knoten-Typ eine entsprechende Struktur aus der formalen Beschreibung zugeordnet werden. Die folgenden Abschnitte führen die verschiedenen Knoten-Typen ein. Einen Überblick gibt Abbildung 4.12 (S. 81).

4.2.3. Karten

Obwohl es in einer Datenbasis nur eine Karte gibt, wird der Typ ***Map*** zur Vereinheitlichung von ***Node*** abgeleitet:

Definition 4.4 (***Map***)
Ein Knoten $map \triangleright$ ***Map*** ist vom Typ ***Node*** abgeleitet. Es gilt:

1. $map.parent = \emptyset$.
2. Die Elemente in $map.ChildNodes$ sind vom Typ ***Module***.
3. $map.Ports = \emptyset$.
4. $map.p_i.Generalization = \emptyset$.
5. $map.p_i.Refinements = \emptyset$. •

Karten werden gemäß Grammatik 3.3 (S. 30) beschrieben.

4.2.4. Ebene MODULES

Ein Knoten des Typs ***Module*** repräsentiert eine **Modul-Instanz**, deren Modul-Schablone die in Grammatik 3.6 (S. 35) angegebene formale Beschreibung hat.

Definition 4.5 (***Module***)
Ein Knoten $module \triangleright$ ***Module*** ist vom Typ ***Node*** abgeleitet. Es gilt:

1. $module.parent \triangleright$ ***Map***.
2. Die Elemente in $module.ChildNodes$ sind vom Typ ***RNNode***.
3. Bei $module.Ports = (p_1, \ldots, p_{n_p})$ ist $p_i \triangleright$ ***ModulePort*** und n_p beliebig, aber fest gewählt[3].
4. Für $v.p \in module.p_i.Edges$ gilt: $v \triangleright$ ***Module***.
5. $module.p_i.Generalization = \emptyset$.
6. Bei einem Element $v.p \in module.p_i.Refinements$ ist $v \triangleright$ ***RNNode***. •

Der Typ ***ModulePort*** ist vom Typ ***Port*** abgeleitet.
Abbildung 4.5 zeigt die Einordnung von ***Module*** in das Modell als Klassendiagramm.
Die Kanten des Graphen MODULES ergeben sich direkt aus der formalen Beschreibung des Modul-Ablaufs gemäß Grammatik 3.5 (S. 31).

[3] In der Praxis hat sich herausgestellt, dass ein Maximum von $n_p = 10$ ausreichend ist.

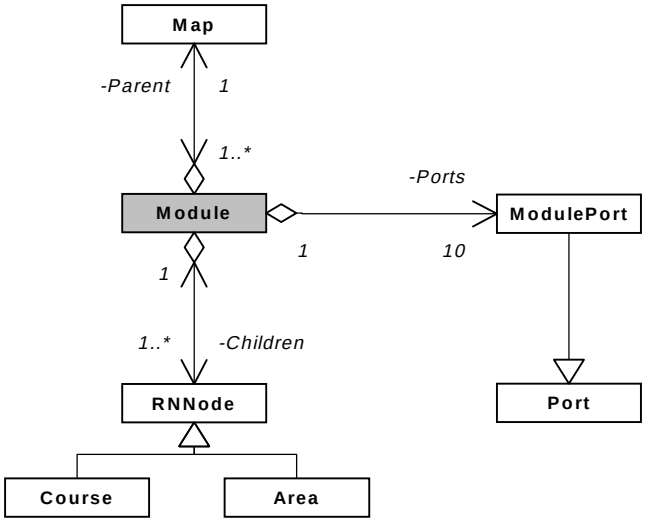

Abbildung 4.5.: Einordnung von ***Module*** in das Modell. Die Kinder von Modulen sind Straßen oder Verkehrsknoten, beide abgeleitet vom Typ ***RNNode***.

4.2.5. Ebene RN

Auf der Ebene RN besteht der Graph aus zwei Typen von Knoten. Ein Knoten vom Typ ***Course*** entspricht der **Instanz einer Straße**. Die dazugehörige Schablone wird mittels Grammatik 3.12 (S. 45) beschrieben. **Instanzen von Verkehrsknoten** (Beschreibung der Schablone in Grammatik 3.15, S. 50) werden durch den Typ ***Area*** modelliert. Beide Typen sind vom gemeinsamen Grundtyp ***RNNode*** abgeleitet.

Definition 4.6 (***RNNode***)
Ein Knoten $rnnode \triangleright$ ***RNNode*** ist vom Typ ***Node*** abgeleitet. In $rnnode.Ports = (p_1, \ldots, p_{n_p})$ ist $p_i \triangleright$ ***RNPort***. •

Die Besonderheiten des Typs ***RNPort*** werden im Abschnitt 5.1.5 definiert, an dieser Stelle wird der Typ nur eingeführt:

Definition 4.7 (***RNPort***)
Der Typ ***RNPort*** ist vom Typ ***Port*** abgeleitet. •

Definition 4.8 (***Course***)
Ein Knoten $course \triangleright$ ***Course*** ist vom Typ ***RNNode*** abgeleitet. Es gilt:

1. $course.parent \triangleright$ ***Module***.
2. Die Elemente in $course.ChildNodes$ sind vom Typ ***CourseLaneCell***.
3. Für $course.Ports = (p_1, \ldots, p_{n_p})$ gilt $n_p = 2$.
4. Für $v.p \in course.p_i.Edges$ gilt: $v \triangleright$ ***RNNode***.
5. Für $course.p_i.Generalization = v.p$ gilt: $v \triangleright$ ***Module***.
6. Bei einem Element $v.p \in course.p_i.Refinements$ ist $v \triangleright$ ***CourseLaneCell*** (dieser Typ wird im nächsten Abschnitt definiert). •

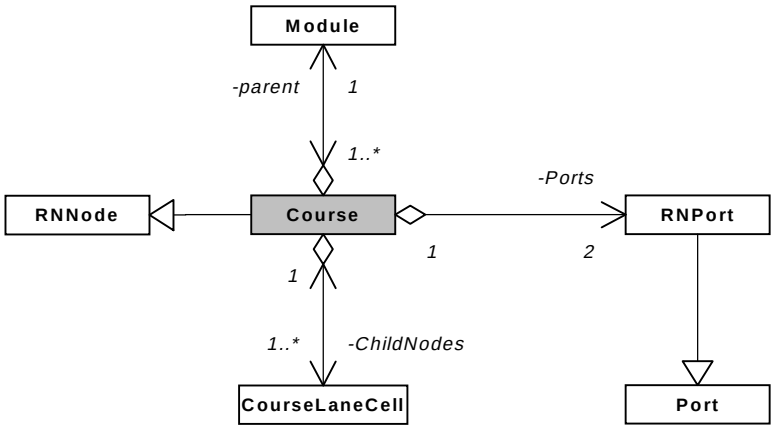

Abbildung 4.6.: Einordnung von ***Course*** in das Modell.

Definition 4.9 (***Area***)
Ein Knoten $area \triangleright$ ***Area*** ist vom Typ ***RNNode*** abgeleitet. Es gilt:

1. $area.parent \triangleright$ ***Module***.
2. Die Elemente in $area.ChildNodes$ sind vom Typ ***AreaLaneCell***.
3. Für $area.Ports = (p_1, \ldots, p_{n_p})$ ist n_p beliebig, aber fest gewählt[4].
4. Für $v.p \in area.p_i.Edges$ gilt: $v \triangleright$ ***RNNode***.

[4] In der Praxis hat sich gezeigt, dass selbst bei komplizierten Kreuzungen 4 Ports pro Seite des Rechtecks, das einem Verkehrsknoten zugrunde liegt, ausreichen. Demnach wird als Maximum z.B. $n_p = 16$ gewählt.

5. Für $area.p_i.Generalization = v.p$ gilt: $v \triangleright \textbf{\textit{Module}}$.

6. Bei einem Element $v.p \in area.p_i.Refinements$ ist $v \triangleright \textbf{\textit{AreaLaneCell}}$ (dieser Typ wird im nächsten Abschnitt definiert). •

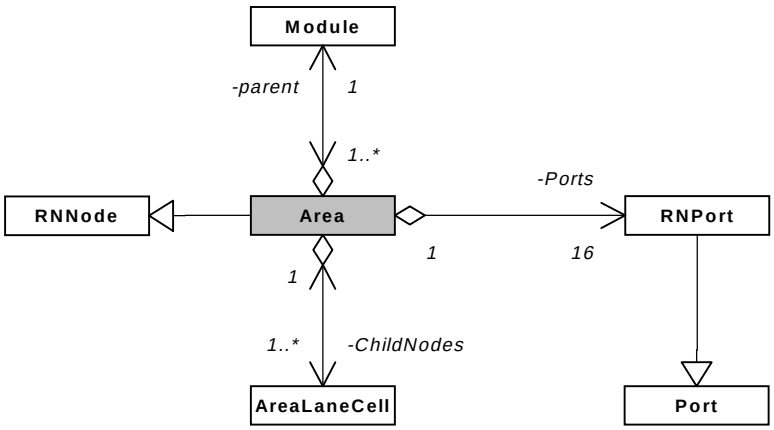

Abbildung 4.7.: Einordnung von ***Area*** in das Modell.

Abbildung 4.6 zeigt die Einordnung von ***Course*** in das Modell, Abbildung 4.7 die von ***Area***.

Innerhalb eines Moduls ist die Verknüpfung der Knoten des Graphs RN in der formalen Beschreibung explizit angegeben (vgl. Grammatik 3.9, S. 37). Werden zwei Module des Graphen MODULES verknüpft, dann werden in RN implizit auch die Strecken verbunden, die gemäß Grammatik 3.11 (S. 44) als Portverfeinerungen der beiden Module beschrieben sind. Dadurch werden die Bedingungen in Definition 4.3 erfüllt.

4.2.6. Ebene LANECELLS

Die Knoten des Graphen LANECELLS sind vom Typ ***CourseLaneCell*** oder ***AreaLaneCell*** abgeleitet, je nachdem, ob der zugehörige Knoten der übergeordneten Ebene vom Typ ***Course*** oder ***Area*** ist. Wie sich im Verlauf dieses Kapitels zeigen wird, haben beide Knotentypen einige Elemente und Eigenschaften gemeinsam. Deswegen werden sie von einem gemeinsamen Grundtyp ***LaneCell*** abgeleitet:

Definition 4.10 (***LaneCell***)
Ein Knoten $lanecell \triangleright$ ***LaneCell*** ist vom Typ ***Node*** abgeleitet. Für $lanecell.Ports = (p_1, \ldots, p_{n_p})$ gilt: $p_i \triangleright$ ***LaneCellPort***. •

Definition 4.11 (***LaneCellPort***)
Der Typ ***LaneCellPort*** ist vom Typ ***Port*** abgeleitet. •

Gemäß Grammatik 3.12 (S. 45) besteht der Verlauf einer Straße aus einer Abfolge von Geraden- und Kurvenstücken, die in der exakten Mitte der Straße verlaufen. Zusammen mit dem Querschnittsprofil erhält man daraus die parallel verlaufenden Spuren. Die **Spurzellen einer Straße** sind damit implizit gegeben[5]:

- Eine Spurzelle enthält alle parallelen Spuren, die zu *einem* Geradenstück gehören.
- Ein Kurvenstück wird aufgeteilt in einen Übergangsbogen beim Kurveneintritt, einem Kreisbogen und einen Übergangsbogen beim Kurvenaustritt. Die Spurzelle einer Straße kann aus allen parallelen Spuren bestehen, die zu *einem* dieser Teile eines Kurvenstücks gehören.

Hieraus ergibt sich auch ein Teil der Kanten von LaneCells: Innerhalb einer Straße sind die Spurzellen aufeinanderfolgender Geraden- und Kurvenstücke miteinander verbunden.

Definition 4.12 (***CourseLaneCell***)
Ein Knoten $clc \triangleright$ ***CourseLaneCell*** ist vom Typ ***LaneCell*** abgeleitet. Es gilt:

1. $clc.parent \triangleright$ ***Course***.
2. Die Elemente in $clc.ChildNodes$ sind vom Typ ***CourseLane***.
3. Für $clc.Ports = (p_1, \ldots, p_{n_p})$ gilt $n_p = 2$.
4. Für $v.p \in clc.p_i.Edges$ gilt : $v \triangleright$ ***LaneCell***.
5. Für $clc.p_i.Generalization = v.p$ gilt: $v \triangleright$ ***Course***.
6. Bei einem Element $v.p \in clc.p_i.Refinements$ ist $v \triangleright$ ***CourseLane***. •

Spurzellen von Verkehrsknoten werden in der formalen Beschreibung explizit angegeben (vgl. Grammatik 3.18, 54). Der Typ ***AreaLaneCell***, der sie im Modell repräsentiert, ist so definiert:

[5] Näheres hierzu findet sich in Abschnitt 4.4.

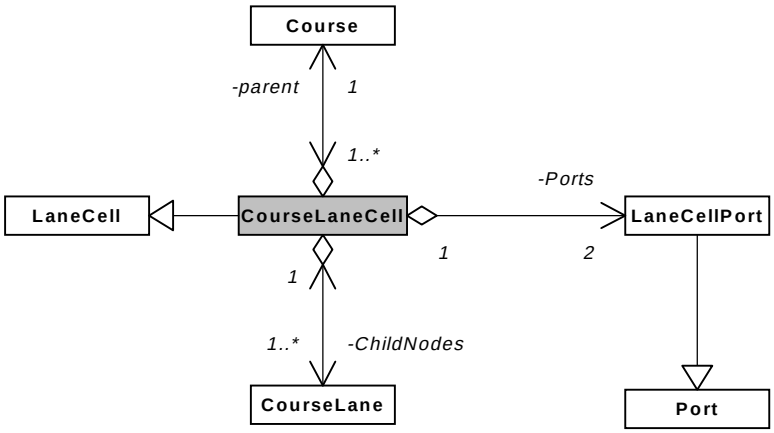

Abbildung 4.8.: Einordnung von ***CourseLaneCell*** in das Modell.

Definition 4.13 (***AreaLaneCell***)
Ein Knoten $alc \triangleright$ ***AreaLaneCell*** ist vom Typ ***LaneCell*** abgeleitet. Es gilt:

1. $alc.parent \triangleright$ ***Area***.
2. Die Elemente in $alc.ChildNodes$ sind vom Typ ***AreaLane***.
3. Bei $alc.Ports = (p_1, \ldots, p_{n_p})$ ist $n_p \geq 1$ [6].
4. Für $v.p \in alc.p_i.Edges$ gilt : $v \triangleright$ ***LaneCell***.
5. Für $alc.p_i.Generalization = v.p$ gilt: $v \triangleright$ ***Area***.
6. Bei einem Element $v.p \in alc.p_i.Refinements$ ist $v \triangleright$ ***AreaLane***. •

Der Typ ***CourseLaneCell*** wird gemäß Abbildung 4.8 in das Modell eingeordnet. Abbildung 4.9 zeigt die Einordnung des Typs ***AreaLaneCell***.
Die Kanten des Graphen LaneCells, die innerhalb eines Verkehrsknotens verlaufen, können aus der Verknüpfung der Spuren, beschrieben mittels Grammatik 3.20 (S. 57), und den in Definition 4.3 genannten Bedingungen abgeleitet werden. Zusätzliche Kanten ergeben sich aus dem mittels Grammatik 3.9 (S. 37) beschriebenen Streckennetz: Bei der Verknüpfung zweier Strecken (Straße oder Verkehrsknoten), werden auch die jeweiligen Spurzellen, gegeben als Verfeinerung der beiden verbundenen Strecken-Ports, verknüpft.

[6] Die Ports von Knoten des Typs ***AreaLaneCell*** werden in Abhängigkeit von den Spurverknüpfungen automatisch erzeugt. Näheres hierzu findet sich in Abschnitt 4.5.5

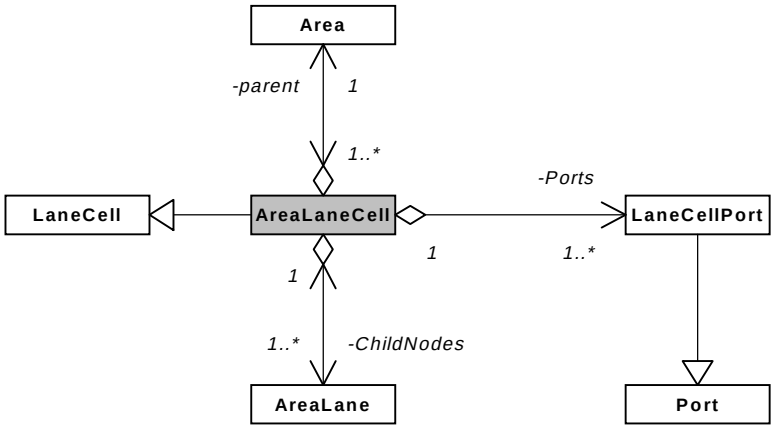

Abbildung 4.9.: Einordnung von ***AreaLaneCell*** in das Modell.

4.2.7. Ebene LANES

Auf der untersten Modell-Ebene bilden die Spuren den Graphen LANES. Dessen Knoten bestehen aus Spuren von Straßen, repräsentiert durch den Typ ***CourseLane***, und aus Spuren von Verkehrsknoten, repräsentiert durch den Typ ***AreaLane***. Wieder gibt es Eigenschaften und Elemente, die beide Spurtypen gemeinsam haben, weswegen der gemeinsame Grundtyp ***Lane*** eingeführt wird.

Definition 4.14 (***Lane***)
Ein Knoten $lane \triangleright$ ***Lane*** ist vom Typ ***Node*** abgeleitet. Es gilt:

1. $lane.ChildNodes = \emptyset$ und $lane.p_i.Refinements = \emptyset$ (Spuren sind die unterste Ebene der Modellierung).

2. Für $lane.Ports = (p_1, \ldots, p_{n_p})$ gilt: $p_i \triangleright$ ***LanePort*** und $n_p = 2$. •

Definition 4.15 (***LanePort***)
Der Typ ***LanePort*** ist vom Typ ***Port*** abgeleietet. •

Wie schon erwähnt, erhält man die **Spuren einer Straße** durch Parallelverschiebung zu der durch den Lageplan (Grammatik 3.13, S. 47) beschriebenen Straßenmitte. Im Modell sind sie vom Typ ***CourseLane***:

Definition 4.16 (***CourseLane***)
Ein Knoten $cl \triangleright$ ***CourseLane*** ist vom Typ ***Lane*** abgeleitet. Es gilt:

1. $cl.parent \triangleright \boldsymbol{CourseLaneCell}$.
2. Für $v.p \in cl.p_1.Edges$ bzw. $(v, .) \in cl.p_2.Edges$ gilt: $v \triangleright \boldsymbol{CourseLane}$ oder $v \triangleright \boldsymbol{AreaLane}$.
3. Für $cl.p_i.Generalization = v.p$ gilt: $v \triangleright CourseLaneCell$. •

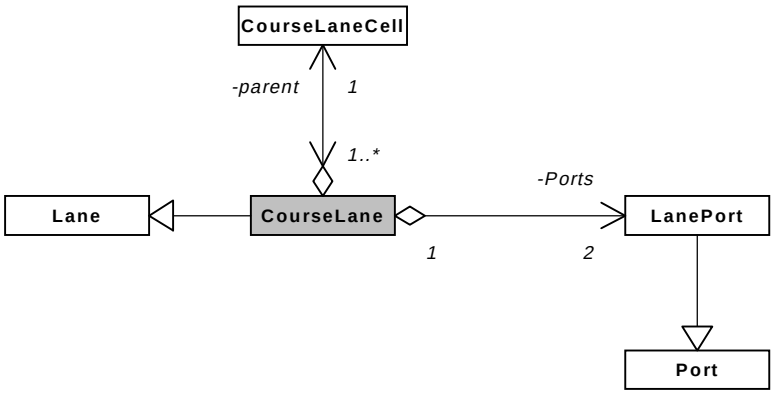

Abbildung 4.10.: Einordnung von ***CourseLane*** in das Modell.

Im Graphen LANES gibt es eine Kante zwischen den Ports zweier Spuren einer Straße, wenn die Generalisierungen dieser Ports im Graphen LANECELLS verbunden sind.

Ein Aufsetzpunkt, der bei der formalen Beschreibung gem. Grammatik 3.22 (S. 63) definiert wird, ist der Beginn einer Spur $l \triangleright \boldsymbol{CourseLane}$ in der ersten Spurzelle einer Straße. Zur Speicherung der Aufsetzpunkte wird der Typ ***Map*** folgendermaßen erweitert:

Definition 4.17 (Erweiterung von ***Map***)
Zusätzlich zu den bisher genannten Elementen enthält ein Objekt $map \triangleright \boldsymbol{Map}$ eine Menge

$$SetupPoints = \{(name_1, l_1), \ldots, (name_{n_{SU}}, l_{n_{SU}})\}$$

von Aufsetzpunkten, an deren Beginn der Fahrer beim Start der Simulation gesetzt werden kann. Dabei ist $name_i$ eine Zeichenkette, die den Namen des Aufsetzpunktes repräsentiert, und $l_i \triangleright \boldsymbol{CourseLane}$. •

Wie Grammatik 3.18 (S. 54) zu entnehmen ist, werden die **Spuren von Verkehrsknoten** explizit beschrieben. Im Modell sind sie durch den Typ ***AreaLane*** repräsentiert:

Definition 4.18 (***AreaLane***)
Ein Knoten $al \triangleright$ ***AreaLane*** ist vom Typ ***Lane*** abgeleitet. Es gilt:

1. $al.parent \triangleright$ ***AreaLaneCell***.
2. Für $v.p \in al.p_1.Edges$ bzw. $(v, .) \in al.p_2.Edges$ gilt: $v \triangleright$ ***AreaLane*** oder $v \triangleright$ ***CourseLane***.
3. Für $al.p_i.Generalization = v.p$ gilt: $v \triangleright AreaLaneCell$. •

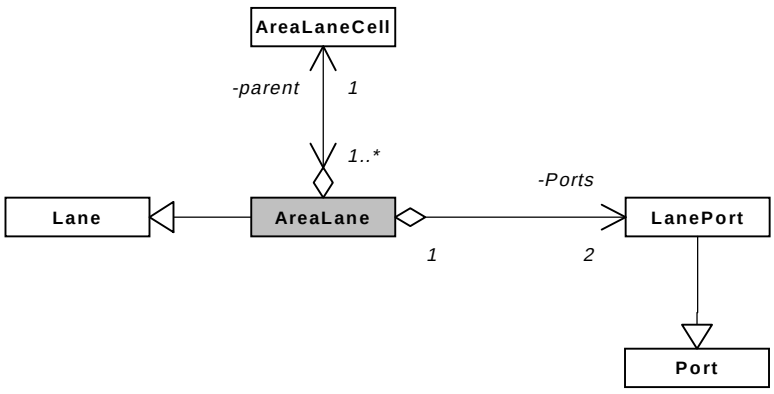

Abbildung 4.11.: Einordnung von ***AreaLane*** in das Modell.

Auch die Kanten zwischen Spuren von Verkehrsknoten sind explizit mittels Grammatik 3.20 (S. 57) beschrieben. Weitere Kanten von LANES ergeben sich wieder aus den Bedingungen in Definition 4.3: Sind zwei Spurzellen in LANECELLS miteinander verbunden, dann auch die Verfeinerungen der entsprechenden Ports.

In den Abbildungen 4.10 und 4.11 ist die Einordnung der beiden Typen von Spurzellen in das Modell dargestellt.

4.2.8. Zusammenfassung

Die Typen von Knoten der einzelnen Modell-Ebenen sind in Abbildung 4.12 in Form eines UML-Klassendiagramms dargestellt. Eingetragen sind auch die Ebenen, zu denen die Knoten-Typen gehören.

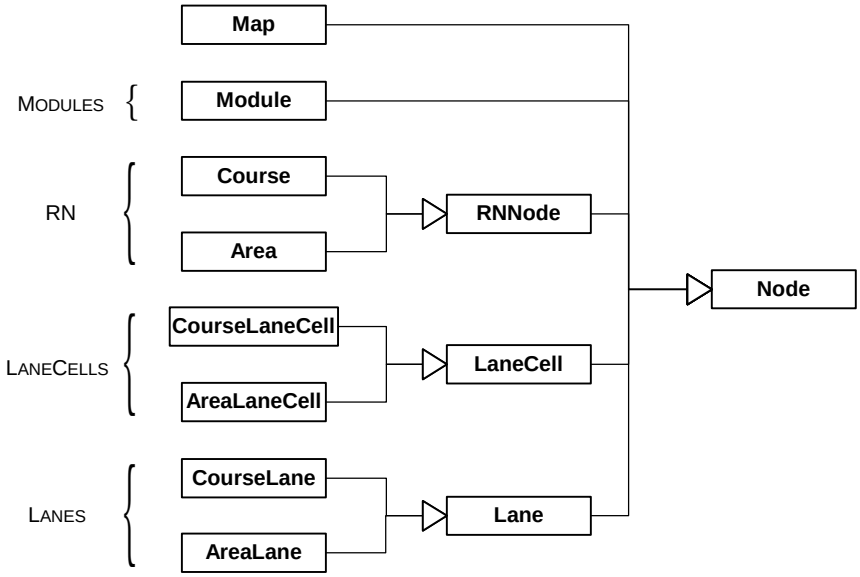

Abbildung 4.12.: Modell-Ebenen als Klassendiagramm

Die Aufteilung der Repräsentation in 5 Modell-Ebenen und die einheitliche Ableitung der Knoten auf den einzelnen Ebenen von gemeinsamen Grundtypen erleichtern die Erweiterbarkeit des Modells (vgl. Anforderung in Abschnitt 2.5). Jeder neue Modellierungsaspekt des Straßennetzwerks wird in der Ebene implementiert, auf die sich seine Informationen beziehen. So werden beispielsweise Daten, die Fahrspuren im Allgemeinen betreffen, auf der Ebene LANES im Grundtyp ***Lane*** gespeichert, neue Typen von Strecken, wie z.B. Stadtgebiete, werden auf der Ebene RN vom Grundtyp ***RNNode*** abgeleitet.

4.2.9. Generierung aus der formalen Beschreibung

Dieser Abschnitt gibt einen groben Überblick, wie die topologische Modell-Struktur der Ebenen MODULES und RN aus der formalen Beschreibung einer Datenbank aufgebaut wird. Die Vorgehensweise bei den Ebenen LANECELLS und LANES wird getrennt für Straßen und Verkehrsknoten in den Abschnitten 4.4 und 4.5 erläutert.

Der Anwender erstellt eine Textdatei, die die formale Beschreibung einer Datenbasis, wie im vorhergehenden Kapitel dargestellt, enthält. Diese Textdatei wird von einem Parser analysiert, dabei auf syntaktische Korrektheit geprüft

und in eine Baumstruktur übertragen. Der Parse-Baum spiegelt lediglich *die Struktur der Beschreibung* wieder. Seine Knoten stellen Repräsentationen von Konstrukten der Beschreibung dar. Dazu gehören z.B. Zuweisungen, Mengen, Tupel und Blöcke der Form

```
<Bezeichner> <Bezeichner>
{
    ...
};
```

Das Modell wird aus dem Parse-Baum gem. Algorithmus 4.1 aufgebaut. Es handelt sich dabei um eine der Übersichtlichkeit wegen gekürzten Darstellung. Weggelassen wird die Beschreibung diverser Überprüfungen, z.B.:

- Wird ein Modul oder eine Strecke von einer Schablone instanziert, die nicht definiert wurde?
- Sind die Namen von Karten, Modulen/Modulschablonen, Strecken/Streckenschablonen eindeutig?
- Gibt es pro Karte mindestens eine Modulinstanz, pro Modulinstanz mindestens eine Streckeninstanz?

Ebenso wird das Einfügen von Aufsetzpunkten in die Karten und von Strecken-Events in Spuren nicht näher dargestellt. Der Algorithmus bekommt den Parse-Baum und den Namen einer Karte übergeben. Er liefert das Modell dieser Karte zurück.

Algorithmus 4.1 *GeneriereModell*(*Parse − Baum*, *Kartenname*)

finde Beschreibung von Karte *Kartenname* im Parse-Baum
erzeuge $m \triangleright$ ***Map***
for all Modulinstanzen der Kartenbeschreibung **do**
  erzeuge $mod \triangleright$ ***Module***
  füge mod zu $m.ChildNodes$ hinzu
  for all Streckeninstanzen der Modulschablone von mod **do**
    if Straße **then**
      erzeuge $s \triangleright$ ***Course*** (vgl. Abschnitt 4.4.4)
    else
      erzeuge $s \triangleright$ ***Area*** (vgl. Abschnitt 4.5.5)
    end if
    füge s zu $mod.ChildNodes$ hinzu
  end for
  verknüpfe die Elemente in $mod.ChildNodes$ wie in der Schablone beschrieben
end for
verknüpfe die Elemente in $map.ChildNodes$ wie in der Karte beschrieben

4.3. Fahrspuren

4.3.1. Überblick

Die geometrische Modellierung der Fahrspuren von Straßen und Verkehrsknoten erfolgt, wie in [3] vorgeschlagen, über ebene Parameterkurven. Die hierzu benötigten Begriffe aus der Differentialgeometrie (wie z.B. Parametrisierung, Bogenlänge, Krümmung) sowie einige grundlegenden Sachverhalte, die Transformationen solcher Parameterkurven betreffen, werden in den Anhängen B.1 (S. 211 ff.) und B.2 (S. 213 ff.) hergeleitet. In 4.3.2 wird das Modell von Fahrspuren vorgestellt, in 4.3.3 die Behandlung von Strecken-Events, die auf den Fahrspuren liegen.

4.3.2. Modell

Fahrspuren von Straßen (Typ ***CourseLane***) und von Verkehrsknoten (Typ ***AreaLane***) sind vom Typ ***Lane*** abgeleitet, der einige grundlegende Eigen-

schaften modelliert: Eine Fahrspur wird durch den Verlauf ihrer exakten Mitte und Querschnittsinformationen beschrieben.

Die verschiedenen Arten von Parameterkurven, die den **Verlauf von Fahrspuren** beschreiben, sind vom Typ ***Curve*** abgeleitet:

Definition 4.19 (***Curve***)
Ein Objekt $curve \triangleright$ ***Curve*** repräsentiert eine ebene Parameterkurve. Es enthält folgende Elemente:

1. Einen Eintrag $S \in \mathbb{R}^+$, die gesamte Länge der Kurve.
2. Eine Funktion $p : [0, S] \to \mathbb{R}^2$, die jedem Wert von $s \in [0, S]$ die Koordinaten der Kurve an dieser Stelle zuweist.
3. Eine Funktion $\alpha : [0, S] \to \mathbb{R}$, die jedem Wert von $s \in [0, S]$ den Winkel zuweist, den die Tangente der Kurve $p(s)$ an dieser Stelle mit der positiven x-Achse einnimmt.
4. Eine Funktion $\kappa : [0, S] \to \mathbb{R}$, die jedem Wert von $s \in [0, S]$ die Krümmung der Kurve an der Stelle s zuweist. •

Bei der formalen Beschreibung einer Datenbasis tauchen **Querschnittsprofile** an drei Stellen auf: Sie werden modulübergreifend gemäß Grammatik 3.21 (S. 58) definiert und Straßen (vgl. Grammatik 3.12, S. 45) bzw. den Ports von Verkehrsknoten (vgl. Grammatik 3.16, S. 51) zugewiesen. Bei Straßen ist das Querschnittsprofil über die gesamte Länge hin konstant, d.h. die Breite und relative Höhe der einzelnen Spuren ändert sich nicht. In Verkehrsknoten können sich diese Spurinformationen ändern: Grammatik 3.19 (S. 55) ist zu entnehmen, dass Breiten- und Höhenvorgaben jeweils getrennt für den Beginn und das Ende einer Spur vorliegen.

Bei Verkehrsknoten kann zusätzlich für jede Spur angegeben werden, in welche Richtung sie führt (vgl. auch hierzu Grammatik 3.19, S. 55). Bei Spuren von Straßen muss diese Information nicht angegeben werden, da sie immer geradeaus führen.

Alle erwähnten Spurinformationen werden im Modell durch folgende Definition vereinheitlicht:

Definition 4.20 (***LaneInfo***)
Ein Objekt $li \triangleright$ ***LaneInfo*** beschreibt eine Spur eines Querschnitts und besteht aus:

1. Der Breite am Spuranfang, $w_1 \in \mathbb{R}^+$, und der Breite am Spurende, $w_2 \in \mathbb{R}^+$.
2. Der relativen Höhe am Spuranfang $h_1 \in \mathbb{R}$, und der relativen Höhe am Spurende, $h_2 \in \mathbb{R}^+$.
3. Der Fahrtrichtung der Spur, $dir \in \{onwards, towards\}$.
4. Der Richtung, in die die Spur führt:

$$heading \subseteq \{straight, left, right\}$$ •

Insgesamt führt dies zu folgender Erweiterung von ***Lane***:

Definition 4.21 (Erweiterung von ***Lane***)
Zusätzlich zu den bisher genannten Elementen gilt für ein Objekt l des Typs ***Lane***:

1. l enthält ein Element $curve$ vom Typ ***Curve***.
2. l enthält ein Element $LaneInfo$ vom Typ ***LaneInfo***.
3. l enthält eine Funktion $w : [0, l.curve.S] \rightarrow \mathbb{R}^+$, die zu jedem Punkt s auf der Parameterkurve der Spur die Breite an dieser Stelle gemäß

$$l.w(s) = (1 - I(\frac{s}{l.curve.S})) \cdot l.LaneInfo.w_1 + I(\frac{s}{l.curve.S}) \cdot l.LaneInfo.w_2$$

   berechnet.
4. l enthält eine Funktion $h_{rel} : [0, l.curve.S] \rightarrow \mathbb{R}$, die zu jedem Punkt s auf der Parameterkurve der Spur die relative Höhe an dieser Stelle gemäß

$$l.h_{rel}(s) = (1 - I(\frac{s}{l.curve.S})) \cdot l.LaneInfo.h_1 + I(\frac{s}{l.curve.S}) \cdot l.LaneInfo.h_2$$

   berechnet.
5. l enthält eine Funktion $h : [0, l.curve.S] \rightarrow \mathbb{R}$, die zu jedem Punkt s auf der Parameterkurve der Spur die absolute Höhe an dieser Stelle berechnet. •

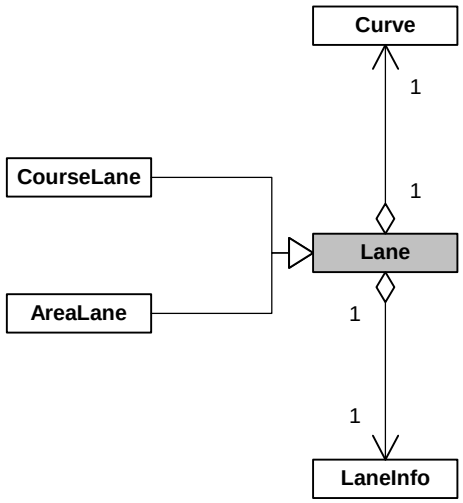

Abbildung 4.13.: Einordnung von ***Lane*** in das Modell.

Bei der Funktion $I : [0,1] \rightarrow [0,1]$ handelt es sich um eine stetige, monoton steigende Interpolationsfunktion mit folgenden Eigenschaften:

$$\begin{aligned} I(0) &= 0 \\ I(1) &= 1 \end{aligned}$$

Im einfachsten Fall kann hierbei eine lineare Interpolation verwendet werden:

$$I(t) = t$$

Einen glatteren Verlauf der Breite bzw. der Höhe erhält man durch

$$I(t) = -2t^3 + 3t^2$$

Die Funktion h aus dem letzten Punkt von Definition 4.21 wird für Spuren von freien Strecken und Spuren von Verkehrsknoten unterschiedlich definiert (Abschnitte 4.4.3 bzw. 4.5.4). In jedem Fall wird dabei aber in h die relative Höhe h_{rel} der Spur eingerechnet.

Die Definition von ***Lane*** zeigt, wie die in Abschnitt 2.4 verlangten Informationen zur Verfügung gestellt werden: Position, Bahntangentenwinkel, Krümmung, Breite und Höhe einer jeden Fahrspur können an beliebiger Stelle innerhalb der Maschinengenauigkeit berechnet werden.

Die Einordnung des erweiterten Typs ***Lane*** ist zusammenfassend in Abbildung 4.13 dargestellt.

4.3.3. Strecken-Events

Bei der formalen Beschreibung einer Datenbasis können Strecken-Events sowohl für Schablonen von Strecken (Schablonen von Straßen s. Grammatik A.1, S. 207, Schablonen von Verkehrsknoten s. Grammatik A.3, S. 208) als auch für Instanzen von Strecken (s. Grammatik A.4, S. 209) definiert werden. Bei der Umwandlung der Beschreibung in das Modell werden die Information der Schablonen in die Instanzen kopiert. Die Strecken-Events können deshalb **direkt in den Fahrspuren gespeichert** werden. Hierbei wird ausgenutzt, dass die Spuren von Straßen und Verkehrsknoten vom Grundtyp ***Lane*** abgeleitet sind:

Definition 4.22 (Erweiterung von ***Lane***)
Zusätzlich zu den bisher genannten Elementen enthält ein Objekt l des Typs ***Lane*** eine Menge

$$Events = \{e_1, \ldots, e_n\}$$

von Strecken-Events. Die Elemente e_i sind dabei vom Typ ***Event***. •

Der Typ ***Event*** enthält die Position des Events auf der Spur sowie die Argumente, die mittels Grammatik A.2 (S. 208) angegeben werden:

Definition 4.23 (***Event***)
Ein Objekt e des Typs ***Event*** enthält folgende Elemente:

1. Eine Zahl s, die die Position auf der Parameterkurve $l.curve$ der Spur l angibt.

2. Einen Bezeichner $swmod$, den Namen des Softwaremoduls, das die Argumente verarbeiten soll, wenn e überfahren wird.

3. Ein Tupel $args$, die Argumente von e. Die Elemente von $args$ können dabei Bezeichner oder Zahlen aus $\mathbb{R}$ sein. •

4.4. Straßen

4.4.1. Überblick

Das **geometrische Modell von Straßen** besteht aus dem Lageplan (Abschnitt 4.4.2), d.h. dem Verlauf der Straßen in der Ebene, und dem Höhenplan (Abschnitt 4.4.3), der zu jedem Punkt auf der Straße eine Höhe festlegt. In Abschnitt 4.4.4 wird gezeigt, wie aus der formalen Beschreibung einer Straße gemäß Grammatik 3.12 (S. 45) die vorgestellte Repräsentation von Lage- und Höhenplan gewonnen wird.

4.4.2. Lageplan

Die Spurzellen einer Straße $c \triangleright \boldsymbol{Course}$ (gespeichert in $c.ChildNodes = \{lc_1, \ldots, lc_{n_c}\}$) werden als geordnete Menge interpretiert. Sie beschreiben den Verlauf der Straße von $c.p_1$, dem Anfang des Streckenstücks c, bis $c.p_2$, dem Ende von c. Dabei beziehen sich die Bezeichnungen "Anfang" bzw. "Ende" nur auf die Richtung, die aus der Beschreibung des Lageplans von c gemäß Grammatik 3.13 (S. 47) resultiert.

Definition 4.24 (Erweiterung von $\boldsymbol{Course}$)
Zusätzlich zu den bisher genannten Eigenschaften und Elementen von Objekten $c \triangleright \boldsymbol{Course}$ gilt:

1. Die Elemente in $c.ChildNodes = \{lc_1, \ldots, lc_{n_c}\}$ sind geordnet. Für $lc_i, lc_{i+1} \in c.ChildNodes$ $(1 \leq i \leq n_c - 1)$ gilt:

$$\begin{aligned} lc_i.p_2.Edges &= \{lc_{i+1}.p_1\} \\ lc_{i+1}.p_1.Edges &= \{lc_i.p_2\} \end{aligned}$$

2. Alle LaneCells von c haben die gleiche Anzahl von Spuren:

$$|lc_i.ChildNodes| = |lc_j.ChildNodes|$$

   für alle $lc_i, lc_j \in c.ChildNodes$. •

Die Tatsache, dass die Fahrbahnen von Straßen bei gleichbleibender Breite und relativer Höhe parallel verlaufen, kann für eine effiziente Repräsentation des

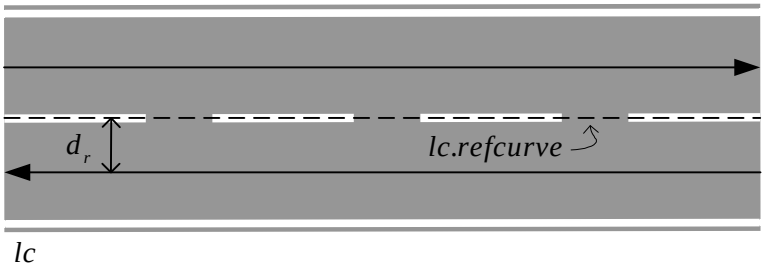

Abbildung 4.14.: Bei Straßen wird der Verlauf der Mitte in den Spurzelle lc gespeichert ($lc.refcurve$, gestrichelte Linie). Die Spuren (durchgezogene Linien) ergeben sich durch eine Parallelverschiebung dieser "Referenzkurve" um einem Abstand d_r, der vom Querschnittsprofil der Straße vorgegeben wird.

Lageplans genutzt werden. Anstatt jede Fahrbahn mit einer eigenen Parameterkurve zu beschreiben, werden sie als **Parallelverschiebung einer sog. Referenzkurve** definiert (Abbildung 4.14).
Die verschiedenen Typen von Referenzkurven werden vom Typ ***CourseCurve*** abgeleitet:

Definition 4.25 (***CourseCurve***)
Der Typ ***CourseCurve*** ist vom Typ ***Curve*** abgeleitet. Ein Objekt $curve \triangleright$ ***CourseCurve*** repräsentiert eine Parameterkurve, die den Verlauf der Mitte einer Straße beschreibt. Zusätzlich zu den Elementen und Eigenschaften von ***Curve*** gilt:

1. $curve$ enthält eine Markierung $invert \in \{0, 1\}$.
2. $curve$ enthält eine Zahl $offset \in \{0, S\}$.
3. $curve$ enthält einen Punkt p_o.
4. $curve$ enthält einen Winkel α_o.
5. Für die Funktion p muss gelten:
$$curve.p(offset) = p_o$$
6. Für die Funktion α muss gelten:
$$curve.alpha(offset) = \alpha_0$$

•

Über die Parameter p_o und α_o kann bei der geometrischen Instanzierung der Anfang ($offset = 0$) bzw. das Ende ($offset = S$) einer durch ein Objekt $curve \triangleright \boldsymbol{CourseCurve}$ repräsentierten Referenzkurve beliebig in der xy-Ebene positioniert und gedreht werden. Diese Parameter werden von Algorithmus 5.8 (S. 158) so berechnet, dass die Spuren von Spurzellen glatt aneinander schließen. Wenn die Spurzellen lc_1 und lc_2 zweier Straßen über die gleichen Ports $lc_1.p_i$ und $lc_2.p_i$ $(i = 1, 2)$ verknüpft sind, muss eine der Referenzkurven "rückwärts" durchlaufen werden. Dies wird durch $refcurve.invert = 1$ angezeigt. Auch dieser Wert wird bei der geometrischen Instanzierung von Algorithmus 5.8 berechnet.

Die Referenzkurven werden in den Spurzellen gespeichert:

Definition 4.26 (Erweiterung von ***CourseLaneCell***)
Zusätzlich zu den bisher genannten Eigenschaften und Elementen von Objekten $clc \triangleright \boldsymbol{CourseLaneCell}$ gilt:

1. clc hat zusätzlich einen Eintrag $refcurve \triangleright \boldsymbol{CourseCurve}$. Dabei ist $refcurve$ über die Bogenlänge parametrisiert.
2. Mit $clc.ChildNodes = \{l_1, \ldots, l_{n_c}\}$:

$$\begin{aligned} clc.p_1.Refinements &= \{l_1.p_1, \ldots, l_{n_c}.p_1\} \\ clc.p_2.Refinements &= \{l_1.p_2, \ldots, l_{n_c}.p_2\} \end{aligned}$$

•

Punkt 2 in obiger Definition ist in Abbildung 4.15 veranschaulicht.

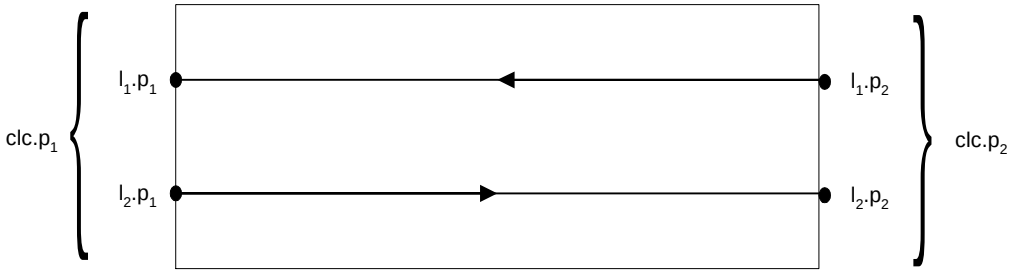

Abbildung 4.15.: Port-Verfeinerungen von Spurzellen von Straßen.

Dargestellt ist eine Spurzelle einer Straße. Ein Port p_i $(i = 1, 2)$ der Spurzelle hat als Verfeinerung genau die korrespondierenden Ports $l_j.p_i$ der Spuren.

Unter der Voraussetzung, dass sich die Gesamtbreite eines durch $clc \triangleright \boldsymbol{CourseLaneCell}$ repräsentierten Ausschnitts einer Straße durch

$$\sum_{i=1}^{clc.n_c} l_i.w \qquad \text{mit} \quad l_i \in clc.ChildNodes$$

berechnen lässt[7] , und unter der Annahme, dass $clc.refcurve$ genau in der Mitte der Straße verläuft, kann für jede Spur l_i ihr seitlicher Offset d_r zur Referenzkurve berechnet werden. Dabei ist $d_r < 0$, falls die Spur (bezogen auf die durch $l_i.p_1, l_i.p_2$ gegebene Richtung) links von der Referenzkurve verläuft und $d_r > 0$, falls sie rechts davon verläuft.

Zu einer Referenzkurve $refcurve$ mit dem Abstand $|d_r|$ parallel verlaufende Parameterkurven werden durch Objekte pc des Typs ***ParallelCurve*** dargestellt und in Objekten vom Typ ***CourseLane***, die eine Spur einer Straße repräsentieren, gespeichert:

Definition 4.27 (***CourseLane***)
Ein Objekt $cl \triangleright \boldsymbol{CourseLane}$ ist vom Typ ***Lane*** abgeleitet. Es gilt:

1. $cl.parent \triangleright \boldsymbol{CourseLaneCell}$.
2. Für $cl.p_i.Generalization = v.p$ gilt: $v \triangleright \boldsymbol{CourseLaneCell}$.
3. $cl.curve$ ist vom Typ ***ParallelCurve***. •

Abbildung 4.16 zeigt die Zusammenhänge in einem Klassendiagramm.

Die Gleichung für $pc.p$ eines Objektes $pc \triangleright \boldsymbol{ParallelCurve}$ lässt sich aus der zugehörigen Referenzkurve $refcurve$ herleiten. Schreibt man

$$refcurve.p'(s) = \begin{pmatrix} x'(s) \\ y'(s) \end{pmatrix}$$

dann erhält man für den Normalenvektor im Punkt s von $refcurve.p$:

$$\begin{pmatrix} -y'(s) \\ x'(s) \end{pmatrix}$$

[7] Dies bedeutet, dass sich die einzelnen Spuren l_i nicht überlappen und "nahtlos" aneinander schließen.

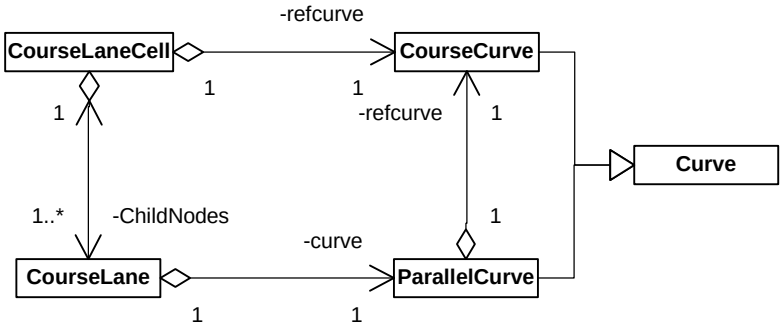

Abbildung 4.16.: Referenzkurven von Straßen.

Ein Punkt $q(s)$ auf der parallelen Kurve hat somit die Koordinaten

$$q(s) = refcurve.p(s) + d_r \begin{pmatrix} -y'(s) \\ x'(s) \end{pmatrix} \tag{4.1}$$

Hieraus ergibt sie die Gleichung für $pc.p$, wenn miteinbezogen wird, in welche Richtung die Referenzkurve durchlaufen wird. Dies wird in einer Markierung $pc.invert \in \{0, 1\}$ gespeichert. Dieser Wert wird bei der geometrischen Instanzierung der zu pc gehörenden Spur berechnet (s. Algorithmus 5.11, S. 164). Ist $pc.invert = 0$, dann muss die Referenzkurve zur Berechnung von $pc.p$ in normaler Richtung, also vom Parameter 0 bis zum Parameter $refcurve.S$ durchlaufen werden. Der Wert $pc.invert = 1$ bedeutet, dass die Referenzkurve "von hinten nach vorne", also vom Parameter $refcurve.S$ bis zum Parameter 0 zu durchlaufen ist. Für $pc.p$ ergibt sich also:

$$pc.p(s) = \begin{cases} q(s) & : \quad pc.invert = 0 \\ q(refcurve.S - s) & : \quad pc.invert = 1 \end{cases} \tag{4.2}$$

Selbst wenn die Referenzkurve über die Bogenlänge parametrisiert ist, gilt dies i.A. für $pc.p$ nicht mehr. Da sich üblicherweise die Längenangaben in realen Straßenplänen ebenfalls auf die Mitte der Straße beziehen, wird diese Einschränkung in Kauf genommen.

Auch beim Tangentenwinkel wird die Durchlaufrichtung berücksichtigt:

$$pc.\alpha(s) = \begin{cases} refcurve.\alpha(s) & : \quad pc.invert = 0 \\ refcurve.\alpha(refcurve.S - s) - \pi & : \quad pc.invert = 1 \end{cases} \tag{4.3}$$

Aus Gleichung 4.1 ergibt sich, dass die durch die Punkte $refcurve.p(s)$ und $q(s)$ verlaufende Gerade senkrecht auf beiden Kurven steht. Folglich besitzen die beiden Kurven auch den gleichen Krümmungsmittelpunkt. Der Krümmungskreis von $refcurve.p(s)$ hat den Radius $\frac{1}{refcurve.\kappa(s)}$. Die im Abstand d_r parallel verlaufende Kurve hat dann den Krümmungsradius $\frac{1}{refcurve.\kappa(s)} - d_r$. Daraus erhält man die Krümmung von pc:

$$pc.\kappa(s) = \begin{cases} \frac{refcurve.\kappa(s)}{1-d_r\, refcurve.\kappa(s)} & : \quad pc.invert = 0 \\ & : \\ -\frac{refcurve.\kappa(refcurve.S-s)}{1-d_r\, refcurve.\kappa(refcurve.S-s)} & : \quad pc.invert = 1 \end{cases} \tag{4.4}$$

Insgesamt führt dies zu folgender Definition:

Definition 4.28 (***ParallelCurve***)
Ein Objekt $pc \triangleright$ ***ParallelCurve*** ist vom Typ ***Curve*** abgeleitet. Es gilt:

1. pc hat zusätzlich ein Element $refcurve \triangleright$ ***CourseCurve***. Es ist

$$pc.refcurve = pc.parent.refcurve$$

2. $pc.S = pc.refcurve.S$
3. pc enthält ein Element $d_r \in \mathbb{R}$.
4. pc enthält eine Markierung $invert \in \{0, 1\}$.
5. $pc.p$ ist wie in Gleichung 4.2 definiert.
6. $pc.\alpha$ ist wie in Gleichung 4.3 definiert.
7. $pc.\kappa$ ist wie in Gleichung 4.4 definiert. •

Die gerichteten Abstände $cl_i.curve.d_r$ von Spuren $cl_i \triangleright$ ***CourseLane*** können aus den Spurinformationen $cl_i.LaneInfo$ berechnet werden (vgl. Algorithmus B.1 im Anhang auf S. 216).

Der Rest dieses Abschnitts stellt die vier Typen von Referenzkurven von Straßen dar:

1. Geradenstücke (modelliert durch den Typ ***CStraightLine***)
2. Kreisbögen (modelliert durch den Typ ***CCircularArc***)

3. Übergangsbögen, die beim Kurveneintritt von Krümmung 0 auf eine Krümmung $\neq 0$ überführen (Typ ***CClothoid1***)

4. Übergangsbögen, die beim Kurvenaustritt von einer Krümmung $\neq 0$ auf die Krümmung 0 überführen (Typ ***CClothoid2***)

Abbildung 4.17 zeigt die Typen und ihre Vererbungshierarchie im Überblick.

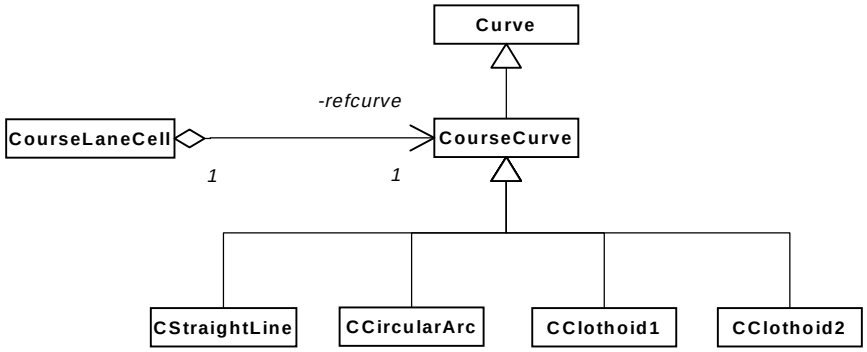

Abbildung 4.17.: Typen von Referenzkurven in Straßen.

Geradenstücke

Der Typ ***CStraightLine***, der eine **geradlinig verlaufenden Referenzkurve** repräsentiert, wird von ***CourseCurve*** abgeleitet. Damit ein Objekt $sl \triangleright$ ***CStraightLine*** den in Definition 4.25 gestellten Bedingungen genügt, wird die Geradengleichung $sl.p(s)$ in der Form

$$sl.p(s) = (1 - \frac{s}{S})p_1 + \frac{s}{S}p_2 \tag{4.5}$$

geschrieben. Offensichtlich handelt es sich dabei um eine Parametrisierung über die Bogenlänge. Zu bestimmen ist der Startpunkt p_1 und der Endpunkt p_2.
Für $sl.offset = 0$ erhält man:

$$p_1 = sl.p_o \qquad \text{und} \quad p_2 = sl.p_o + S \begin{pmatrix} \sin(-sl.\alpha_o) \\ \cos(sl.\alpha_o) \end{pmatrix} \tag{4.6}$$

Ist $sl.offset = S$, dann ergibt sich:

$$p_2 = sl.p_o \qquad \text{und} \quad p_1 = sl.p_o - S \begin{pmatrix} \cos(-sl.\alpha_o) \\ \sin(sl.\alpha_o) \end{pmatrix} \tag{4.7}$$

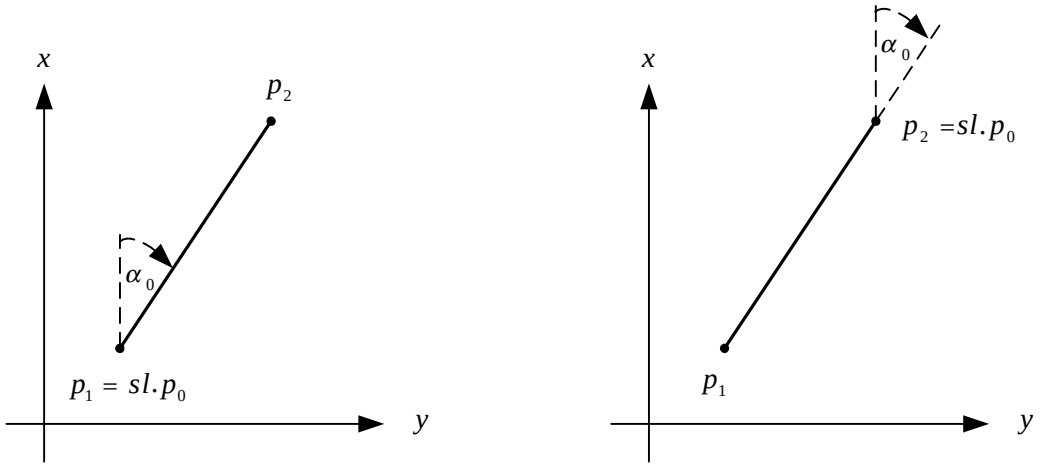

Abbildung 4.18.: Objekt sl des Typs ***CStraighLine***, links mit $sl.offset = 0$, rechts mit $sl.offset = S$.

Beide Fälle sind in Abbildung 4.18 verdeutlicht.
Der Tangentenwinkel $sl.\alpha(s)$ ist konstant:

$$sl.\alpha(s) = \alpha_o \tag{4.8}$$

Die Definition des Typs ***CStraightLine*** sieht damit wie folgt aus:

Definition 4.29 (***CStraightLine***)
Ein Objekt $sl \triangleright$ ***CStraightLine*** ist vom Typ ***CourseCurve*** abgeleitet. Es gilt:

1. sl hat zusätzlich zwei Punkte $sl.p_1 \in \mathbb{R}^2$ und $sl.p_2 \in \mathbb{R}^2$, die gemäß den Gleichungen 4.6 bzw. 4.7 berechnet werden.
2. $sl.p$ ist wie in Gleichung 4.5 definiert.
3. $sl.\alpha$ ist wie in Gleichung 4.8 definiert.
4. $sl.\kappa \equiv 0$ •

Kreisbögen

Gemäß RAS-L soll der Fahrer den längsten Teil von Kurven mit konstantem Lenkwinkel durchfahren können. Dies setzt voraus, dass der längste Teil einer Kurve eine konstante Krümmung aufweist. Der Verlauf der Fahrbahn hat in diesem Fall also die Form eines Kreisbogens mit Radius r. Vereinbarungsgemäß

ist bei Linkskurven $r < 0$ und bei Rechtskurven $r > 0$.

Der Kreisbogen wird von Objekten ca des Typs ***CCircularArc*** modelliert. In Parameterform besitzt der durch $ca.p(s)$ beschriebene Kreisbogen mit Radius r folgende Darstellung:

$$ca.p(s) = R(\varphi) \begin{pmatrix} r\cos(\frac{s}{r} - \frac{\pi}{2}) \\ r\left(1 + \sin(\frac{s}{r} - \frac{\pi}{2})\right) \end{pmatrix} + a \tag{4.9}$$

Dabei ist $R(\varphi) \in \mathbb{R}^{2\times 2}$ eine Matrix, die einen Vektor in der Ebene um einen Winkel φ rotiert:

$$R(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \tag{4.10}$$

Die Addition von $a \in \mathbb{R}^2$ bewirkt eine Verschiebung des Kreisbogens. Dass Gleichung 4.9 eine Bogenlängen Parametrisierung eines Kreisbogens darstellt, zeigt der folgende Satz:

Satz 4.1 Für jede Wahl von $\varphi \in \mathbb{R}$ und $a \in \mathbb{R}^2$ ist die durch Gleichung 4.9 definierten Kurve über die Bogenlänge parametrisiert.

Beweis: Gezeigt wird zunächst, dass Gleichung 4.9 im Falle $\varphi = 0$ und $a = (0,0)$ eine Bogenlängen Parametrisierung darstellt:

$$\begin{aligned} \| \frac{d}{ds} \begin{pmatrix} r\cos(\frac{s}{r} - \frac{\pi}{2}) \\ r\left(1 + \sin(\frac{s}{r} - \frac{\pi}{2})\right) \end{pmatrix} \| &= \| \begin{pmatrix} \cos(\frac{s}{r}) \\ \sin(\frac{s}{r}) \end{pmatrix} \| \\ &= \sqrt{\cos^2(\frac{s}{r}) + \sin^2(\frac{s}{r})} = 1 \end{aligned}$$

Hieraus und aus Satz B.3 (S. 214) folgt die Behauptung. □

Sowohl φ als auch a müssen so bestimmt werden, dass $ca.p$ die Bedingungen in Definition 4.25 erfüllt (vgl. hierzu auch Abbildung 4.19).

Im Falle $ca.offset = 0$ ist

$$\varphi = ca.\alpha_o \qquad \text{und} \quad a = ca.p_o \tag{4.11}$$

Wenn in Gleichung 4.11 $\varphi = 0$ und $a = (0,0)$ gesetzt wird, dann hat der Kreisbogen im Punkt $ca.p(S)$ den Tangentenwinkel $\frac{S}{r}$. Im Falle $ca.offset =$

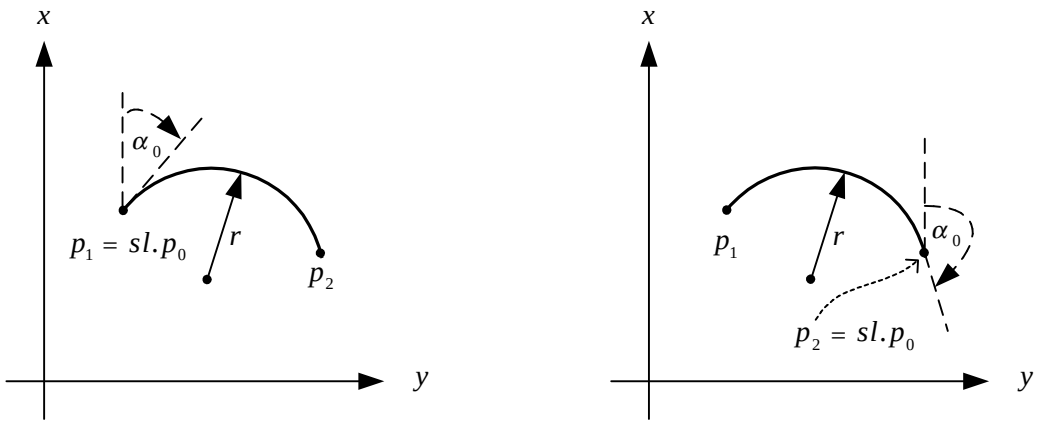

Abbildung 4.19.: Objekt ca des Typs ***CCircularArc***, links mit $ca.offset = 0$, rechts mit $ca.offset = S$.

S muss der Tangentenwinkel im Punkt $ca.p(S)$ der Kurve gleich $ca.\alpha_o$ sein. Folglich muss die Kurve um einen Winkel

$$\varphi = ca.\alpha_o - \frac{S}{r} \tag{4.12}$$

rotiert werden.
Analog lässt sich für $ca.offset = S$ die Verschiebung a bestimmen, damit $ca.p(S) = p_o$ erfüllt ist: Setzt man in Gleichung 4.11 den in Gleichung 4.12 bestimmten Wert für φ und $a = (0, 0)$, dann ist

$$ca.p(S) = R(ca.\alpha_o - \frac{S}{r}) \begin{pmatrix} r\cos(\frac{S}{r} - \frac{\pi}{2}) \\ r\left(1 + \sin(\frac{S}{r} - \frac{\pi}{2})\right) \end{pmatrix}$$

Damit also $ca.p(S) = p_o$ erfüllt ist, muss die Kurve um

$$a = p_o - R(ca.\alpha_o - \frac{S}{r}) \begin{pmatrix} r\cos(\frac{S}{r} - \frac{\pi}{2}) \\ r\left(1 + \sin(\frac{S}{r} - \frac{\pi}{2})\right) \end{pmatrix} \tag{4.13}$$

verschoben werden.
Damit ist $ca.p$ vollständig definiert.

Für den Tangentenwinkel des Kreisbogens an einer Stelle s gilt:

$$ca.\alpha(s) = \frac{s}{r} + ca.\alpha_o \tag{4.14}$$

Der Kreisbogen hat eine konstante Krümmung:

$$ca.\kappa(s) = \frac{1}{r} \tag{4.15}$$

Zusammenfassend kann der Typ ***CCircularArc*** wie folgt definiert werden:

Definition 4.30 (***CCircularArc***)
Ein Objekt $ca \triangleright$ ***CCircularArc*** ist vom Typ ***CourseCurve*** abgeleitet. Es gilt:

1. ca hat zusätzlich eine Zahl $r \in \mathbb{R}$, den Radius des Kreisbogens.
2. $ca.p$ ist wie in Gleichung 4.9 definiert.
3. $ca.\alpha$ ist wie in Gleichung 4.14 definiert.
4. $ca.\kappa$ ist wie in Gleichung 4.15 definiert. •

Übergangsbögen mit zunehmender Krümmung

Wäre im Lageplan ein Kreisbogen direkt an ein Geradenstück angeschlossen, dann würde sich die Krümmung sprunghaft von 0 (Geradenstück) auf eine Krümmung $\neq 0$ im Kreisbogen ändern. Für eine korrekte Spurhaltung müsste in diesem Fall der Fahrer den Lenkwinkel sprunghaft ändern, was in der Realität nicht möglich ist. Deshalb schreiben die RAS-L zwischen Streckenstücken mit unterschiedlicher konstanter Krümmung (z.B. Geraden und Kreisbögen) sog. Übergangsbögen vor. Diese haben die Aufgabe, die unterschiedlichen **Krümmungen langsam ineinander zu überführen**.

Zur Herleitung der Parameterkurve eines Übergangsbogens wird angenommen, dass die Krümmung proportional zum zurückgelegten Weg zunimmt. Wird also mit s die Bogenlänge auf dem Übergangsbogen bezeichnet und mit $\kappa(s)$ die Krümmung in diesem Punkt, dann lautet die Bedingung:

$$\frac{s}{\kappa(s)} = A^2$$

Für den Tangentenwinkel $\alpha(s)$ an der Stelle s gilt damit:

$$\frac{d\alpha}{ds}(s) = \kappa(s) = \frac{s}{A^2}$$

Die Integration dieser Gleichung liefert:

$$\alpha(s) = \int_0^s \frac{s}{A^2} ds = \frac{s^2}{2A^2} \tag{4.16}$$

Weiterhin gilt:

$$\begin{aligned} \frac{dx}{ds}(s) &= \cos\alpha(s) \\ \frac{dy}{ds}(s) &= \sin\alpha(s) \end{aligned}$$

Integrieren und Einsetzen von Gleichung 4.16 ergibt:

$$x(s) = \int_0^s \cos\alpha(s)ds = \int_0^s \cos\frac{s^2}{2A^2}ds \tag{4.17}$$

$$y(s) = \int_0^s \sin\alpha(s)ds = \int_0^s \sin\frac{s^2}{2A^2}ds \tag{4.18}$$

Bei den Integralen in obiger Gleichung handelt es sich um die sog. Fresnel-Integrale (vgl. [8]). Die dadurch beschriebene Parameterkurve heißt "Klothoide" oder "Cornu'sche Spirale" (Abbildung 4.20).

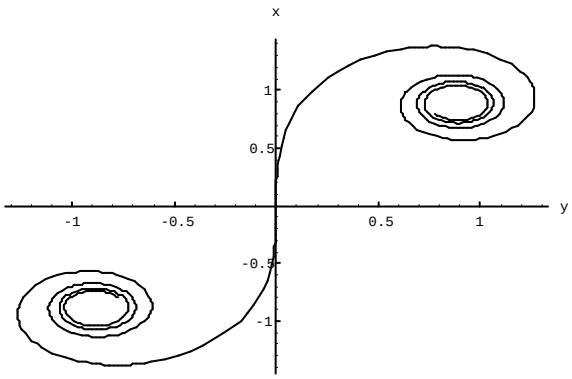

Abbildung 4.20.: Klothoide für $s \in [-7, 7]$ und $A = 1$

Im Lageplan von Straßen gibt es **zwei Typen von Übergangsbögen**: Solche, die die Krümmung 0 auf eine betragsmäßig größere überführen (z. B. die einer Geraden vom Typ ***CStraightLine***, Krümmung 0, auf die eines Kreisbogens vom Typ ***CCircularArc***, Krümmung $\neq 0$). Diese werden durch Objekte *cl1* des Typs ***CClothoid1*** modelliert. Der zweite Typ leitet von einer betragsmäßig größeren Krümmung auf die Krümmung 0 über. Die entsprechenden Objekte *cl2* sind vom Typ ***CClothoid2***. In diesem Abschnitt wird eine mit der Definition 4.25 konforme Herleitung des Typs ***CClothoid1*** beschrieben. Der Typ ***CClothoid2*** folgt im nächsten Abschnitt.

Für $cl1.p$ wird hierzu analog zu den Kreisbögen im vorhergehenden Abschnitt und unter Verwendung der Gleichungen 4.17 und 4.18 der Ansatz

$$cl1.p(s) = R(\varphi) \begin{pmatrix} \int_0^s \cos \frac{s^2}{2A^2} ds \\ \operatorname{sgn} r \int_0^s \sin \frac{s^2}{2A^2} ds \end{pmatrix} + a \tag{4.19}$$

gemacht. Die Matrix $R(\varphi) \in \mathbb{R}^{2\times 2}$ ist die durch Gleichung 4.10 definierte Rotationsmatrix, der Vektor $a \in \mathbb{R}^2$ bewirkt eine Verschiebung. Die Zahl $r \in \mathbb{R}$ ist der Radius des Kreisbogens, auf den $cl1$ überblenden soll. Die Krümmung von $cl1$ verläuft also von 0 für $s = 0$ bis $\frac{1}{r}$ für $s = S$.

Zunächst wird gezeigt, dass es sich bei Gleichung 4.19 um eine Parametrisierung über die Bogenlänge handelt:

Satz 4.2 Für jede Wahl von $\varphi \in \mathbb{R}$ und $a \in \mathbb{R}^2$ ist die durch Gleichung 4.19 definierte Kurve über die Bogenlänge parametrisiert.

Beweis: Für $\varphi = 0$ und $a = (0, 0)$ handelt es sich bei Gleichung 4.19 um eine Bogenlängen Parametrisierung:

$$\begin{aligned} \| \frac{d}{ds} \begin{pmatrix} \int_0^s \cos \frac{s^2}{2A^2} ds \\ \operatorname{sgn} r \int_0^s \sin \frac{s^2}{2A^2} ds \end{pmatrix} \| &= \| \begin{pmatrix} \cos(\frac{s^2}{2A^2}) \\ \operatorname{sgn} r \sin(\frac{s^2}{2A^2}) \end{pmatrix} \| \\ &= \sqrt{\cos^2(\frac{s^2}{2A^2}) + \sin^2(\frac{s^2}{2A^2})} = 1 \end{aligned}$$

Mit Satz B.3 folgt die Behauptung. □

Die Bestimmung der Parameter φ und a erfolgt wieder getrennt für die Fälle $cl1.offset = 0$ und $cl1.offset = S$ (vgl. hierzu Abbildung 4.21).

Für $cl1.offset = 0$ ist

$$\varphi = cl1.\alpha_o \qquad \text{und} \quad a = ca.p_o \tag{4.20}$$

Im Fall $cl1.offset = S$ werden die Parameter φ und a wie folgt ermittelt: Setzt man in Gleichung 4.19 $\varphi = 0$ und $a = (0, 0)$, dann beträgt der Tangentenwinkel an der Stelle S gem. Gleichung 4.16 und unter Einbeziehung der Kurvenrichtung $\operatorname{sgn} r \frac{S^2}{2A^2}$. Damit der Tangentenwinkel in diesem Punkt, wie in Definition 4.25 gefordert, den Wert $cl1.\alpha_o$ hat, muss die Kurve um den Winkel

$$\varphi = cl1.\alpha_o - \operatorname{sgn} r \frac{S^2}{2A^2} \tag{4.21}$$

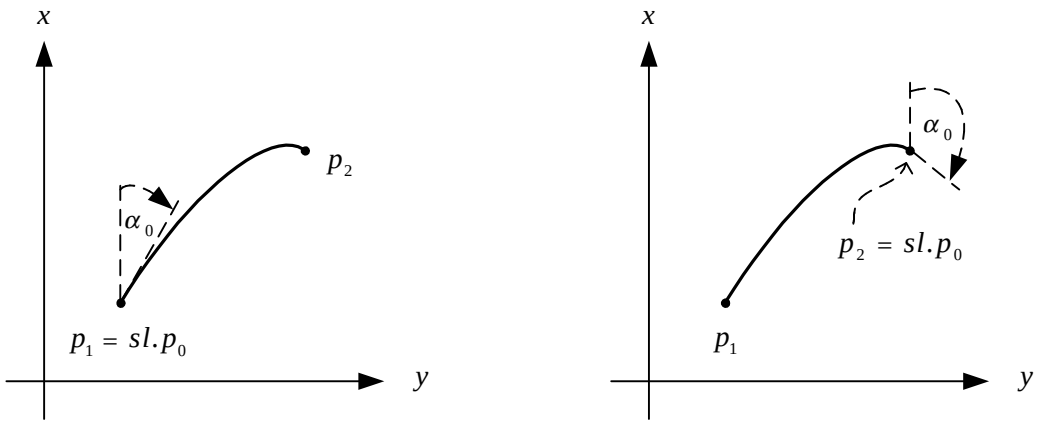

Abbildung 4.21.: Objekt $cl1$ des Typs ***CClothoid1***, links mit $cl1.offset = 0$, rechts mit $cl1.offset = S$.

gedreht werden.
Zur Bestimmung von a werden in Gleichung 4.19 der in Gleichung 4.21 berechnete Wert für φ, $a = (0, 0)$ und $s = S$ eingesetzt:

$$cl1.p(s) = R(cl1.\alpha_o - \operatorname{sgn} r \frac{S^2}{2A^2}) \begin{pmatrix} \int_0^s \cos \frac{s^2}{2A^2} ds \\ \operatorname{sgn} r \int_0^s \sin \frac{s^2}{2A^2} ds \end{pmatrix}$$

Folglich wird durch eine Verschiebung um

$$a = cl1.p_o - R(cl1.\alpha_o - \operatorname{sgn} r \frac{S^2}{2A^2}) \begin{pmatrix} \int_0^s \cos \frac{s^2}{2A^2} ds \\ \operatorname{sgn} r \int_0^s \sin \frac{s^2}{2A^2} ds \end{pmatrix} \tag{4.22}$$

die Bedingung $cl1.p(S) = cl1.p_o$ erfüllt.

Nach Gleichung 4.16 und unter Berücksichtigung der zusätzlichen Rotation von φ sowie der Kurvenrichtung gilt für den Tangentenwinkel:

$$cl1.\alpha(s) = \operatorname{sgn} r \frac{s^2}{2A^2} + cl1.\alpha_o \tag{4.23}$$

Gemäß der obigen Konstruktion der Klothoide und wiederum unter Berücksichtigung der Kurvenrichtung errechnet sich die Krümmung zu:

$$ca.\kappa(s) = \operatorname{sgn} r \frac{s}{A^2} \tag{4.24}$$

Damit kann der Typ ***CClothoid1*** definiert werden:

Definition 4.31 (***CClothoid1***)
Ein Objekt $cl1 \triangleright$ ***CClothoid1*** ist vom Typ ***CourseCurve*** abgeleitet. Es gilt:

1. $cl1$ hat zusätzlich eine Zahl $A \in \mathbb{R}$.
2. $cl1.p$ ist wie in Gleichung 4.19 definiert.
3. $cl1.\alpha$ ist wie in Gleichung 4.23 definiert.
4. $cl1.\kappa$ ist wie in Gleichung 4.24 definiert. •

Übergangsbögen mit abnehmender Krümmung

Wie bereits im vorherigen Abschnitt erwähnt, müssen Objekte $cl2$ des Typs ***CClothoid2*** eine Krümmung $\neq 0$ auf die Krümmung 0 überführen. Dies leistet eine "rückwärts" durchlaufene Klothoide. Mit der Abkürzung

$$q(s) = \begin{pmatrix} \int_0^{S-s} \cos \frac{s^2}{2A^2} ds \\ -\text{sgn}\, r \int_0^{S-s} \sin \frac{s^2}{2A^2} ds \end{pmatrix}$$

führt das zu folgendem Ansatz:

$$cl2.p(s) = R(\varphi)(q(s) - q(0)) + a \tag{4.25}$$

Wiederum beschreibt $R(\varphi) \in \mathbb{R}^{2\times 2}$ eine Rotation mit Winkel φ und $a \in \mathbb{R}^2$ eine Verschiebung. Die Krümmung von $cl2.p$ verläuft von $\frac{1}{r}$ für $s = 0$ bis 0 für $s = S$.

Satz 4.3 Für jede Wahl von $\varphi \in \mathbb{R}$ und $a \in \mathbb{R}^2$ ist die durch Gleichung 4.25 definierte Kurve über die Bogenlänge parametrisiert.

Beweis: Für $\varphi = 0$ und $a = (0, 0)$ handelt es sich bei Gleichung 4.25 um eine Bogenlängen Parametrisierung:

$$\begin{aligned} \| \frac{d}{ds}(q(s) - q(0)) \| &= \|q'(s)\| \\ &= \| \frac{d}{ds} \begin{pmatrix} \int_0^{S-s} \cos \frac{s^2}{2A^2} ds \\ -\text{sgn}\, r \int_0^{S-s} \sin \frac{s^2}{2A^2} ds \end{pmatrix} \| \\ &= \| \begin{pmatrix} \cos(\frac{(S-s)^2}{2A^2}) \\ -\text{sgn}\, r \sin(\frac{(S-s)^2}{2A^2}) \end{pmatrix} \| \\ &= \sqrt{\cos^2(\frac{(S-s)^2}{2A^2}) + \sin^2(\frac{(S-s)^2}{2A^2})} = 1 \end{aligned}$$

Mit Satz B.3 folgt die Behauptung. □

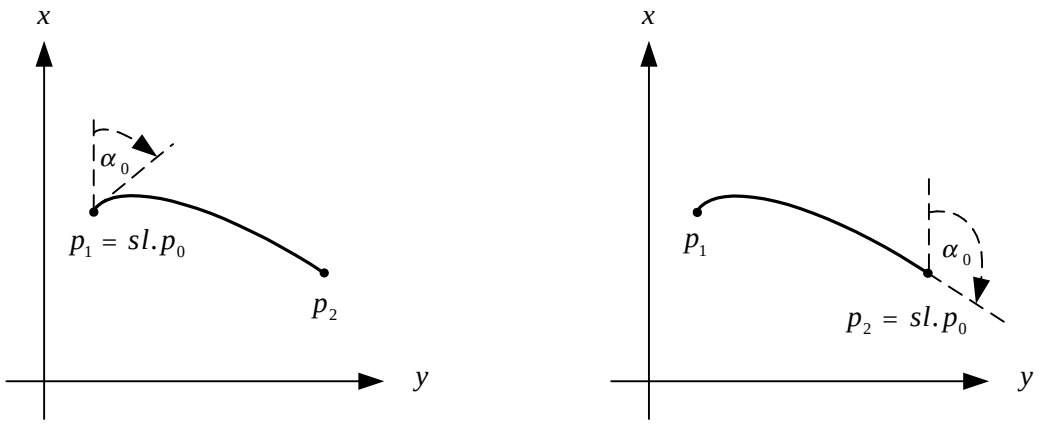

Abbildung 4.22.: Objekt $cl2$ des Typs ***CClothoid2***, links mit $cl2.offset = 0$, rechts mit $cl2.offset = S$.

Werte von φ und a, die mit der Definition 4.25 konform sind, werden so bestimmt (vgl. hierzu auch Abb. 4.22):

Fall $cl2.offset = 0$:
Setzt man in Gleichung 4.25 $\varphi = 0$ und $a = 0$, dann beträgt im Punkt $cl2.p(0)$ der Tangentenwinkel $\operatorname{sgn} r \frac{S^2}{2A^2} - \pi$. Deswegen wird mit

$$\varphi = cl2.\alpha_0 - (\operatorname{sgn} r \frac{S^2}{2A^2} - \pi) \tag{4.26}$$

erreicht, dass, wie in Definition 4.25 gefordert, der Tangentenwinkel im Punkt $cl2.p(0)$ den Wert $cl2.\alpha_0$ hat. Mit

$$a = cl2.p_o \tag{4.27}$$

gilt dann auch $cl2.p(0) = cl2.p_o$.

Fall $cl2.offset = S$:
Für $\varphi = 0$ und $a = 0$ hat $cl2.p(S)$ den Tangentenwinkel $-\pi$. Damit der Tangentenwinkel an der Stelle S den Wert $cl2.\alpha_0$ hat, muss

$$\varphi = cl2.\alpha_0 + \pi \tag{4.28}$$

gesetzt werden. Für diesen Wert von φ und mit $a = 0$ gilt:

$$cl2.p(S) = R(cl2.\alpha_0 + \pi)(q(S) - q(0))$$

Wird also

$$a = cl2.p_o - R(cl2.\alpha_0 + \pi)(q(S) - q(0)) \tag{4.29}$$

gewählt, dann gilt, wie in Definition 4.25 verlangt, $cl2.p(S) = p_o$.

Für den Tangentenwinkel an der Stelle s gilt:

$$cl2.\alpha(s) = \operatorname{sgn} r \frac{(S-s)^2}{2A^2} + cl2.\alpha_o - \pi \tag{4.30}$$

Die Krümmung im Punkt s errechnet sich so:

$$cl2.\kappa(s) = \operatorname{sgn} r \frac{S-s}{A^2} \tag{4.31}$$

Insgesamt wird der Typ ***CClothoid2*** wie folgt definiert:

Definition 4.32 (***CClothoid2***)
Ein Objekt $cl2 \triangleright$ ***CClothoid2*** ist vom Typ ***CourseCurve*** abgeleitet. Es gilt:

1. $cl2$ hat zusätzlich eine Zahl $A \in \mathbb{R}$.
2. $cl2.p$ ist wie in Gleichung 4.25 definiert.
3. $cl2.\alpha$ ist wie in Gleichung 4.30 definiert.
4. $cl2.\kappa$ ist wie in Gleichung 4.31 definiert. •

4.4.3. Höhenplan

In den vorhergehenden Abschnitten wurde der Verlauf der Fahrbahnen von Straßen in der x-y-Ebene beschrieben. Im Folgenden wird diese zweidimensionale Repräsentation um die Dimension der z-Achse erweitert, so dass **jedem Punkt auf einer Spur eine Höhe zugeordnet** werden kann. Diese Höhe setzt sich zusammen aus der relativen Höhe der Spur, wie sie vom Querschnittsprofil vorgegeben wird (vgl. Grammatik 3.21, S. 58, und Definition 4.20, S. 85) sowie einem Höhenverlauf, der jedem Punkt auf den Referenzkurven der Spurzellen eine Höhe zuordnet. Demnach haben alle zu den Referenzkurven parallel verlaufende Spuren in diesem Punkt die Höhe der Referenzkurve plus ihre relative Höhe.

Zunächst wird ein Grundtyp ***HeightProfile*** für Funktionen definiert, die Höhenprofile von Straßen repräsentieren. Während die den Lageplan

definierenden Referenzkurven in jeder Spurzelle einer Straße gespeichert sind, wird der Höhenplan Spurzellen übergreifend in den Objekten des Typs ***Course*** abgelegt. Hierzu wird die Definition dieses Typs entsprechend ergänzt. Danach wird als konkretes Beispiel für ein Höhenprofil das von der RAS-L vorgeschlagene beschrieben. Dabei handelt es sich um das Modell für den durch Grammatik 3.14 (S. 49) beschriebenen Höhenplan. Die Einführung einer Grundklasse für Höhenprofile soll eine spätere Erweiterung des Modells um andere Arten von Höhenprofilen ermöglichen. Denkbar wären hier beispielsweise Spline-Kurven, mit denen Höhenverläufe realer Straßen nachgebildet werden können, die nicht nach den RAS-L entworfen sind.

Sei c ein Objekt des Typs $Course$. Der Course c habe die Spurzellen $\{lc_1, \ldots, lc_{n_c}\}$. Diese Spurzellen bestimmen in ihrer Abfolge den Lageplan von c. Die gesamte Länge S_c der durch c repräsentierten Straße setzt sich demnach aus den Längen der Referenzkurven der Spurzellen zusammen:

$$S_c = \sum_{i=1}^{n_c} lc_i.refcurve.S$$

Stellt man sich den Lageplan von c als eine einzige Parameterkurve vor, dann verläuft deren Parameter s_c von 0 bis S_c. Jedem Wert von s_c kann eindeutig eine Spurzelle $lc_i \in c.ChildNodes$ und ein Parameter s auf der Referenzkurve von lc_i zugeordnet werden und umgekehrt. Diese Umrechnung leiste folgende Abbildung (und ihre Inverse):

$$convert(s_c) = (lc_i, s)$$

mit $s_c \in [0, S_c]$, $lc_i \in c.ChildNodes$ und

$$s \in \begin{cases} [0, lc_i.S) & : \quad 0 \leq i < n_c \\ [0, lc_i.S] & : \quad i = n_c \end{cases}$$

Aufgabe des Höhenprofils ist es, jedem Punkt an einer Stelle s_c auf c eine Höhe zuzuweisen:

Definition 4.33 (*HeightProfile*)
Ein Objekt $hp \triangleright$ ***HeightProfile*** enthält folgende Elemente:

1. Eine Zahl $S_c \in \mathbb{R}$, die gesamte Länge des Höhenprofils.
2. Eine Markierung $invert \in \{0, 1\}$.

3. Eine Zahl $\mathit{offset} \in \{0, S_c\}$.

4. Eine Zahl $h_o \in \mathbb{R}$.

5. Eine Funktion $h : [0, S_c] \to \mathbb{R}$, die jedem Wert von $s \in [0, S_c]$ eine Höhe zuweist. Es muss gelten:

$$hp.h(\mathit{offset}) = h_o$$

Ferner:

$$hp.h'(0) = hp.h'(S_c) = 0$$

•

Die Forderung $hp.h'(0) = hp.h'(S_c) = 0$ in obiger Definition verhindert, dass sich beim glatten Aneinanderhängen der Höhenprofile zweier Straßen Steigungen bzw. Gefälle aufsummieren.

Der Typ ***HeightProfile*** wird wie folgt in ***Course*** integriert:

Definition 4.34 (Erweiterung von ***Course***)
Zusätzlich zu den bisher genannten Eigenschaften und Elementen von Objekten $c \triangleright$ ***Course*** enthält c ein Objekt $hp \triangleright$ ***HeightProfile***. Es gilt:

$$hp.S_c = \sum_{i=1}^{n_c} lc_i.refcurve.S$$

wobei $lc_i \in c.ChildNodes$ und $n_c = |c.ChildNodes|$. •

Damit kann die Funktion $l.h$ aus Definition 4.21 (S. 85) für Spuren $l \triangleright$ ***CourseLane*** definiert werden:

Definition 4.35 (Erweiterung von ***CourseLane***)
Die Funktion $h(s)$ eines Objekts $l \triangleright$ ***CourseLane***, mit der die absolute Höhe an einer Stelle s auf der Parameterkurve der Spur gemessen wird, ist so definiert:

$$l.h(s) := c.hp.h(convert^{-1}(lc, s)) + l.h_{rel}(s)$$

falls $l.curve.invert = 0$ ist und

$$c.hp.h(convert^{-1}(lc, lc.refcurve.S - s)) + l.h_{rel}(lc.refcurve.S - s)$$

falls $l.curve.invert = 1$ ist. Dabei ist $lc := l.parent$ und $c := lc.parent$. •

Bei der Berechnung von $l.h(s)$ wird also die Durchlaufrichtung der Parameterkurve $l.curve$ einbezogen: Wenn zur Berechnung von $l.curve.p$ die Referenzkurve in umgekehrter Richtung zu durchlaufen ist ($l.curve.invert = 1$), dann muss auch die Höhe in umgekehrter Richtung abgefragt werden.

Die in den **RAS-L vorgeschriebenen Kurven für den Höhenplan** von Straßen setzen sich aus Segmenten mit konstanter Steigung bzw. konstantem Gefälle (d.h. konstanter Ableitung) und Übergängen zwischen diesen Segmenten zusammen (vgl. [36]). Die Übergangssegmente, auch Ausrundungen genannt, dienen zum glatten Überblenden der unterschiedlichen Ableitungen der an sie anschließenden Segmente. Aus Gründen der Fahrdynamik werden die Ausrundungen in Form von Parabel-Bögen konzipiert. Der Rest dieses Abschnitts beschreibt den Typ ***RASLHP***, der vom Typ ***HeightProfile*** abgeleitet ist und diese Art von Höhenprofil implementiert.

Ein einzelnes Segment des Höhenprofils vom Typ ***RASLHP*** wird durch die nachstehend definierten Typen ***RASLHPSegConst*** (Stück, bei dem Steigung/Gefälle konstant ist), ***RASLHPSegSag*** (Ausrundung in Form einer Wanne) bzw. ***RASLHPSegCrest*** (Ausrundung in Form einer Kuppe) modelliert. Da diese unterschiedlichen Segment-Typen einige Informationen gemeinsam haben, werden sie von einem Typ ***RASLHPSeg*** abgeleitet:

Definition 4.36 (***RASLHPSeg***)
Ein Objekt seg vom Typ ***RASLHPSeg*** enthält folgende Elemente:

1. Eine Zahl $S \in \mathbb{R}$, die die Länge der Strecke angibt, für die seg die Höhen angibt.
2. Eine Zahl s_c, die angibt, bei welchem Wert von s bezogen auf die Gesamtlänge der zugehörigen Straße dieses Segment beginnt[8].
3. Eine Zahl h_c, die die Höhe an der Stelle s_c der zugehörigen Straße angibt.
4. Eine Funktion $h : [s_c, s_c + S] \rightarrow \mathbb{R}$. •

Die **Typen der einzelnen Segmente** werden um die zusätzlich benötigten Informationen ergänzt:

[8] Das Segment seg endet folglich bei $seg.s_c + seg.S$

Definition 4.37 (***RASLHPSegConst***)
Ein Objekt c vom Typ ***RASLHPSegConst*** ist vom Typ ***RASLHPSeg*** abgeleitet. Zusätzlich enthält es eine Zahl $m \in \mathbb{R}$, die Steigung von h. Die Funktion h ist so definiert:

$$h(s) = h_c + m \cdot (s - s_c)$$

für $s \in [s_c, s_c + S]$. •

Definition 4.38 (***RASLHPSegSag***)
Ein Objekt sag vom Typ ***RASLHPSegSag*** ist vom Typ ***RASLHPSeg*** abgeleitet. Zusätzlich enthält es eine Zahl $R \in \mathbb{R}^+$, den Ausrundungshalbmesser von h, sowie eine Zahl $m \in \mathbb{R}$, die Steigung von h an der Stelle s_c. Für h gilt:

$$h(s) = h_c + m \cdot (s - s_c) + \frac{(s - s_c)^2}{2R}$$

mit $s \in [s_c, s_c + S]$. •

In Abbildung 4.23 sind die möglichen Ausrundungen durch Objekte $sag \triangleright$ ***RASLHPSegSag*** dargestellt.

Definition 4.39 (***RASLHPSegCrest***)
Ein Objekt sag vom Typ ***RASLHPSegCrest*** ist vom Typ ***RASLHPSeg*** abgeleitet. Zusätzlich enthält es eine Zahl $R \in \mathbb{R}^+$, den Ausrundungshalbmesser von h, sowie eine Zahl $min\mathbb{R}$, die Steigung von h an der Stelle s_c. Es ist

$$h(s) = h_c + m \cdot (s - s_c) - \frac{(s - s_c)^2}{2R}$$

für $s \in [s_c, s_c + S]$. •

Abbildung 4.24 illustriert die Arten von Ausrundungen durch Objekte $crest \triangleright$ ***RASLHPSegCrest***.
Folgende Definition zeigt, wie sich der Typ ***RASLHP*** aus den Segmenten zusammen setzt:

Definition 4.40 (***RASLHP***)
Der Typ ***RASLHP*** ist vom Typ ***HeightProfile*** abgeleitet. Für ein Objekt $raslhp \triangleright$ ***RASLHP*** gilt zusätzlich:

1. ***raslhp*** enthält ein Tupel $segments = (seg_1, \ldots, seg_{n_s})$. Dabei ist seg_i vom Typ ***RASLHPSeg***.

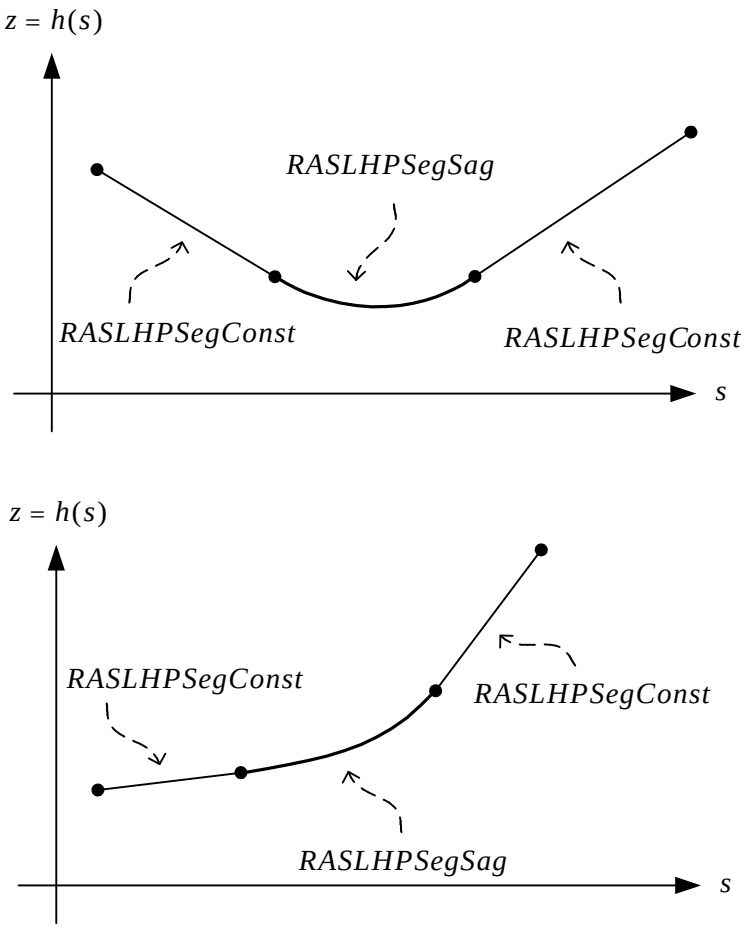

Abbildung 4.23.: Ausrundungen in Wannenform, oben eine "echte" Wanne, unten eine "unechte".

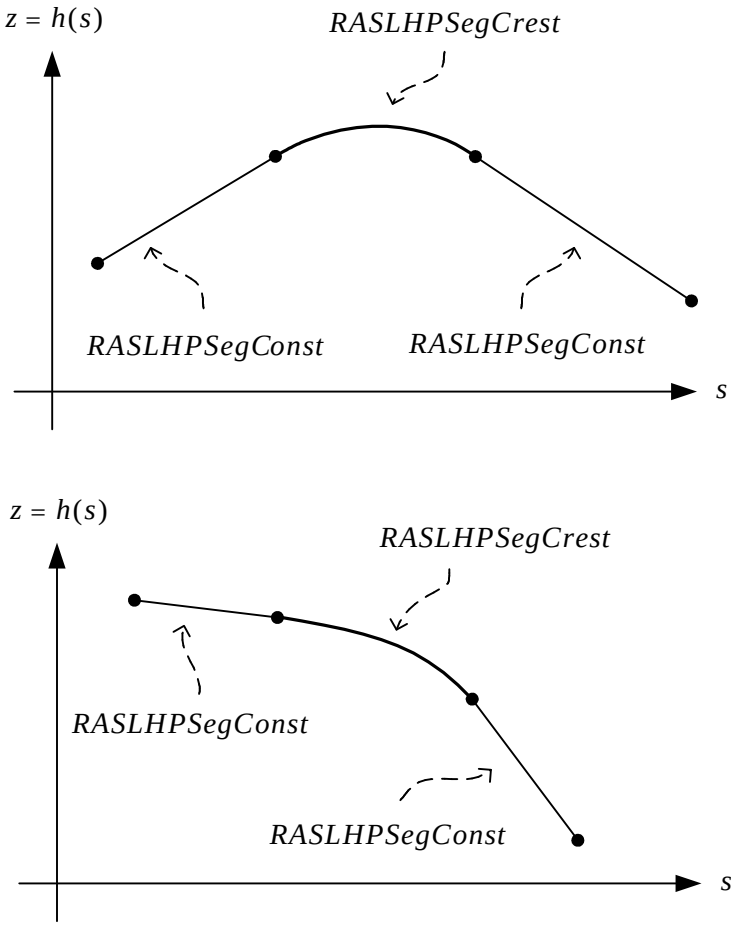

Abbildung 4.24.: Ausrundungen in Kuppenform, oben eine "echte" Kuppe, unten eine "unechte".

2. Die Segmente haben insgesamt die Länge des gesamten Höhenprofils:

$$\sum_{i=1}^{n_s} seg_i.S = S_c$$

3. Das erste und das letzte Segment haben keine Steigung bzw. kein Gefälle:

$$\begin{aligned} seg_1 \triangleright \boldsymbol{RASLHPSegConst} \quad &, \quad seg_1.m = 0 \\ seg_{n_s} \triangleright \boldsymbol{RASLHPSegConst} \quad &, \quad seg_{n_s}.m = 0 \end{aligned}$$

4. Zwischen zwei Segmenten mit konstanter Steigung bzw. konstantem Gefälle liegt eine Ausrundung, d.h. für $seg_i, seg_{i+2} \triangleright \boldsymbol{RASLHPSegConst}$ $(1 \leq i \leq n_s - 2)$ gilt:

$$\begin{aligned} seg_i.m < seg_{i+2}.m \quad &\Rightarrow \quad seg_{i+1} \triangleright \boldsymbol{RASLHPSegSag} \\ seg_i.m > seg_{i+2}.m \quad &\Rightarrow \quad seg_{i+1} \triangleright \boldsymbol{RASLHPSegCrest} \end{aligned}$$

5. Sei $s \in [0, S_c]$. Mit dem (eindeutig bestimmten) Index i, der die Bedingung

$$s \in \begin{cases} [seg_i.s_c, seg_i.s_c + seg_i.S) & : \quad 0 \leq i < n_s \\ [seg_i.s_c, seg_i.s_c + seg_i.S] & : \quad i = n_s \end{cases}$$

erfüllt, lässt sich $h(s)$ so berechnen:

$$h(s) = h_o + seg_i.h(s)$$

•

Die Zusammenhänge sind in Abbildung 4.25 dargestellt.

4.4.4. Generierung aus der formalen Beschreibung

Für jede Instanz einer Straße in der formalen Beschreibung wird ein Objekt $c \triangleright \boldsymbol{Course}$ erzeugt (vgl. Algorithmus 4.1, S. 83). Dabei werden die Informationen der zugehörigen Schablone, die gemäß Grammatik 3.12 definiert ist, in c übernommen. Dieser Abschnitt beschreibt, wie bei diesem Vorgang die RAS-L (vgl. [36]) überwacht werden.

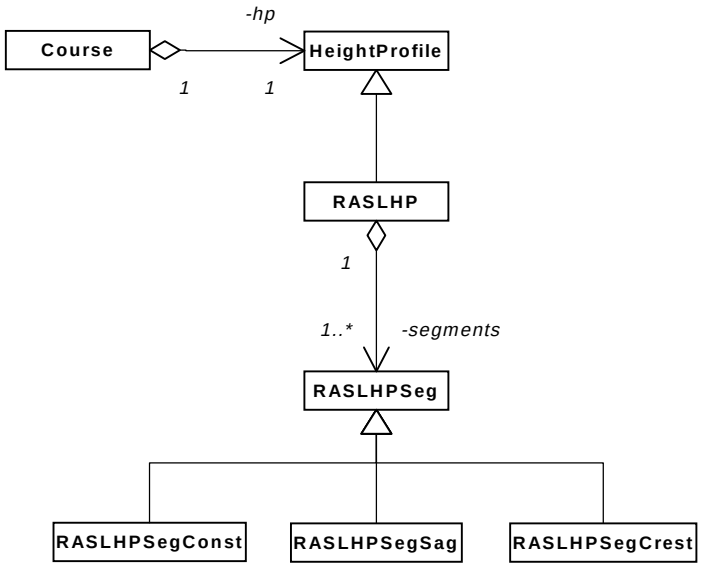

Abbildung 4.25.: Höhenprofile von Straßen.

RAS-L bei Höhenplänen

Bei der Beschreibung eines Höhenplans gem. Grammatik 3.14, S. 49, gibt der Anwender eine Folge von Segmenten mit konstanter Steigung an, deren Gesamtlänge die der Straße ist. Um daraus das Modell des Höhenprofils zu gewinnen, verlangen die RAS-L, dass zwischen den Segmenten mit konstanter Steigung Ausrundungen eingefügt werden. Die Ausrundungen müssen dabei bei der Gesamtlänge berücksichtigt werden.
Hat der Anwender diese Ausrundungen nicht spezifiziert, werden sie automatisch aus den RAS-L bezogen. Das im folgenden beschriebene Verfahren übernimmt diese Aufgabe. Es wird *einmal* für jede Straße bei der Erzeugung des Modells aus der formalen Beschreibung durchgeführt. Dies geschieht *vor der Simulation*.

Der Übersichtlichkeit wegen arbeitet das Verfahren in zwei Schritten: Im ersten Schritt werden aus den Eingaben die Gleichungen der Segmente mit konstanter Ableitung berechnet. Der zweite Schritt berechnet die Ausrundungen und verkürzt die Länge der Segmente mit konstanter Ableitung so, dass die für die Ausrundungen übrigbleibenden Längen den durch die RAS-L vorgegebenen entsprechen.

Tabelle 4.1.: Ausrundungshalbmesser gem. RAS-L (untere Schranken)

	$v_e = 50km/h$	60	70	80	90	100	120
Kuppe $[m]$	1400	2400	3150	4400	5700	8300	16000
Wanne $[m]$	500	750	1000	1300	2400	3800	8800

Mit der Bezeichnung $c.hp.segments = (seg_1, \ldots, seg_{n_s})$ hat das Verfahren also folgende Eingaben:

- Für alle $seg_i \triangleright$ ***RASLHPSegConst***: $seg_i.S$
  Länge des Streckenstücks der zugehörigen Straße, für die seg_i die Höhen definiert. Die Längen der Ausrundungen werden dabei auf 0 gesetzt.

- Für alle $seg_i \triangleright$ ***RASLHPSegConst***: $seg_i.m$
  Steigung/Gefälle von seg_i. In der formalen Beschreibung wird diese Angabe gem. Grammatik 3.14 (S. 49) in Prozent gemacht. Dabei bedeutet eine Steigung von q Prozent $(q > 0)$ bzw. ein Gefälle von q Prozent $(q < 0)$, dass sich auf einer Strecke von 100 Metern die Höhe um q Meter ändert. Folglich gilt:
  $$seg_i.m = \frac{q}{100}$$

- Für alle $seg_i \triangleright$ ***RASLHPSegSag*** oder ***RASLHPSegSag***: $seg_i.R$
  Entweder der Anwender hat diesen Wert bei der Beschreibung des Höhenplans angegeben oder er wird, je nach Straßenkategorie, automatisch in den RAS-L nachgeschlagen[9]. Tabelle 4.1 zeigt die Richtwerte für die Kategoriengruppe A.

- $seg_1.h_c = 0$
  Höhenoffset, bei dem das erste Segment der Straße beginnen soll.

Hieraus werden die Gleichungen der Segmente, wie in Algorithmus 4.2 angegeben, bestimmt.
Jetzt ist sicher zu stellen, dass ein mit dem Algorithmus 4.2 erzeugtes Höhenprofil den Bedingungen in Definition 4.33 genügt. Insbesondere betrifft dies die in 6. genannte Bedingung der Definition. Aufgrund der Forderung in Punkt 3. der Definition 4.40 gilt für $raslhp \triangleright$ ***RASLHP*** offensichtlich:

$$raslhp.h'(0) = raslhp.h'(ralshp.S_c) = 0$$

[9] Die Kategoriengruppen von Straßen sind in Abschnitt 3.4.3 (S. 57) erläutert.

Algorithmus 4.2 Bestimmung der Gleichungen von Segmenten eines Höhenprofils vom Typ ***RASLHP***

// 1. Schritt: Bestimme Gleichungen der konstanten Segmente
$l := 0$
$j := 1$
for all $seg_i \in raslhp.segments$, $seg_i \triangleright$ ***RASLHPSegConst*** **do**
  $seg_i.s_c := l$
  if $i > 1$ **then**
    $seg_i.h_c := seg_j.h_c + seg_j.h(seg_j.S)$
  end if
  $seg_i.h(s) := seg_i.m \cdot s$
  $j := i$
end for
// 2. Schritt: Bestimme Gleichungen der Ausrundungen
for all $seg_i \in raslhp.segments$, $seg_i \triangleright$ ***RASLHPSegConst*** **do**
  // Breite der Ausrundung
  $T := \frac{seg_i.R}{2}|seg_{i+1}.m - seg_{i-1}.m|$
  // konstante Segmente anpassen
  $seg_{i-1}.S := seg_{i_1}.S - T$
  $seg_{i+1}.s_c := seg_{i+1}.s_c + T$
  $seg_{i+1}.h_c := seg_{i+1}.h_c + seg_{i+1}.h(T)$
  // Ausrundung einpassen
  $seg_i.s_c := seg_{i-1}.s_c + seg_{i-1}.S$
  $seg_i.h_c := seg_{i-1}.h_c + seg_{i-1}.h(seg_{i-1}.S)$
end for

Im Falle $raslhp.offset = 0$ ist wie gefordert

$$raslhp.h(raslhp.offset) = raslhp.h_o$$

Auch für $raslhp.offset = raslhp.S_c$ kann dies mit Algorithmus 4.2 erreicht werden: Nach der Anwendung des Verfahrens beträgt die Höhe am Ende des Höhenprofis

$$h^\star := raslhp.h(raslhp.S_c)$$

Damit also $raslhp.h(S_c) = raslhp.h_o$ erfüllt ist, müssen die Segmente wie folgt angepasst werden:

for all $seg_i \in raslhp.segments$ **do**
  $seg_i.h_c := seg_i.h_c + (raslhp.h_o - h^\star)$
end for

RAS-L bei Lageplänen

Die in den RAS-L aufgeführten Richtlinien für Lagepläne werden auf zwei Arten eingesetzt: Zum einen wird der Anwender bei der Generierung des Modells aus der formalen Beschreibung gewarnt, wenn er eine Richtlinie verletzt hat. Zum anderen werden Parameter der Übergangsbögen automatisch berechnet, wenn sie der Anwender in Grammatik 3.13 (S. 47) nicht angegeben hat. Dieser Abschnitt skizziert die Verwendung der RAS-L für beide Arten.

Gem. RAS-L überwiegen bei **langen Geradenstücken** verkehrstechnische und optische Nachteile. Hierzu zählt beispielsweise, dass Geschwindigkeit und Abstand entgegenkommender Fahrzeuge schwerer einzuschätzen sind. Auch verleiten lange Geraden zu unangemessenen Geschwindigkeitssteigerungen. Aus der starken Blendwirkung entgegenkommender Fahrzeuge wird in der RAS-L eine maximale Geradenlänge $S_{G,max}$ (in Meter) abgeleitet:

$$S_{G,max} = 20 \cdot v_e$$

Dabei ist v_e die Entwurfsgeschwindigkeit, die der Anwender im Querschnitt der Straße gemäß Grammatik 3.21 (S. 58) angibt. Diese Blendwirkung tritt nur auf, wenn das Höhenprofil der gesamten Geraden eine konstante Steigung oder ein konstantes Gefälle aufweist.
Für alle Spurzellen lc mit $lc.refcurve \triangleright$ ***CStraightLine*** erhält der Anwender also eine Warnung, wenn

1. $lc.refcurve.S > S_{G,max}$ ist und

2. das gesamte Höhenprofil von $lc.refcurve$ durch ein Segment vom Typ ***RASLHPSegConst*** repräsentiert wird.

Die **minimalen Radien von Kreisbögen** in Kurven r_{min} (in Meter) lassen sich aus Grenzwerten der Fahrzeugphysik ableiten:

$$r_{min} = \frac{v_e^2}{127 \cdot (f_{r,max}(v_e) \cdot n + q)}$$

Die Funktion $f_{r,max}(v)$ berechnet in Abhängigkeit von der Geschwindigkeit den maximalen radialen Kraftschlussbeiwert. Er ist ein Maß für die Stärke von Adhäsion, Reibung und Verzahnung zwischen den Oberflächen von Reifen eines Fahrzeugs und Fahrbahn. In [32] wird hierfür eine empirisch ermittelte Formel angegeben. Dieser Kraftschlussbeiwert soll in der Kurve zu n Prozent ausgenutzt werden. Je nach Kategoriengruppe der Straße (vgl. Tabelle 3.1, S. 59) beträgt dieser Wert $n = 30$ (Kategoriengruppen A und B) bzw. $n = 50$ (restliche Gruppen). Der Parameter q bezeichnet die Querneigung der Straße in Prozent gegen die Straßenmitte. Solche Querneigungen werden aus Entwässerungsgründen eingeführt, sind aber im Modell nicht berücksichtigt. Deswegen wird $q = 0$ gesetzt.
Der Anwender wird also bei allen Spurzellen lc mit $lc.refcurve \triangleright$ ***CCircularArc*** gewarnt, wenn er

$$|lc.refcurve.r| < r_{min}$$

definiert hat.

Die **Klothoidenlängen** in Kurven, die wie im vorliegenden Modell aus der Folge Übergangsbogen - Kreisbogen - Übergangsbogen aufgebaut sind, lassen sich aus einer unteren Schranke A_{min} für den Klothoidenparameter A (vgl. Gleichungen 4.17 und 4.18, S. 99) und Grenzwerten der Fahrphysik herleiten. Die Klothoiden am Kurveneintritt und -austritt dürfen eine Länge von

$$S_{K,min} = \max\{\frac{A_{min}^2}{|r|}, 0.75\sqrt{|r| \cdot v_e}\}$$

nicht unterschreiten. Dabei ist r der Radius des Kreisbogens der Kurve, v_e die Entwurfsgeschwindigkeit der Straße und $A_{min} = |r|/3$.
Wenn der Anwender in der Beschreibung des Lageplans gemäß Grammatik 3.13 (S. 47) keine Längen für die Übergangsbögen angegeben hat, dann werden sie

auf $S_{k,min}$ gesetzt, damit der Kreisbogen der Kurve möglichst lang ist. Hat er die Längen angegeben, dann wird er gewarnt, falls er

$$lc.refcurve.S < S_{K,min}$$

gewählt hat. Dabei ist lc eine Spurzelle, für die $lc.refcurve$ den Typ ***CClothoid1*** oder ***CClothoid2*** hat.

Für die **Länge von Kreisbögen** gibt es eine Richtlinie, nach der in einer Kurve mindestens 2 Sekunden konstanter Kreisfahrt mit der Entwurfsgeschwindigkeit möglich sein sollen. Der Anwender erhält also eine Warnung, wenn

$$lc.refcurve.S < 2 \cdot \frac{v_e}{3.6}$$

erfüllt ist, wobei die Spurzelle lc einen Kreisbogen repräsentiert ($lc.refcurve \triangleright$ ***CCircularArc***).

4.5. Verkehrsknoten

4.5.1. Überblick

Wie bei Straßen wird auch bei Verkehrsknoten im geometrischen Modell zwischen Lageplan und Höhenplan unterschieden. Der Beschreibung von Lage- bzw. Höhenplan in den Abschnitten 4.5.3 bzw. 4.5.4 sind einige Besonderheiten, die Ports von Verkehrsknoten aufweisen (Abschnitt 4.5.2), vorangestellt. In Abschnitt 4.5.5 wird gezeigt, wie aus der formalen Beschreibung eines Verkehrsknotens gemäß Grammatik 3.15 (S. 50) die vorgestellte Repräsentation von Lage- und Höhenplan gewonnen wird.

4.5.2. Ports

Das Gebiet, das durch einen Verkehrsknoten, modelliert durch ein Objekt a vom Typ ***Area***, abgedeckt wird, ist ein **Rechteck** mit Breite $a.w$ und Länge $a.l$. Bei der formalen Beschreibung wird der Verkehrsknoten zunächst in einem Koordinatensystem dargestellt, dessen Ursprung in der Mitte des zugrundeliegenden Rechtecks liegt (vgl. Grammatik 3.15, S. 50). Dieses Koordinatensystem wird

im folgenden als "Entwurfskoordinatensystem" bezeichnet. Es hat die Achsen x_E und y_E.
Die **Menge der Ports** $a.Ports$ wird **in 4 Mengen partitioniert**: $a.Ports_s$ enthält die Ports, die auf Seite $s \in \{top, bottom, left, right\}$ des Rechtecks von a liegen. Dies entspricht der Beschreibung in Grammatik 3.16 (S. 51).

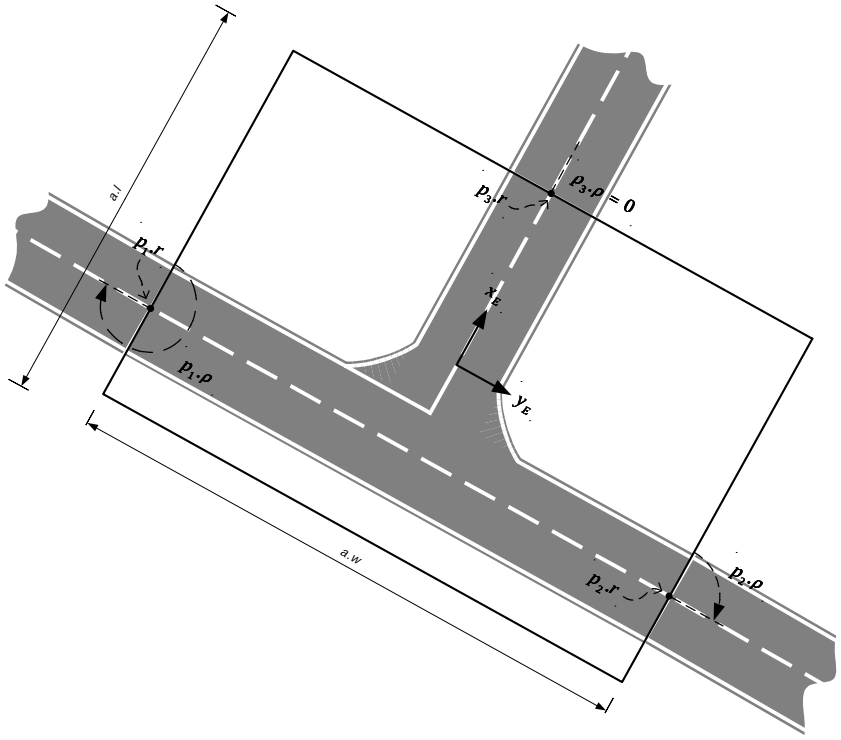

Abbildung 4.26.: T-Kreuzung nach der geometrischen Instanzierung. An jeden Port schließt eine Straße an (angedeutet). In der Mitte der T-Kreuzung ist das Entwurfskoordinatensystem mit den Achsen x_E und y_E eingezeichnet.

Jeder Port p_i von a kann mit einem Punkt $p_i.r \in \mathbb{R}^2$ assoziiert werden, dem Referenzpunkt des Ports. Nach der geometrischen Instanzierung liegt dieser Punkt genau in der Mitte des Querschnitts der Straße, die an p_i anschließt. Analog gehört zu p_i ein Referenzwinkel $p_i.\rho$, der angibt, welchen Winkel zur x_E-Achse von dem Port wegführende Spuren haben. Der Sachverhalt ist in Abbildung 4.26 am Beispiel einer T-Kreuzung dargestellt[10].

[10] Die formale Beschreibung dieser T-Kreuzung findet sich in Beispiel A.1 auf Seite 203.

Beim Entwurf von a wird angegeben, um wie viel ein Port p_i gegenüber der Mitte der Seite, auf der er liegt, verschoben ist (vgl. Grammatik 3.16, S. 51). Diese Verschiebung ist in $p_i.d$ gespeichert. Daraus lässt sich der zugehörige Referenzpunkt $p_i.r$ sowie der Winkel $p_i.\rho$ berechnen:

$$\begin{aligned}
p_i \in a.Ports_{top} &: p_i.r = (a.l/2, p_i.d)^T \\
&\quad p_i.\rho = 0 \\
p_i \in a.Ports_{bottom} &: p_i.r = (-a.l/2, p_i.d)^T \\
&\quad p_i.\rho = \pi \\
p_i \in a.Ports_{left} &: p_i.r = (p_i.d, -a.w/2)^T \\
&\quad p_i.\rho = -\pi/2 \\
p_i \in a.Ports_{right} &: p_i.r = (p_i.d, a.w/2)^T \\
&\quad p_i.\rho = \pi/2
\end{aligned}$$

Die neu eingeführten Elemente der Ports von Areas sind in dieser Definition nocheinmal zusammengefasst:

Definition 4.41 (***AreaPort***)
Der Typ ***AreaPort*** ist vom Typ ***Port*** abgeleitet. Ein Objekt $p \triangleright$ ***AreaPort*** repräsentiert einen Port eines Objekts vom Typ ***Area***. Zusätzlich zu den Elementen von ***Port*** enthält p:

1. Eine Zahl $d \in \mathbb{R}$.
2. Einen Punkt $r \in \mathbb{R}^2$, den Referenzpunkt von p.
3. Einen Winkel $\rho \in \mathbb{R}$, den Referenzwinkel von p. •

Die geometrische Instanzierung dreht und verschiebt einen Verkehrsknoten a, um ihn mit einem bestimmten Port an eine Straße anzuschließen (vgl. S. 154). Die dabei vorgenommene Transformation wird in a in Form eines Winkels und eines Vektors für die Translation gespeichert. Zusammen mit den Ausmaßen des Rechtecks des Verkehrsknotens sowie der angesprochenen Partitionierung der Port-Menge führt dies zu folgenden Erweiterungen:

Definition 4.42 (Erweiterung von ***Area***)
Zusätzlich zu den bisher genannten Eigenschaften und Elementen von Objekten $a \triangleright$ ***Area*** gilt:

1. a enthält Zahlen $w \in \mathbb{R}$ bzw. $l \in \mathbb{R}$, die die Breite bzw. die Länge des durch a abgedeckten Rechtecks angeben.
2. a enthält einen Winkel $\varphi \in \mathbb{R}$ und einen Vektor $v \in \mathbb{R}^2$. Im Entwurfskoordinatensystem ist $\varphi = 0$ und $v = 0$.
3. Für $p_i \in Ports$ gilt: $p_i \triangleright \boldsymbol{AreaPort}$.
4. Die Menge $Ports$ wird in vier Mengen partitioniert: $Ports_{top}$, $Ports_{bottom}$, $Ports_{left}$, $Ports_{right}$ •

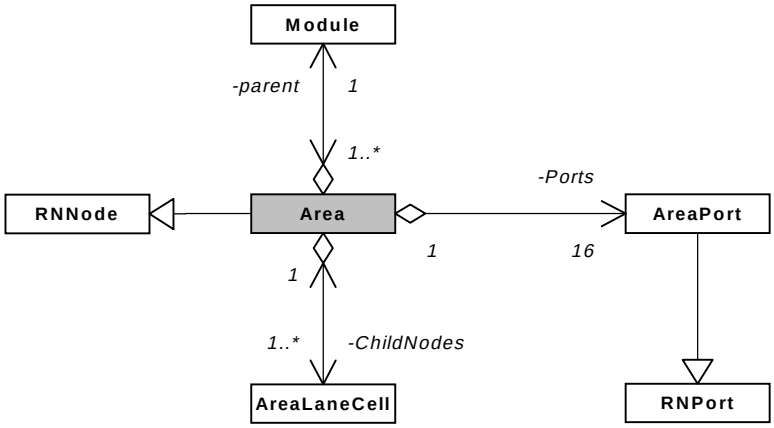

Abbildung 4.27.: Einordnung von ***Area*** in das Modell.

Durch die Einführung des Typs ***AreaPort*** kann Abbildung 4.7 (S. 75) durch Abbildung 4.27 präzisiert werden.

4.5.3. Lageplan

Der Lageplan von Verkehrsknoten wird mittels Grammatik 3.17 (S. 53) beschrieben. Sowohl Super-Spuren als auch die Spuren in den Spurzellen sind vom gemeinsamen Typ ***AreaCurve*** abgeleitet. Dieser Typ enthält eine Parameterkurve p_E, die den **Verlauf der Spur im Entwurfskoordinatensystem** definiert. Über die Parameter $a.\varphi$ und $a.v$, die bei der geometrischen Instanzierung des zugehörigen Verkehrsknotens a ermittelt werden, wird p_E transformiert, was den **Verlauf p der Spur während der Simulation** ergibt:

Definition 4.43 (***AreaCurve***)
Der Typ ***AreaCurve*** ist vom Typ ***Curve*** abgeleitet. Ein Objekt $ac \triangleright$ ***AreaCurve*** repräsentiert eine ebene Parameterkurve im Lageplan eines Verkehrsknotens. Zusätzlich zu den Elementen und Eigenschaften von ***Curve*** gilt:

1. ac enthält einen Winkel $\varphi \in \mathbb{R}$ und einen Vektor $v \in \mathbb{R}^2$. Wenn für $al \triangleright$ ***AreaLane*** gilt $al.curve = ac$, dann ist

$$\begin{aligned} ac.\varphi &= al.parent.parent.\varphi \\ ac.v &= al.parent.parent.v \end{aligned}$$

2. ac enthält eine Funktion $p_E : [0, S] \to \mathbb{R}^2$, die jedem Wert von $s \in [0, S]$ die Koordinaten der Kurve an dieser Stelle im Entwurfskoordinatensystem zuweist.

3. ac enthält eine Funktion $\alpha_E : [0, S] \to \mathbb{R}$, die jedem Wert von $s \in [0, S]$ den Winkel zuweist, den die Tangente der Kurve $p_E(s)$ an dieser Stelle mit der positiven x_E-Achse des Entwurfskoordinatensystems einnimmt.

4. ac enthält eine Funktion $\kappa_E : [0, S] \to \mathbb{R}$, die jedem Wert von $s \in [0, S]$ die Krümmung von p_E an dieser Stelle zuweist.

5. Die Rotationsmatrix $R(\varphi)$ sei wie in Gleichung 4.10 (S. 96) definiert. $p(s)$ wird dann so berechnet:

$$p(s) = R(\varphi) \cdot p_E(s) + v$$

6. Für $\alpha(s)$ gilt:

$$\alpha(s) = \alpha_E(s) + \varphi$$

7. Für $\kappa(s)$ gilt:

$$\kappa(s) = \kappa_E(s)$$

   (Die Krümmung einer Parameterkurve ändert sich durch Rotation und Translation nicht (Satz B.4, 215).) •

Die verschiedenen Arten von Parameterkurven, die den Verlauf einer Spur in einem Verkehrsknoten beschreiben, werden vom Grundtyp ***AreaCurve*** abgeleitet. Gemäß Grammatik 3.17 (S. 53) gibt es hierfür zwei verschiedene Möglichkeiten: Zum einen wird der Verlauf von Spuren direkt in den Spurzellen festgelegt. Hierfür gibt es die Typen ***AStraightLine*** und ***ABezierCurve***. Sie werden im Element *curve* von ***AreaLane*** gespeichert. Der Typ ***AreaLane*** ist von ***Lane*** abgeleitet. Der Sachverhalt ist in den folgenden Definitionen zusammengefasst und in Abbildung 4.28 illustriert.

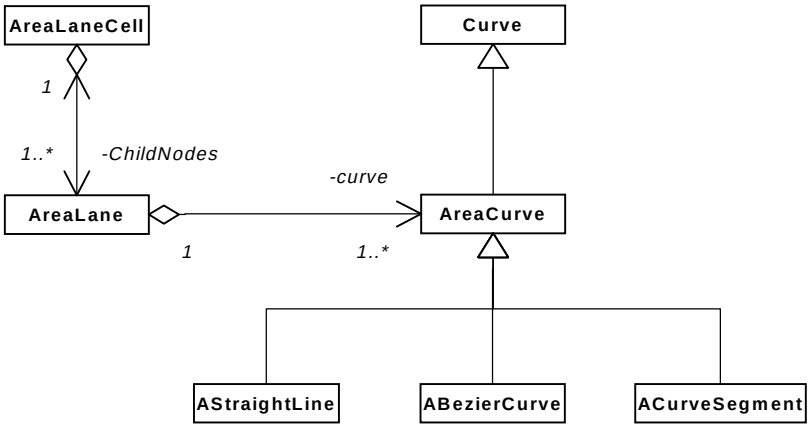

Abbildung 4.28.: Typen von Spurverläufen eines Verkehrsknotens, die in den Spurzellen gespeichert werden.

Definition 4.44 (***AreaLane***)
Ein Objekt $al \triangleright$ ***AreaLane*** ist vom Typ ***Lane*** abgeleitet. Es gilt:

1. $al.parent \triangleright$ ***AreaLaneCell***.
2. Für $al.p_i.Generalization = v.p$ gilt: $v \triangleright AreaLaneCell$.
3. $al.curve$ ist vom Typ ***AreaCurve*** abgeleitet. •

Definition 4.45 (Erweiterung von ***AreaLaneCell***)
Zusätzlich zu den bisher genannten Elementen und Eigenschaften eines Objekts $alc \triangleright$ ***AreaLaneCell*** hat ein Element $al \in alc.ChildNodes$ einen von ***AreaLane*** abgeleiteten Typ. •

Zum anderen gibt es Super-Spuren, deren Verlauf Spurzellen übergreifend im Typ ***Area*** gespeichert wird:

Definition 4.46 (Erweiterung von ***Area***)
Zusätzlich zu den bisher genannten Eigenschaften und Elementen von Objekten $a \triangleright$ ***Area*** enthält a eine Menge $SuperCurves = \{sc_1, \ldots, sc_{n_{sc}}\}$. Jedes $sc_i \in SuperCurves$ ist vom Typ ***AreaCurve*** abgeleitet. •

In der formalen Beschreibung ist der Typ ***ACircularArc*** momentan das einzige Beispiel für eine solche Super-Spur. Das Modell kann jedoch leicht um weitere Typen von Super-Spuren erweitert werden (vgl. Abbildung 4.29). Eine

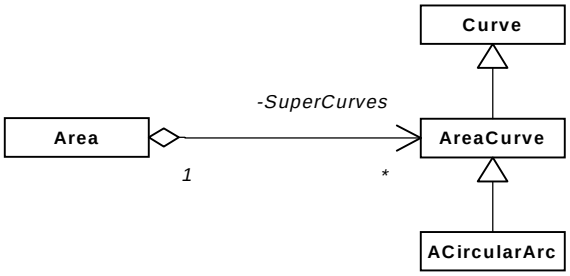

Abbildung 4.29.: Super-Spuren im Verkehrsknoten gespeichert.

Spur vom Typ ***ACurveSegment***, gespeichert in einer Spurzelle, repräsentiert einen Abschnitt einer Super-Spur (Abbildung 4.28).

Im Folgenden werden die erwähnten Typen von Spurverläufen im Detail beschrieben.

Geradenstücke

Ähnlich wie bei Straßen des Typs ***CStraightLine*** repräsentiert ein Objekt sl des Typs ***AStraightLine*** eine geradlinig verlaufende Fahrspur in Verkehrsknoten. Dabei verbindet sl über die Länge S zwei Punkte $p_1, p_2 \in \mathbb{R}^2$. Die Objekte dieses Typs sind in Abbildung 4.30 am Beispiel der T-Kreuzung eingezeichnet. Die formale Beschreibung dieses Typs erfolgt gem. Grammatik 3.18 (S. 54).

Die Gleichung der Parameterkurve $sl.p_E(s)$ im Entwurfskoordinatensystem lautet:

$$sl.p_E(s) = (1 - \frac{s}{S})p_1 + \frac{s}{S}p_2 \tag{4.32}$$

Dabei handelt es sich um eine Parametrisierung über die Bogenlänge.
Schreibt man die Punkte p_1 und p_2 als

$$p_1 = \begin{pmatrix} px_1 \\ py_1 \end{pmatrix}, p_2 = \begin{pmatrix} px_2 \\ py_2 \end{pmatrix}$$

dann ergibt sich für den Tangentenwinkel:

$$sl.\alpha_E(s) = \begin{cases} \arctan \frac{py_2 - py_1}{px_2 - px_1} & : \quad px_2 \neq px_1 \\ \pi/2 & : \quad px_2 = px_1, py_1 < py_2 \\ -\pi/2 & : \quad px_2 = px_1, py_1 > py_2 \end{cases} \tag{4.33}$$

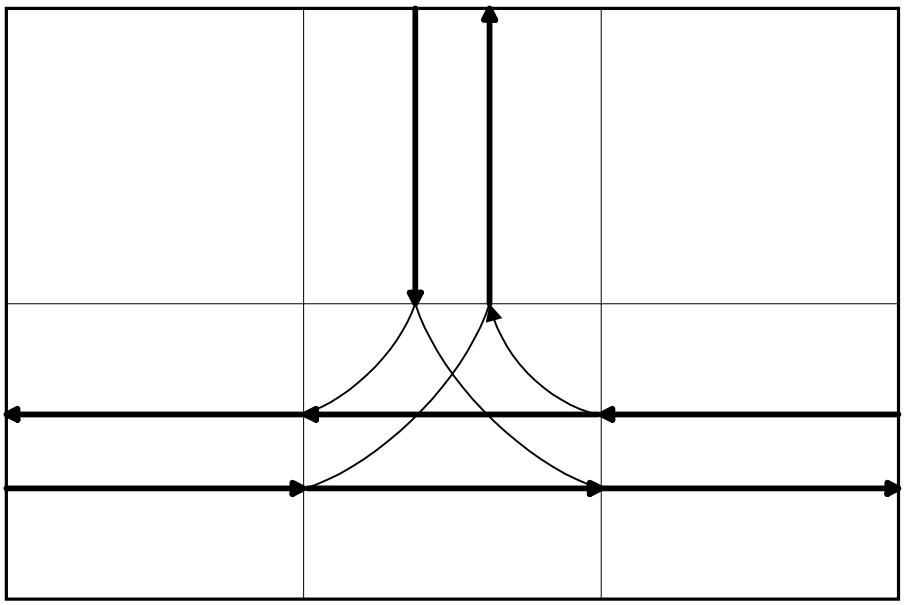

Abbildung 4.30.: Die Spuren, deren Verlauf durch Objekte des Typs ***AStraightLine*** modelliert werden, sind in dieser T-Kreuzung fett eingezeichnet.

Definition 4.47 (***AStraightLine***)
Der Typ ***AStraightLine*** ist vom Typ ***AreaCurve*** abgeleitet. Zusätzlich zu den Elementen und Eigenschaften von ***AreaCurve*** gilt für ein Objekt $sl \triangleright$ ***AStraightLine***:

1. sl enthält zwei Punkte $p_1, p_2 \in \mathbb{R}^2$.
2. $p_E(s)$ ist wie in Gleichung 4.32 definiert.
3. $\alpha_E(s)$ ist wie in Gleichung 4.33 definiert.
4. $\kappa_E(s) \equiv 0$ •

Bezier-Kurven

Der Typ ***ABezierCurve*** dient dazu, zwei Objekte vom Typ ***AStraightLine*** oder ***ACurveSegment*** glatt zu verbinden. Abbildung 4.31 zeigt die Fälle, die dadurch abgedeckt werden.
Die formale Beschreibung ist durch Grammatik 3.18 (S. 54) gegeben.

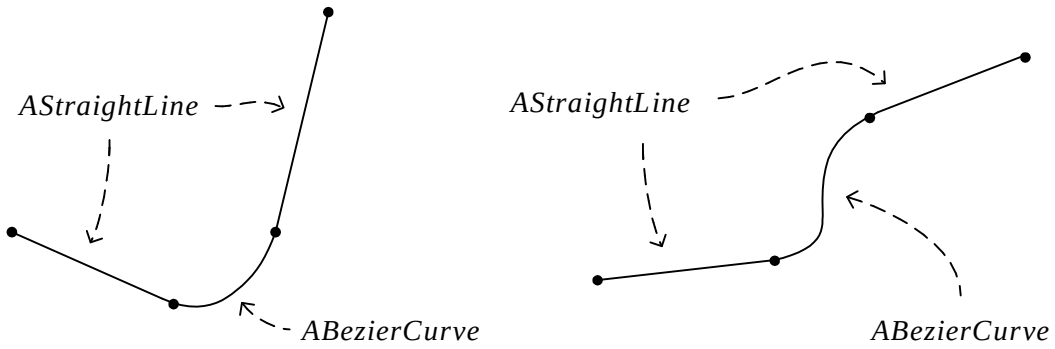

Abbildung 4.31.: Durch Objekte des Typs ***ABezierCurve*** können zwei Geradenstücke sowohl über eine einfach Kurve (links) als auch S-förmig (rechts) miteinander verbunden werden.

Im Gegensatz zu Kurven von Straßen, die aus einer Folge von Objekten der Typen ***CClothoid1***, ***CCircularArc***, ***CClothoid2*** aufgebaut sind, stellt der Verlauf der Krümmung von ***ABezierCurve*** jedoch eine differenzierbare Funktion dar. Da die RASL bei Straßen nur linear verlaufende Krümmungen vorschreibt, kommt der Typ ***ABezierCurve*** nur in Verkehrsknoten zum Einsatz. Hier bietet er wesentlich mehr Flexibilität bei der Modellierung des Lageplans.
Abbildung 4.32 zeigt die Verwendung des Typs am Beispiel der T-Kreuzung.

Über ein Objekt $abc \triangleright$ ***ABezierCurve*** sollen zwei Punkte $p_1 = (px_1, py_1)$ und $p_2 = (px_2, py_2)$ miteinander verbunden werden. Um eine glatte Verbindung zu gewährleisten, werden die Tangentenwinkel α_1 und α_2 in den beiden Punkten vorgeschrieben. Man erhält ein Interpolationsproblem, das überlicherweise von sog. Hermite-Splines gelöst wird. Da Splines im allgemeinen die Eigenschaft haben, die Gesamtkrümmung zu minimieren, sind die aus Hermite-Splines resultierenden Kurven zu eng. Aus diesem Grund wird das Interpolationsproblem durch Bezier-Kurven gelöst. Durch die Wahl der Kontrollpunkte dieser Kurven lässt sich der Krümmungsverlauf beeinflussen.

Mit den Kontrollpunkten $b_i = (bx_i, by_i)$, $i = 0, \ldots, 3$, und $t \in [0, 1]$ besitzt eine Bezier-Kurve die Darstellung (vgl. [14])

$$p_E(t) = \sum_{i=0}^{3} B(t, i, 3) \cdot b_i \tag{4.34}$$

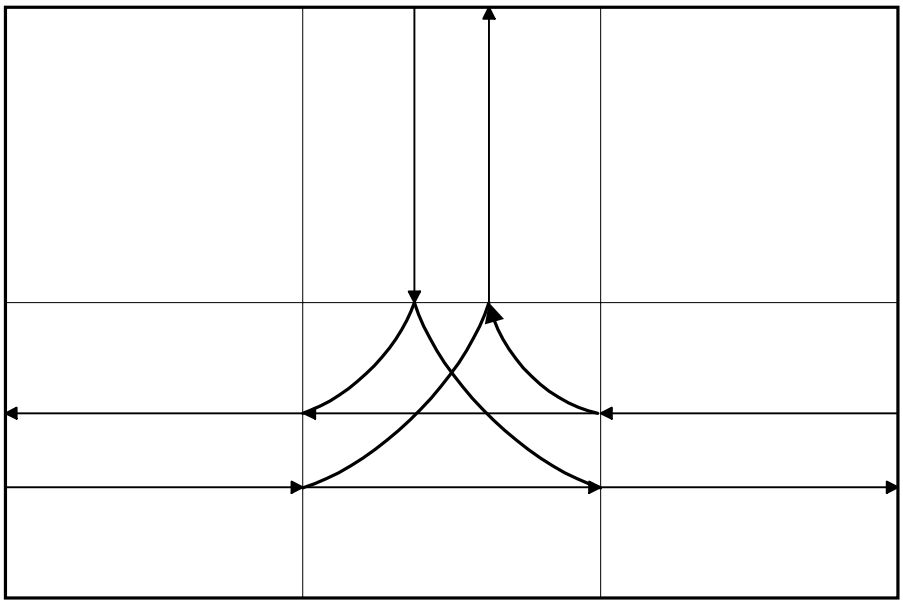

Abbildung 4.32.: Die Spuren, deren Verlauf durch Objekte des Typs ***ABezierCurve*** modelliert werden, sind in dieser T-Kreuzung fett eingezeichnet.

Bei $B(t,i,n)$ handelt es sich um die sog. Bernstein-Polynome:

$$B(t,i,n) = \binom{n}{i} t^i (1-t)^{n-i}$$

Für die Wahl der Kontrollpunkte können folgende Eigenschaften von Bezier-Kurven ausgenützt werden (vgl. [14]):

- $p_E(0) = b_0$ und $p_E(1) = b_3$
- Der Tangentenwinkel im Punkt $p_E(0)$ entspricht dem Tangentenwinkel der durch b_0 und b_1 verlaufenden Geraden.
- Der Tangentenwinkel im Punkt $p_E(1)$ entspricht dem Tangentenwinkel der durch b_2 und b_3 verlaufenden Geraden.

Das vorliegende Interpolationsproblem wird also durch die Kontrollpunkte

$$\begin{aligned} b_0 &= p_1 \\ b_1 &= p_1 + \begin{pmatrix} \frac{|px_2 - px_1|}{c} \cdot \cos\alpha_1 \\ \frac{|py_2 - py_1|}{c} \cdot \sin\alpha_1 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} b_2 &= p_2 - \begin{pmatrix} \frac{|px_2 - px_1|}{c} \cdot \cos\alpha_2 \\ \frac{|py_2 - py_1|}{c} \cdot \sin\alpha_2 \end{pmatrix} \\ b_3 &= p_2 \end{aligned}$$

gelöst. Durch den Parameter $c > 0$ kann der Krümmungsverlauf beeinflusst werden. In den meisten Verkehrsknoten hat sich die Wahl von $c = 2$ als günstig erwiesen, da die so erzeugten Krümmungsverläufe die Route eines Fahrers durch einen Verkehrsknoten gut annähern.

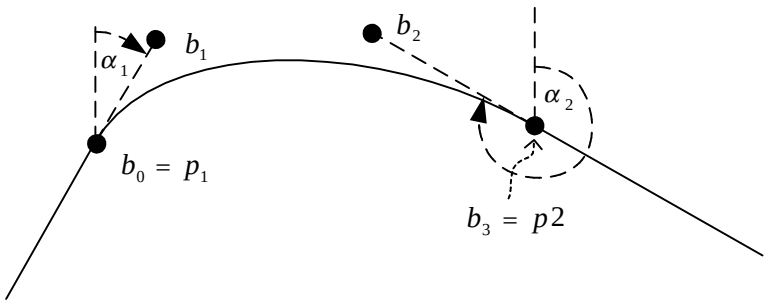

Abbildung 4.33.: Beispiel für die Wahl der Bezier-Punkte $b_0, \ldots, b_2$ bei gegebenen Punkten p_1, p_2 und Winkeln α_1, α_2.

Die Lage der Kontrollpunkte mit diesem Wert von c ist in Abbildung 4.33 zu sehen.

Für den Tangentenwinkel der Bezier-Kurve ergibt sich:

$$\alpha_E(t) = \begin{cases} \arctan \frac{py'_E(t)}{px'_E(t)} & : \quad px'_E(t) \neq 0 \\ \pi/2 & : \quad px'_E(t) = 0, py'_E(t) > 0 \\ -\pi/2 & : \quad px'_E(t) = 0, py'_E(t) < 0 \end{cases} \tag{4.35}$$

Bei gegebenen Kontrollpunkten ist dabei nur die erste Ableitung der Bernstein-Polynome zu berechnen.

Die Berechnung der Krümmung erfolgt nach der Formel aus Definition B.5 (S. 213):

$$\kappa(t) = \frac{px'_E(t) \cdot py''_E(t) - py'_E(t) \cdot px''_E(t)}{(px'^2_E(t) + py'^2_E(t))^{3/2}} \tag{4.36}$$

Hier muss im Wesentlichen die erste und zweite Ableitung der Bernstein-Polynome ausgewertet werden.

Es gibt i.a. keine Bogenlängen-Parametrisierung für Bezier-Kurven. In [15] wird jedoch ein effizientes Verfahren beschrieben, mit dem sich eine solche Parametrisierung gut annähern lässt. Zur Vereinfachung geht daher die folgende Definition von einer Parametrisierung über die Bogenlänge aus, d.h. Bezier-Kurve, Tangentenwinkel und Krümmung sind in der Form $p_E(s)$, $\alpha_E(s)$ und $\kappa(s)$ mit $s \in [0, S]$ gegeben.

Dies führt zu folgender Definition des Typs ***ABezierCurve***:

Definition 4.48 (***ABezierCurve***)
Der Typ ***ABezierCurve*** ist vom Typ ***AreaCurve*** abgeleitet. Zusätzlich zu den Elementen und Eigenschaften von ***AreaCurve*** gilt für ein Objekt $abc \triangleright$ ***ABezierCurve***:

1. abc enthält zwei Punkte $p_1, p_2 \in \mathbb{R}^2$.
2. abc enthält zwei Winkel $\alpha_1, \alpha_2 \in \mathbb{R}$.
3. $p_E(s)$ ist über die Bogenlängen Parametrisierung von Gleichung 4.34 definiert.
4. $\alpha_E(s)$ ist über die Bogenlängen Parametrisierung von Gleichung 4.35 definiert.
5. $\kappa(s)$ ist über die Bogenlängen Parametrisierung von Gleichung 4.36 definiert. •

Kreisbögen als Super-Kurven

Zur Modellierung von Kreisen bzw. Kreisbögen wird folgender Ansatz gewählt:

$$p_E(s) = p_0 + \begin{pmatrix} |r| \cos \frac{s}{r} \\ r \sin \frac{s}{r} \end{pmatrix} \tag{4.37}$$

Der Kreis hat den Mittelpunkt $p_0 \in \mathbb{R}^2$ und den Radius r. Für $r > 0$ wird der Kreis im Uhrzeigersinn durchlaufen, für $r < 0$ gegen den Uhrzeigersinn. Einen Vollkreis erhält man mit $s \in [0, 2|r|\pi]$. Durch analoge Rechnung wie im Beweis von Satz 4.1 (S. 96) sieht man, dass es sich bei 4.37 um eine Bogenlängen Parametrisierung des Kreises handelt. Für den Tangentenwinkel an der Stelle s ergibt sich:

$$\alpha_E(s) = \operatorname{sgn} r \cdot \frac{\pi}{2} + \frac{s}{r} \tag{4.38}$$

Die Krümmung ist

$$\kappa(s) = \frac{1}{r} \tag{4.39}$$

Als Super-Kurve zur Modellierung einer Fahrbahn in einem Kreisverkehr kann also folgender Typ verwendet werden:

Definition 4.49 (***ACircularArc***)
Der Typ ***ACircularArc*** ist vom Typ ***AreaCurve*** abgeleitet. Zusätzlich zu den Elementen und Eigenschaften von ***AreaCurve*** gilt für ein Objekt $aca \triangleright$ ***ACircularArc***:

1. aca enthält einen Punkt $p_0 \in \mathbb{R}^2$, den Mittelpunkt des Kreisbogens im Entwurfskoordinatensystem.
2. aca enthält eine Zahl $r \in \mathbb{R}$, den Radius des Kreisbogens.
3. $p_E(s)$ ist wie in Gleichung 4.37 definiert.
4. $\alpha_E(s)$ ist wie in Gleichung 4.38 definiert.
5. $\kappa(s)$ ist wie in Gleichung 4.39 definiert. •

Eine Parameterkurve, die ein Segment einer übergeordneten Kurve (wie z.B. eine des Typs ***ACircularArc***) repräsentiert, wird so definiert:

Definition 4.50 (***ACurveSegment***)
Der Typ ***ACurveSegment*** ist vom Typ ***AreaCurve*** abgeleitet. Zusätzlich zu den Elementen und Eigenschaften von ***AreaCurve*** gilt für ein Objekt $acs \triangleright$ ***ACurveSegment***:

1. acs enthält ein Element $sc \triangleright$ ***AreaCurve***. Wenn für $al \triangleright$ ***AreaLane*** gilt $al.curve = acs$, dann ist
$$acs.sc \in al.parent.parent.SuperCurves$$
2. acs enthält Zahlen s_1 und s_2 mit $s_1, s_2 \in [0, sc.S]$ und $s_1 < s_2$.
3. Die Länge S des Segments ist durch $S = s_2 - s_1$ gegeben.
4. $p_E(s) = sc.p_E(s + s_1)$
5. $\alpha_E(s) = sc.\alpha_E(s + s_1)$
6. $\kappa(s) = sc.\kappa_E(s + s_1)$ •

4.5.4. Höhenplan

Ein Objekt $a \triangleright \textbf{\textit{Area}}$ wird, bis auf relative Höhen von Spuren, als flach angenommen. Von der geometrischen Instanzierung wird a an den Port $c.p_i$ einer Straße $c \triangleright \textbf{\textit{Course}}$ angeschlossen, der auf einer bestimmten, durch $c.hp$ gegebenen Höhe liegt. Dieses Höhenoffset wird in a gespeichert und an die Höhenprofile der Straßen weitergegeben, die ebenfalls an a anschließen.

Definition 4.51 (Erweiterung von ***Area***)
Zusätzlich zu den bisher genannten Eigenschaften und Elementen von Objekten $a \triangleright \textbf{\textit{Area}}$ enthält a eine Zahl $h \in \mathbb{R}$. h ist die Höhe, auf der a liegt. •

Im Gegensatz zu Straßen $c \triangleright \textbf{\textit{Course}}$, bei denen die relativen Höhen von Spuren über den gesamten Verlauf konstant sind, für deren Spuren l also

$$l.LaneInfo.h_1 = l.LaneInfo.h_2$$

gilt, kann eine Spur eines Verkehrsknotens am Anfang und am Ende unterschiedliche relative Höhen haben. Sind zwei Spuren l_i, l_j eines Verkehrsknotens im Graphen der Ebene LANES über die Ports $l_i.p_1$ und $l_j.p_2$ verbunden, dann muss jedoch gelten:

$$l_i.LaneInfo.h_1 = l_j.LaneInfo.h_2$$

Die Funktion $l.h$ aus Definition 4.21 (S. 85) für Spuren $l \triangleright \textbf{\textit{AreaLane}}$ ist folglich so definiert:

Definition 4.52 (Erweiterung von ***AreaLane***)
Für die Funktion $h(s)$ eines Objekts $l \triangleright \textbf{\textit{AreaLane}}$, mit der die absolute Höhe an einer Stelle s auf der Parameterkurve der Spur gemessen wird, gilt:

$$l.h(s) := a.h + l.h_{rel}(s)$$

mit $a := l.parent.parent$. Dabei beschreibt $l.h_{rel}$ die relative Spurhöhe (vgl. Definition 4.21, S. 85). •

4.5.5. Generierung aus der formalen Beschreibung

Aus jeder Instanz eines Verkehrsknotens, dessen Schablone gemäß Grammatik 3.15 (S. 50) beschrieben wurde, erzeugt Algorithmus 4.1 (S. 83) ein Objekt $a \triangleright \textbf{\textit{Area}}$. Die Informationen der Schablone werden in a gespeichert. Im folgenden werden die hierfür benötigten Verfahren beschrieben, die nicht unmittelbar aus den Definitionen dieses Kapitels hervorgehen.

Parameterkurven der Spuren

Wie Grammatik 3.19 (S. 55) zeigt, gibt der Anwender Koordinaten nur für den Beginn einer Spur an. Wenn dieser an einem Port des Verkehrsknotens liegt, können die Koordinaten anhand von Referenzpunkt, -winkel und Querschnitt des Ports ermittelt werden. Gleiches gilt für den Tangentenwinkel in diesem Punkt.

Aus diesen Informationen müssen bei der Generierung des Modells die Gleichungen für die Parameterkurven aufgestellt werden. Dies geschieht in zwei Schritten: Zunächst werden für alle geraden Spuren die Koordinaten der Endpunkte bestimmt. Zusammen mit den Punkten des Beginns sind damit die Parameterkurven vollständig bestimmt. Folglich stehen auch die Tangentenwinkel am Beginn und am Ende fest. Bekannt sind auch Beginn- und Endpunkte sowie die zugehörigen Tangentenwinkel von Segmenten von Super-Kurven. Aus diesen Daten werden im zweiten Schritt die Kontrollpunkte der Bezier-Kurven berechnet. Das Verfahren ist in Algorithmus B.2 (im Anhang, S. 217) zusammengefasst.

Verknüpfungen der Spurzellen

Über Grammatik 3.20 (S. 57) werden die Spuren eines Verkehrsknotens verknüpft, nicht jedoch die Spurzellen, in denen sie sich befinden. Der Teilgraph der Ebene LaneCells, der als Knoten die Spurzellen eines bestimmten Verkehrsknotens $a \triangleright Area$ enthält, muss also aus diesen Informationen aufgebaut werden. Dies erfolgt nach Verfahren 4.3. Im Algorithmus werden mit l, l' beliebige Spuren von a bezeichnet, unabhängig davon, in welchen Spurzellen sie sich befinden.
Zur Verbindung eines Knotens des Typs ***CourseLaneCell*** mit einer Spurzelle $lc \in a.ChildNodes$, die an einem Port von a liegt, wird ebenfalls ein neuer Port zu $lc.Ports$ hinzugefügt.

Algorithmus 4.3 Verknüpfung der Spurzellen eines Verkehrsknotens $a \triangleright$ ***Area***

for all $lc \in a.ChildNodes$ **do**
  $lc.Ports := \emptyset$
end for
for all Spuren l, l' von a, die über $l.p_i$ und $l'p_j$ verbunden sind **do**
  if $l.parent$ noch nicht mit $l'.parent$ verbunden **then**
    Füge neuen Port p zu $l.parent.Ports$ hinzu.
    Füge neuen Port p' zu $l'.parent.Ports$ hinzu.
    Verbinde $l.parent$ und $l'.parent$ über p und p'.
  end if
end for

5. Simulation

5.1. Geometrische Instanzierung

5.1.1. Überblick

Auf jeder der Ebenen MODULES, RN, LANECELLS und LANES bildet das Modell des Straßennetzwerks einen Graphen. Es handelt sich um eine topologische Repräsentation, da der Anwender durch topologische Verkehrsknoten oder bedingte Verknüpfungen Mehrdeutigkeiten bei der Abfolge von Stecken einführt[1] Aufgabe der geometrischen Instanzierung ist es, in jedem Schritt der Simulation auf jeder Ebene einen Teilgraphen zu ermitteln, der den Sichtbarkeitsbereich des Fahrers vollständig enthält und zusätzlich geometrisch konsistent ist, d.h.:

1. Jeder Teilgraph ist zusammenhängend.
2. Die Verknüpfungsinformationen aller Ports eines Knotens eines jeden Teilgraphen sind eindeutig.
3. Die durch die Knoten der Teilgraphen repräsentierten Streckenabschnitte schließen bezüglich ihres Lage- und Höhenplans glatt aneinander, sie bilden also ein durchgängiges Streckennetz.

Sobald ein Port p eines Knotens des Typs ***Course*** oder ***Area*** in den Sichtbarkeitsbereich des Fahrers kommt, wird unter den Alternativen in $p.Edges$ eine eindeutige Kante ausgewählt. Über die Generalisierung $p.Generalization$ des Ports kann dann die Verknüpfung auf der darüberliegenden Ebene der Module eindeutig festgelegt werden. Mit den Verfeinerungen $p.Refinements$ können die Verbindungen auf den darunterliegenden Ebenen der Spurzellen und Spuren bestimmt werden. Die folgenden Beispiele veranschaulichen die Vorgehensweise.

[1] Abschnitt 3.3.2 zeigt, wie solche Streckennetze vom Anwender definiert werden.

Ein möglicher Ablauf der Simulation bei der **Navigationsaufgabe**[2] ist in Abbildung 5.1 gezeigt.
Die Kreise stellen den Sichtbarkeitsbereich des Fahrers dar. Die dunkelgrauen Strecken sind Knoten des Teilgraphen auf der Ebene RN, die hellgrauen sind im Lauf der Simulation aus dem Sichtbarkeitsbereich des Fahrers herausgefallen und liegen daher nicht im Teilgraphen. Das Visualierungsmodul der Fahrsimulation stellt in jedem Schritt nur die Knoten dar, die in den Teilgraphen liegen. Der Fahrer nähert sich der T-Kreuzung auf der Straße s_1 (linkes Bild). Erst wenn der Port $s_1.p_2$ in den Sichtbarkeitsbereich gelangt, wird die T-Kreuzung k so transformiert, dass sie glatt an s_1 anschließt. Der Fahrer fährt in k fälschlicherweise geradeaus (mittleres Bild). Er gelangt auf die Strecke s_3, an die über eine topologische Schleife $s_1.p_1$ angeschlossen ist. Die Straße s_1 ist zu diesem Zeitpunkt bereits wieder aus dem Sichtbarkeitsbereich des Fahrers herausgefallen. Sobald $s_3.p_2$ in den Sichtbarkeitsbereich das Fahrers gerät, wird s_1 so transformiert, dass sie glatt an $s_3.p_2$ anschließt. Auch die T-Kreuzung k, zu der der Fahrer jetzt ein zweites mal über s_1 kommt, wird entsprechend transformiert, sobald $s_1.p_2$ sichtbar wird (rechtes Bild).

Abbildung 5.2 zeigt einen möglichen Ablauf für die Fahraufgabe **Scharfe Kurve**[3].
Wieder deuten die Kreise den Sichtbarkeitsbereich des Fahrers an. Der geometrisch konsistente Teilgraph auf der Ebene RN wird von den dunkelgrauen Strecken gebildet. Im oberen Bild nähert sich der Fahrer auf den Straßen s_1 und s_2 einem vorausfahrenden Fahrzeug (weiß). Sobald der Port $s_2.p_2$ in den Sichtbarkeitsbereich kommt (mittleres Bild), wird geprüft, ob das Vorderfahrzeug bereits genügend nahe ist. Die Bedingung, die der Anwender hierfür definiert hat, lautet

```
traffic.DistSameAheadNext <= 50
```

Dabei misst die Variable `DistSameAheadNext`, die vom Softwaremodul `traffic` zur Verfügung gestellt wird, den Abstand zum nächsten Fahrzeug auf der eigenen Spur. Angenommen, diese Bedingung ist im mittleren Bild noch nicht erfüllt. Dann wird als eindeutige Verknüpfung die zwischen $s_2.p_2$ und $s_1.p_1$

[2] Die Navigationsaufgabe ist auf Seite 19 erklärt. Das Streckennetz wird auf Seite 38 beschrieben.

[3] Die Fahraufgabe Scharfe Kurve wird auf Seite 19 erklärt. Das Streckennetz wird auf Seite 41 definiert.

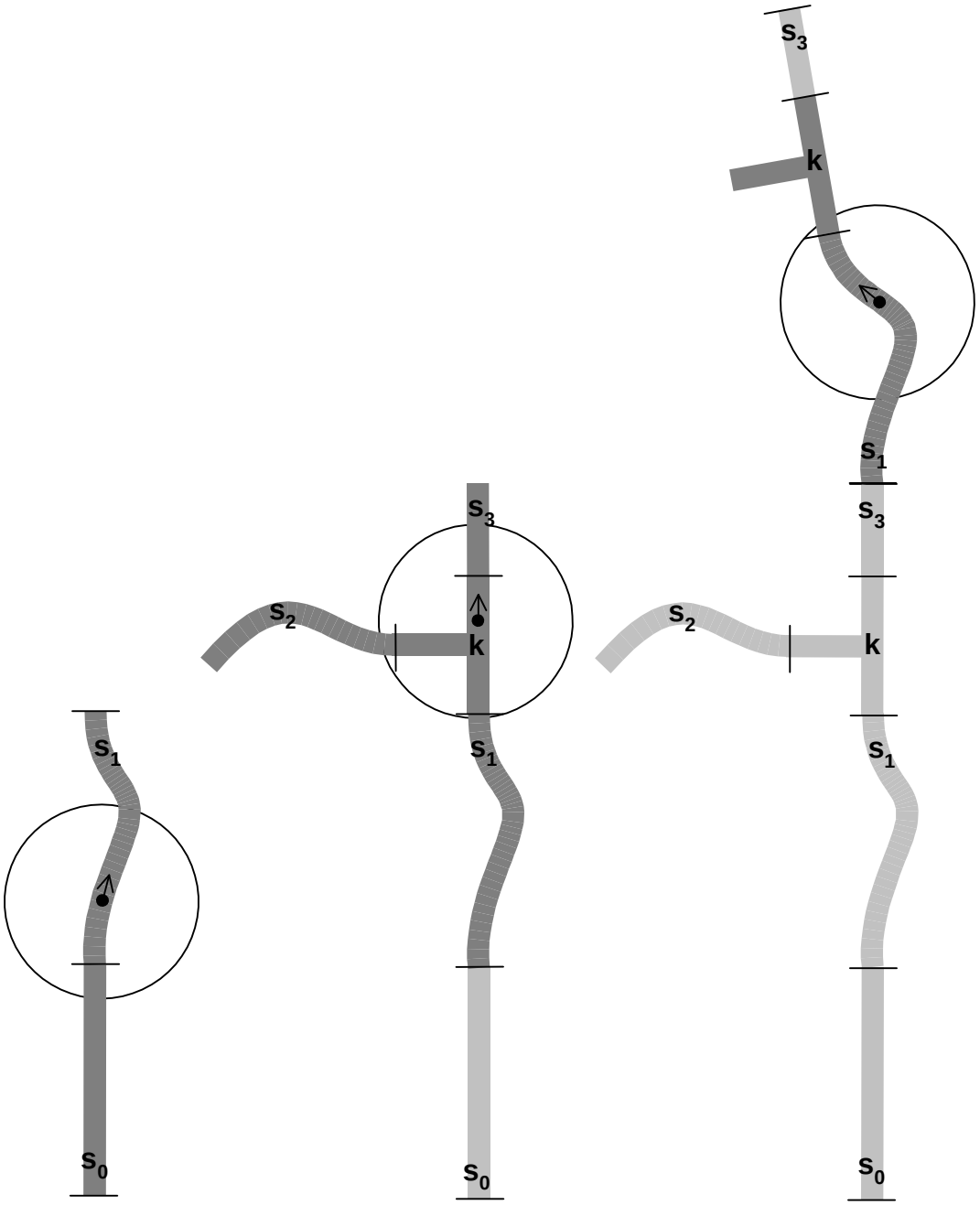

Abbildung 5.1.: Möglicher Ablauf der Navigationsaufgabe (von links nach rechts). Die Kreise stellen den Sichtbarkeitsbereich des Fahrers dar. Die dunkelgrauen Knoten bilden auf der Ebene RN einen geometrisch konsistenten Teilgraphen.

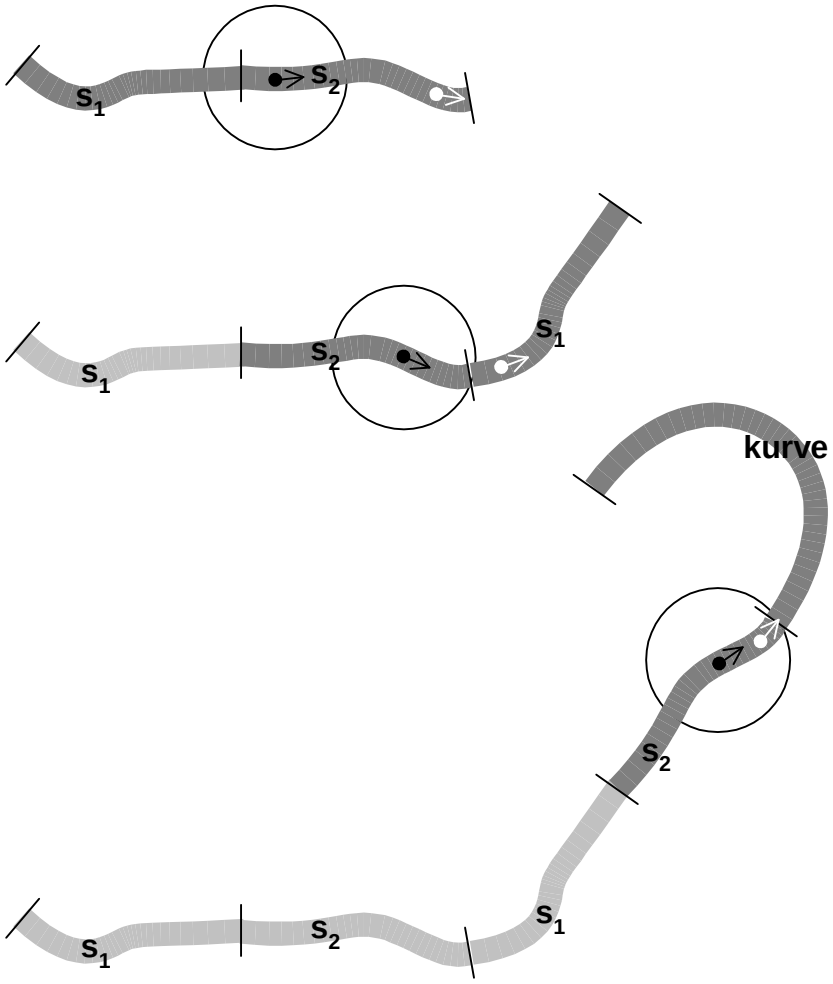

Abbildung 5.2.: Möglicher Ablauf der Situation "Scharfe Kurve" (von oben nach unten). Die Kreise stellen den Sichtbarkeitsbereich des Fahrers dar. Die dunkelgrauen Knoten bilden auf der Ebene RN einen geometrisch konsistenten Teilgraphen.

gewählt. Die Strecke s_1 wird ans Ende von s_2 transformiert und der Fahrer nähert sich weiter dem vorausfahrenden Fahrzeug an. Im unteren Bild kommt $s_2.p_2$ zum zweiten mal in den Sichtbarkeitsbereich. Jetzt ist die Bedingung erfüllt, d.h. der Fahrer ist weniger als 50 Meter vom Vordermann entfernt. Deswegen wird die Verknüpfung zwischen $s_2.p_2$ und $kurve.p_1$ selektiert. Der Fahrer folgt dem Führungsfahrzeug mit zu hoher Geschwindigkeit in die scharfe Kurve.

5.1.2. Grundlegender Ablauf

Da die Knoten einer jeden Ebene vom Grundtyp ***Node*** abgeleitet sind, kann der Algorithmus zur Bestimmung des geometrisch konsistenten Teilgraphen als Methode von ***Node*** implementiert werden. Diese Methode wird $v.Update$ genannt. Ausgehend vom Knoten, auf dem sich der Fahrer gerade befindet, ruft $v.Update$ in jedem Simulationsschritt die Methoden der angeschlossenen Knoten auf. Dadurch wird der Graph einer Ebene rekursiv (in depth first order) traversiert. Die Rekursion endet, wenn ein Knoten erreicht wird, der sich nicht im Sichtbarkeitsbereich befindet. Dies wird für jede Modell-Ebene durchgeführt, beginnend mit der Ebene RN, da der Anwender das Streckennetz auf dieser Ebene aufgebaut und Regeln für die Auflösung nicht eindeutiger Verknüpfungen angegeben hat (vgl. Abschnitt 3.3.2, S. 36).

Ein typischer Ablauf einer Simulation ist in Algorithmus 5.1 skizziert. Die genaue Integration dieses Ablaufs in die Simulation wird im Abschnitt 5.2.3 beschrieben.

Algorithmus 5.1 Ablauf der geometrischen Instanzierung

```
// 1. Schritt der Simulation: Aufsetzen des Fahrers
// Fahrer wird in Modul m auf Course c, LaneCell lc ∈ c.ChildNodes aufgesetzt
c.Update(...)
m.Update(...)
lc.Update(...)
for all l ∈ lc.ChildNodes do
  l.Update(...)
end for
// geometrische Instanzierung während der Simulation
while Simulation läuft do
  ermittle sichtbare LaneCell lc , in der sich Fahrer befindet
  lc.parent.Update(...)
  lc.parent.parent.Update(...)
  lc.Update(...)
  for all l ∈ lc.ChildNodes do
    l.Update(...)
  end for
end while
```

Gerät ein Knoten in einem Simulationsschritt in den Sichtbarkeitsbereich, so sind, je nach Ebene des Graphen, in dem sich der Knoten befindet, unterschiedliche Aufgaben zu erledigen. Handelt es sich beispielsweise um einen Knoten $lc \triangleright$ ***CourseLaneCell***, dann muss die Referenzkurve von lc so transformiert werden, dass sie glatt an die Spurzelle anschließt, von der aus $lc.Update$ aufgerufen wurde. Ein Knoten des Typs ***Area***, der neu in den Sichtbarkeitsbereich kommt, wird als Ganzes transformiert und so glatt mit der aufrufenden Straße verbunden.

Um solche Aufgaben für die speziellen Knotentypen ausführen zu können, wird der Grundklasse ***Node*** eine rein virtuelle Methode $DoGeoInst$ hinzugefügt. Diese Methode wird von den abgeleiteten Klassen überschrieben und jedes Mal aufgerufen, wenn ein Knoten neu in den Sichtbarkeitsbereich kommt. Mit der Implementation von $DoGeoInst$ für die verschiedenen Knotentypen befassen sich die Abschnitte 5.1.5 bis 5.1.8 dieses Kapitels.

Um feststellen zu können, ob ein Knoten im Vergleich zum letzten Simulationsschritt neu in den Sichtbarkeitsbereich kommt oder ob er bereits sichtbar war, bekommt er einen Zeitstempel:

Definition 5.1 (Erweiterung von ***Node***)
Zusätzlich zu den bisher genannten Elementen von Objekten $v \triangleright \textbf{\textit{Node}}$ enthält v eine Zahl $T_{update} \in \mathbb{N}_0$. Beim Start der Simulation gilt stets $T_{update} = 0$. •

Jedesmal, wenn ein Knoten bei der Traversierung seines Graphen berührt wird, wenn also seine Methode ***Update*** aufgerufen wird, und er sichtbar ist, wird T_{update} auf die Nummer T des aktuellen Simulationsschritts gesetzt.

Algorithmus 5.2 zeigt die Methode $\textbf{\textit{Node}} :: Update$.
Sie bekommt folgende Argumente übergeben:

- T: Nummer des aktuellen Simulationsschritts
- $S_{p,max}, s^{\star}_{p}, d_p, dir_p$: Werden im nächsten Abschnitt (5.1.3) erläutert.
- p_{from}: Port, der an den aufrufenden Knoten v_{caller} anschließt.
- v_{caller}: Knoten, von dem aus die Methode aufgerufen wurde.
- p_{caller}: Port von v_{caller}, mit dem p_{from} verbunden ist.

Kommentare zu Algorithmus 5.2:

Zeile 2 - 4: Wenn in einem Schritt der Simulation das Update eines Knotens v durchgeführt wurde, dann wird in Zeile 21 der Zeitstempel $v.T_{update}$ auf den aktuellen Simulationsschritt T gesetzt. Trifft also die Bedingung $T = v.T_{update}$ in Zeile 2 zu, dann kann die Rekursion abgebrochen werden, da v in diesem Simulationsschritt schon behandelt wurde.

Zeile 6 - 12: Wenn v nicht im Sichtbarkeitsbereich liegt, muss unterschieden werden, ob v im letzten Schritt noch im Sichtbarkeitsbereich war, d.h. ob v den Zeitstempel $T_{update} = T - 1$ trägt. In diesem Fall ist sicher, dass v aus dem Teilgraphen herausfällt. Andernfalls (Zeile 11) handelt es sich bei v um einen Kandidaten für eine zukünftige geometrische Instanzierung. Beide Informationen sind für einige Softwaremodule der Fahrsimulation von Interesse. Hierauf wird in Abschnitt 5.2.2 näher eingegangen.

Algorithmus 5.2 ***Node*** :: $Update(T, S_{p,max}, s_p^\star, d_p, dir_p, p_{from}, v_{caller}, p_{caller})$

```
1:  // Wurde Knoten in diesem Update-Schritt schon behandelt?
2:  if T = T_update then
3:    return
4:  end if
5:
6:  if Knoten nicht im Sichtbarkeitsbereich then
7:    // War Knoten beim letzen mal noch im Sichtbarkeitsbereich?
8:    if T_update = T − 1 then
9:      Knoten fällt aus dem Teilgraphen heraus.
10:   else
11:     Knoten könnte bald sichtbar werden.
12:   end if
13: else
14:   // Wurde Knoten beim letzten mal noch nicht behandelt?
15:   if T_update < T − 1 then
16:     Geometrische Instanzierung durchführen.
17:   end if
18:
19:   Knoten liegt jetzt im Teilgraphen.
20:
21:   T_update := T
22:
23:   // Update der angeschlossenen Knoten
24:   for all p ∈ Ports do
25:     if Port ist sichtbar then
26:       Bestimme eindeutige Verknüpfungen des Ports.
27:       Rufe Update der an p angeschlossenen Knoten auf.
28:     else
29:       // Port ist nicht in Sichtbarkeit
30:       Alle an p angeschlossenen Kanten könnten demnächst geometrisch
          instanziert werden.
31:     end if
32:   end for
33: end if
```

Zeile 14-17: Der Knoten v befindet sich im Sichtbarkeitsbereich des Fahrers. Wenn sein Zeitstempel T_{update} älter ist als $T - 1$, dann ist v neu in den Sichtbarkeitsbereich eingetreten. In Zeile 16 muss dann die geometrische Instanzierung von v ausgeführt werden. Dies geschieht durch den Aufruf von $v.DoGeoInst$.

Zeile 19: v liegt in diesem Simulatiosschritt im Sichtbarkeitsbereich des Fahrers. Auch diese Information wird, wie in Abschnitt 5.2.2 beschrieben, anderen Softwaremodulen der Simulation zur Verfügung gestellt.

Zeile 21: Der Zeitstempel von v wird mit dem aktuellen Zeitschritt T der Simulation abgeglichen.

Zeile 23 - 33 (Überblick): Dieser Block befindet sich im **else**-Zweig der Abfrage in Zeile 6, was bedeutet, dass sich v in der Sicht des Fahrers befindet. Folglich wurde in diesem oder in einem vorausgehenden Simulationsschritt bereits die Methode $DoGeoInst(\ldots)$ aufgerufen (nämlich als v neu in den Sichtbarkeitsbereich eintrat).

Zeile 25 - 27: Für jeden Port eines Knotens v wird ermittelt, ob er sich im Sichtbarkeitsbereich des Fahrers befindet. Ist dies der Fall, werden die Kanten des Ports eindeutig festgelegt (Zeile 26) und $Update$ der durch diese Kanten mit v verbundenen Knoten durchgeführt (Zeile 27).

Zeile 28 - 30: Ist der Port nicht sichtbar, dann wird trotzdem $Update$ *aller* über diesen Port verbundenen Knoten aufgerufen, da diese Knoten Kandidaten für eine baldige geometrische Instanzierung sind. Die Methode $Update$ solcher Knoten endet in Zeile 11.

Die folgenden Abschnitte präzisieren Algorithmus 5.2.

5.1.3. Sichtbarkeit

Für die Sichtbarkeit von Knoten v gilt:

1. Der Fahrer befindet sich in v oder mindestens einer der Ports von v liegt im Sichtbarkeitsbereich.
2. Für einen Knoten lc der Ebene LaneCells gilt: lc ist sichtbar $\Rightarrow$ alle Spuren in $lc.ChildNodes$ sind sichtbar. Der Grund dafür ist, dass die Ports von Spuren immer mit den Ports der Spurzelle zusammenfallen, zu der sie gehören.

3. Ein Knoten des Typs ***Area*** ist stets als Ganzes sichtbar, d.h. ist ein Knoten des Typs ***Area*** sichtbar, dann sind auch alle seine Spurzellen sichtbar und damit, wie im vorhergehenden Punkt beschrieben, auch alle seine Spuren. Diese Annahme vereinfacht die geometrische Instanzierung und ist aufgrund der geringen Ausmaße von Verkehrsknoten im Vergleich zu Straßen zu vertreten.

Die Sichtbarkeit von Ports wird in Zeile 25 von Algorithmus 5.2 geprüft. Ein naheliegendes Kriterium hierfür wäre eine obere Schranke für den euklidischen Abstand des Ports zum Fahrer[4]. Eine solche Überprüfung hat jedoch einige Nachteile:

1. Um für die geometrische Konsistenz der Teilgraphen eindeutige Verknüpfungen festzulegen, wird die Methode ***Node*** :: *Update* zuerst auf der Ebene RN ausgeführt. So wird bei jedem Aufruf geprüft, ob die Ports eines Knotens $c \triangleright$ ***Course*** sichtbar sind. Um den euklidischen Abstand des Ports zum Fahrer berechnen zu können, werden die Koordinaten des Ports benötigt. Diese stehen aber erst fest, wenn *alle* Referenzkurven der Spurzellen von c glatt aneinander gehängt wurden, also nach der geometrischen Instanzierung auf der Ebene LaneCells.

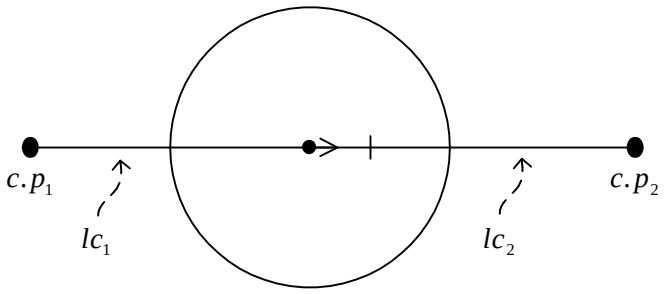

Abbildung 5.3.: Bei einem Sichtbarkeitstest über den euklidischen Abstand müsste lc_2 der Straße c bereits geometrisch instanziert sein, um die Sichtbarkeit von $c.p_2$ entscheiden zu können.

Das Problem ist in Abbildung 5.3 veranschaulicht. Die Straße c besteht aus zwei Spurzellen, lc_1 und lc_2. Eingezeichnet sind deren Referenzkurven, die jeweils eine Gerade repräsentieren. Der euklidische Sichtbarkeitsbereich des Fahrers ist als Kreis um seine Position eingezeichnet. Ohne

[4] In den Abbildungen 5.1 und 5.2 wurde der Sichtbarkeitsbereich zur Veranschaulichung nach dieser Methode eingezeichnet.

dass die Referenzkurve von lc_2 geometrisch instanziert wird, sind die Koordinaten von $c.p_2$ nicht bekannt.

2. Selbst wenn das Problem aus dem vorhergehenden Punkt beseitigt wäre, kann es vorkommen, dass ein Port $c.p_i$ eines Knotens $c \triangleright \textbf{\textit{Course}}$ sichtbar ist, obwohl einige Ports seiner Spurzellen nicht sichtbar sind. Dies ist in Abbildung 5.4 dargestellt.

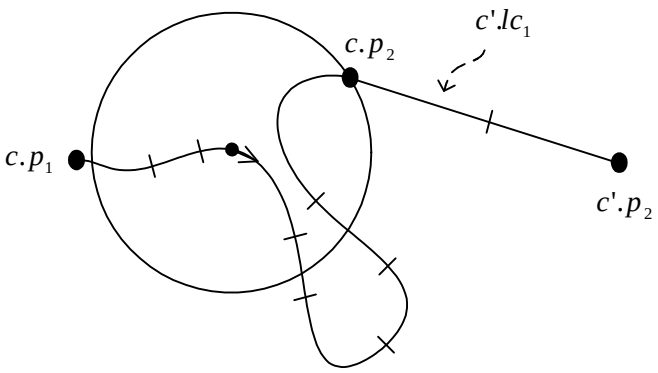

Abbildung 5.4.: Bei einem Sichtbarkeitstest über den euklidischen Abstand würde c' geometrisch instanziert werden, die Spurzelle $c'.lc_1$ jedoch nicht, weil ein Teil der Spurzellen von c nicht sichtbar ist.

   In diesem Beispiel würde die Methode $\textbf{\textit{Node}} :: \mathit{Update}$ die an $c.p_i$ anschließende Strecke c' instanzieren, die Spurzelle $c'.lc_1$ jedoch nicht.

3. Wie in Abschnitt 3.3.2 (S. 36) erläutert, muss der Anwender den Sichtbarkeitsbereich leicht einschätzen können, um gültige topologische Verkehrknoten und Schleifen zu definieren. Der Beschreibung des Lageplans ist der euklidische Sichtbarkeitsbereich jedoch nur schwer zu entnehmen.

Eine alternative Art, den Sichtbarkeitstest durchzuführen, beruht auf dem sog. modifizierten Sichtbarkeitsbereich. Bei Straßen werden dabei die Abstände zum Fahrer *entlang von Spuren* gemessen. Der modifizierte Sichtbarkeitsbereich ist durch eine Schranke $S_{p,max}$ für diese Abstände begrenzt. Aufgrund ihrer im Vergleich zu Straßen geringen Ausmaße werden Verkehrsknoten nicht miteingerechnet, sie tragen also nichts zum Abstand zum Fahrer bei. Dadurch werden die genannten Problem gelöst:

1. Bei einem Knoten $c \triangleright \textbf{\textit{Course}}$ steht die Länge fest, unabhängig davon, wie der Lageplan verläuft. Für jeden Port kann die Sichtbarkeit daher unabhängig von den Spurzellen festgestellt werden.

2. Da sich die Länge der Referenzkurven einer Straße zur Gesamtlänge aufsummiert, folgt aus der Tatsache, dass ein Port eines $c \triangleright \textbf{\textit{Course}}$ sichtbar ist, auch die Sichtbarkeit aller Spurzellen zwischen der Fahrerposition und diesem Port.

3. Der Anwender legt bei der Beschreibung des Lageplans einer Straße die Gesamtlänge selbst fest. Verkehrsknoten besitzen im Sinne des modifizierten Sichtbarkeitsbereich keine Ausdehnung. So kann der modifizierte Sichtbarkeitsbereich leicht eingeschätzt werden.

Die Abstände entlang von Spuren können effizient über den sog. propagierten Streckenmeter berechnet werden. Er misst die Länge des Weges zum Aufsetzpunkt des Fahrers beim Start der Simulation. Der Aufsetzpunkt hat folglich den propagierten Streckenmeter 0. Die Fahrtrichtung im Aufsetzpunkt legt das Vorzeichen fest: Vom Aufsetzpunkt in Aufsetz-Fahrtrichtung nach vorne wächst der propagierte Streckenmeter an, in die andere Richtung nimmt er ab. Im Modell wird lediglich der propagierte Streckenmeter gespeichert, an dem die Ports eines Knotens liegen:

Definition 5.2 (Erweiterung von ***Port***)
Zusätzlich zu den bisher genannten Elementen von Objekten $p \triangleright \textbf{\textit{Port}}$ enthält p eine Zahl $s_p \in \mathbb{R}$, den propagierten Streckenmeter, an dem p liegt. Beim Start der Simulation ist $p.s_p = \infty$ [5]. •

Die Werte $p.s_p$ werden bei der geometrischen Instanzierung eines Knotens v, also in der Methode $v.DoGeoInst$ berechnet. Der Port $v.p_i$, der an den Knoten v', von dem aus $v.Update$ aufgerufen wurde, über den Port $v'.p_j$ anschließt, hat den propagierten Streckenmeter $v'.p_j.s_p$. Die propagierten Streckenmeter der anderen Ports von v lassen sich über den Lageplan ermitteln, falls v vom Typ ***Course***, ***CourseLaneCell*** oder ***CourseLane*** ist. Wenn v vom Typ ***Area***, ***AreaLaneCell*** oder ***AreaLane*** ist, dann werden die propagierten Streckenmeter aller Ports auf $v'.p_j.s_p$ gesetzt. Auf die Berechnung der propagierten Streckenmeter von Ports wird, je nach Modellebene, in den Abschnitten 5.1.5 bis 5.1.8 eingegangen.

[5] Mit dem Wert ∞ ist eine Zahl gemeint, die in jedem Fall größer ist, als alle propagierten Streckenmeter, die während einer Simulation auftreten können.

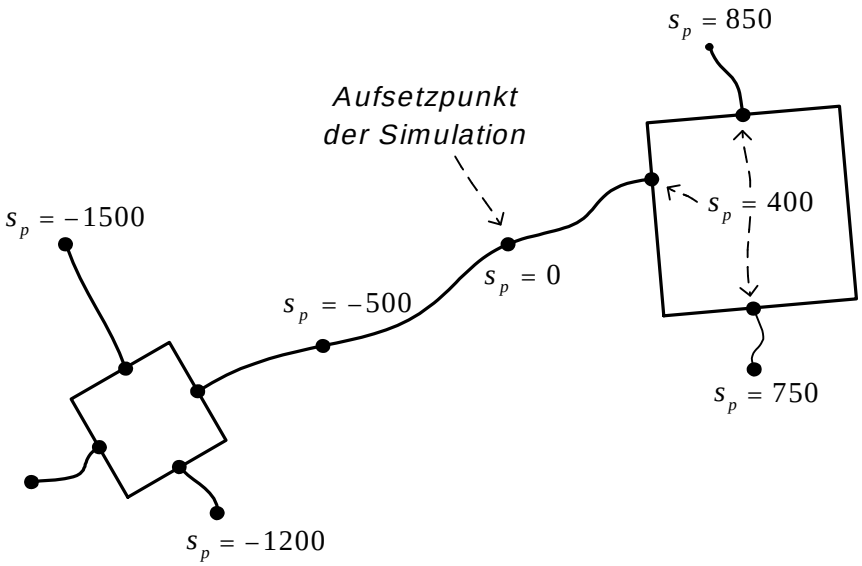

Abbildung 5.5.: Straßennetzwerk auf der Ebene RN (nicht maßstabsgetreu). Die Zahlen an den Ports sind die propagierten Streckenmeter. Alle Ports eines Verkehrsknotens haben denselben propagierten Streckenmeter. Deswegen besitzt ein Verkehrsknoten im modifizierten Sichtbarkeitsbereich keine Ausdehnung.

Abbildung 5.5 zeigt ein Straßennetzwerk auf der Ebene RN, in das die propagierten Streckenmeter einiger Ports eingetragen sind.

Mit diesen Informationen lässt sich der Sichtbarkeitstest in Zeile 25 von Algorithmus 5.2 präzisieren. Hierzu sind zwei Fälle zu unterscheiden.

1. Der aktuelle Knoten v ist der erste, von dem aus in einem Simulationsschritt $v.Update$ aufgerufen wurde (d.h. v_{caller} exisitiert nicht). In diesem Fall ist ein Port p die Strecke $|s_p^\star - p.s_p|$ vom Startpunkt $s_p^\star$ der geometrischen Instanzierung eines Simulationsschritts entfernt. Damit ist von Port p aus noch die Strecke

$$d'_p = S_{p,max} - |s_p^\star - p.s_p|$$

   sichtbar, wobei $S_{p,max}$ die Schranke für den Sichtbarkeitsbereich angibt. Folglich ist Port p sichtbar, wenn $d'_p > 0$ ist. Die Parameter $S_{p,max}$ und $s_p^\star$ werden der Methode $v.Update$ übergeben.

2. Die Methode $v.Update$ wurde von einem anderen Knoten v_{caller} aus aufgerufen. Als Parameter wird der Teil d_p des Sichtbarkeitsbereichs übergeben, der von v aus noch sichtbar ist. Ein Port p ist dann sichtbar, wenn

$$d'_p = d_p - |p_{from}.s_p - p.s_p| > 0$$

   ist.

Der Wert d'_p wird den Update-Methoden der an v angeschlossenen Knoten übergeben (Zeile 27 von Algorithmus 5.2). In Zeile 6 von Algorithmus 5.2 ist zu prüfen, ob dieser übergebene Wert $d_p \leq 0$ ist.

Ein Knoten gilt als sichtbar, wenn sich der Fahrer gerade auf ihm befindet oder mindestens einer seiner Ports sichtbar ist. Bei einem unsichtbaren Knoten sind auch die propagierten Streckenmeter seiner Ports nicht definiert. Vor dem ersten Aufruf der geometrischen Instanzierung beim Start der Simulation ist noch kein Knoten sichtbar. Deswegen werden die propagierten Streckenmeter der Ports gem. Definition 5.2 mit ∞ initialisiert. Fällt ein Knoten aus dem Sichtbarkeitsbereich heraus, dann werden die Streckenmeter seiner Ports wieder auf ∞ gesetzt. In Algorithmus 5.2 geschieht dies in Zeile 9.

Der Parameter dir_p von Algorithmus 5.2 gibt an, ob ein Knoten v, vom ersten Knoten der geometrischen Instanzierung im Zeitschritt T aus, in

Richtung zunehmender ($dir_p = 1$) oder abnehmender ($dir_p = 0$) propagierter Streckenmeter liegt. Der Wert von dir_p, der den Methoden $Update$ der an v anschließenden Knoten übergeben wird, wird in Zeile 27 bestimmt. Die Vorgehensweise soll hier nur kurz skizziert werden: Falls v selbst der erste Knoten der geometrischen Instanzierung ist, kann dir_p leicht über den Vergleich von s_p^{star} mit den propagierten Streckenmetern der Ports ermittelt werden. Andernfalls kann der an $v.Update$ übergebe Wert von dir_p an die Methoden $Update$ der anschließenden Knoten weitergereicht werden.

Zu beachten ist, dass der Parameter $S_{p,max}$ genügend groß gewählt werden muss, damit der tatsächliche (euklidische) Sichtbarkeitsbereich innerhalb des modifizierten liegt. In der Praxis hat sich gezeigt, dass der Wert $S_{p,max} = 2km$ für die meisten Straßennetzwerke ausreicht. Probleme können jedoch bei "ungünstigen" Lageplänen wie in Abbildung 5.6 auftreten.

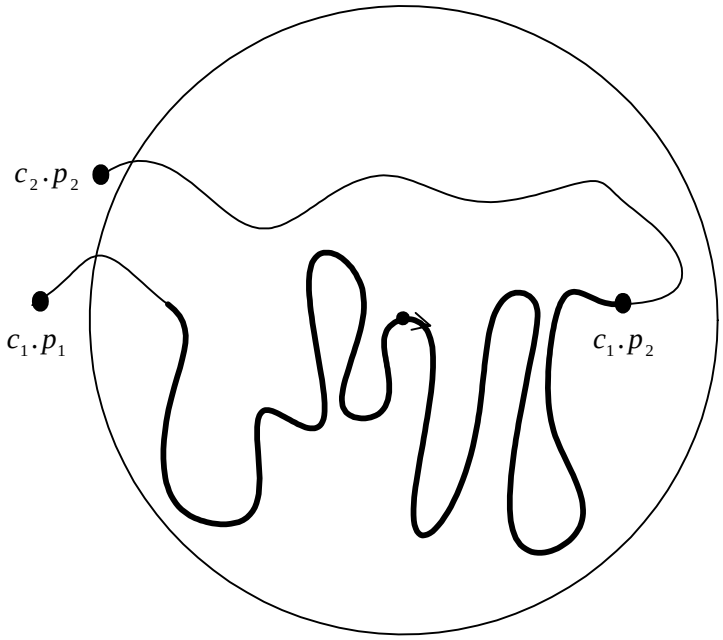

Abbildung 5.6.: Der modifizierte Sichtbarkeitsbereich (fett) kann im tatsächlichen (Kreis) enthalten sein.

Die Strecken c_1 und c_2 liegen beide im tatsächlichen Sichtbarkeitsbereich (Kreis). Der modifizierte Sichtbarkeitsbereich (fett) liegt innerhalb des tatsächlichen. Nähert der Fahrer sich c_2, dann bemerkt er die geometrische Instanzierung von c_2 dadurch, dass plötzlich eine Straße vor und neben ihm

auftaucht. In solchen Fällen müssen die Lagepläne der Straßen entsprechend angepasst werden.

5.1.4. Repräsentation von geometrisch konsistenten Teilgraphen

Auf den einzelnen Modellebenen sind die Kanten der Graphen in der Menge $p.Edges$ von Ports p gespeichert. Es ist naheliegend, die Auswahl der Kanten, die zum geometrisch konsistenten Teilgraphen gehören, ebenfalls in den Ports zu speichern. Für jeden sichtbaren Port eines Knotens wird diese Auswahl in Zeile 26 von Algorithmus 5.2 getroffen. Für die Verfahren, die das erledigen, wird in den Ports ein Zeitstempel $p.T_{update}$ benötigt (ähnlich $v.T_{update}$ bei $v \triangleright \boldsymbol{Node}$).

Definition 5.3 (Erweiterung von ***Port***)
Zusätzlich zu den bisher genannten Eigenschaften gilt für ein Objekt p des Typs ***Port***:

1. p enthält eine Menge $ActiveEdges \subseteq Edges$. Beim Start der Simulation ist diese Menge leer.

2. Aus
$$v'.p' \in v.p.ActiveEdges$$
folgt stets
$$v.p \in v'.p'.ActiveEdges$$
Dabei ist $v, v' \triangleright \boldsymbol{Node}$.

3. p enthält einen Zeitstempel $T_{update} \in \mathbb{N}_0$. Beim Start der Simulation ist $p.T_{update} = 0$. •

Zur Bestimmung von $p.ActiveEdges$ wird im Grundtyp ***Port*** eine rein virtuelle Methode $SelectActiveEdges$ vorgesehen, deren Aufruf in Zeile 26 von Algorithmus 5.2 erfolgt:

$p.SelectActiveEdges(T)$

Diese Methode wird je nach Modellebene unterschiedlich implementiert, die Beschreibung der Implementation findet sich entsprechend in den Abschnitten 5.1.5 bis 5.1.8.

5.1.5. Ebene RN

Selektion der aktiven Kanten

In jedem Simulationsschritt wird als erstes die Methode $v.Update$ der Knoten v auf der Ebene RN aufgerufen[6]. Die Ports dieser Knoten sind vom Typ ***RNPort***. Für Ports dieses Typs kann der Anwender bei der formalen Beschreibung Bedingungen zur Selektion der aktiven Kanten angegeben[7]. Die Bedingungen werden in in ***RNPort*** gespeichert:

Definition 5.4 (Erweiterung von ***RNPort***)
Zusätzlich zu den bisher genannten Elementen und Eigenschaften enthält ein Objekt p des Typs ***RNPort*** eine Funktion

$$p.sel : \mathbb{R} \rightarrow p.Edges$$

Wenn bei der formalen Beschreibung der Datenbasis nicht anders definiert, dann gilt:

$$p.sel(x) = v_1.p_{i_1} \qquad \forall x \in \mathbb{R}$$

wobei $v_1.p_{i_1}$ das erste Element aus $p.Edges$ ist. •

Die Funktion $p.sel(x)$ weißt also jedem Wert von x *eine* Kante aus $p.Edges$ zu. Bei x handelt es sich um einen Parameter, den ein Softwaremodul des Fahrsimulators zur Verfügung stellt.

Damit kann die Methode $p.SelectActiveEdges$ von $p \triangleright$ ***RNPort*** implementiert werden:

[6] vgl. Algorithmus 5.1, S. 138.
[7] vgl. Grammatik 3.10, S. 42.

Algorithmus 5.3 ***RNPort*** :: $SelectActiveEdges(T)$

```
1:  if T_update < T − 1 then
2:      if Edges ≠ ∅ then
3:          ActiveEdges := p.sel(x)
4:          Mit ActiveEdges = {v'.p'}:
5:          v'.p'.ActiveEdges := {v.p}
6:          v'.p'.T_update := T
7:      else
8:          ActiveEdges := ∅
9:      end if
10: end if
11: T_update := T
```

In den Zeilen 4 - 6 wird als aktive Kante von $v'.p'$ die Kante (v, p) eingetragen und der Zeitstempel $v'.p'.T_{update}$ auf den aktuellen Wert gesetzt. Dadurch erfolgt keine erneute Bestimmung der aktiven Kante, wenn von $v'.Update$ aus die Methode $v'.p'.SelectActiveEdges(T)$ aufgerufen wird.

Geometrische Instanzierung von Straßen

In der Methode $v.DoGeoInst$ eines Knotens v vom Typ ***Course*** müssen zum einen die propagierten Streckenmeter der beiden Ports ermittelt werden. Zum anderen muss dafür gesorgt werden, dass das Höhenprofil $v.hp$ glatt an das des aufrufenden Knotens, der vom Typ ***Course*** oder ***Area*** sein kann, anschließt. Im Überblick sieht die Methode also so aus:

Algorithmus 5.4 ***Course*** :: $DoGeoInst(p_{from}, v_{caller}, p_{caller}, dir_p)$

Prop. Streckenmeter der Ports p_1, p_2 und deren Generalisierungen (falls vorhanden) berechnen

Je nachdem, wie v_{caller} anschließt, Höhenkurve initialisieren

Die folgenden Abschnitte erklären die Schritte genauer.

Berechnung der Port-Positionen:

Zunächst wird in $s_{p,c}$ der propagierte Streckenmeter des Ports p_{caller} gespeichert, über den v_{caller} mit v verbunden ist. Hierbei muss berücksichtigt werden, dass v derjenige Knoten sein kann, von dem aus die geometrische Instanzierung im ersten Simulationsschritt gestartet wurde und deshalb kein v_{caller} existiert:

if kein v_{caller} **then**
  $s_{p,c} := 0$
else
  $s_{p,c} := v_{caller}.p_{caller}.s_p$
end if

Mit

$$S = \sum_{i=1}^{n_c} lc_i.refcurve.S$$

wird die Gesamtlänge der durch v repräsentierten Strecke bezeichnet. Wenn $v.Update$ vom Port $v.p_1$ aus aufgerufen wurde (d.h. $p_{from} = v.p_1$), dann ist die Position von $v.p_1$ gleich der von p_{caller}. Wenn die geometrische Instanzierung in Richtung wachsender propagierter Streckenmeter erfolgt, also $dir_p = 1$ ist, dann ist die Position von $v.p_2$ gleich der von $v.p_1$ plus der Streckenlänge S. Im Fall $dir_p = 0$ muss S entsprechend von der Position von $v.p_1$ abgezogen werden. Die Berechnung der Port-Positionen erfolgt analog, wenn $v.Update$ vom Port $v.p_2$ aus aufgerufen wurde, wie folgender zusammenfassender Pseudo-Code zeigt:

if $p_{from} = p_1$ **then**
  $p_1.s_p := s_{p,c}$
  if $dir_p = 1$ **then**
    $p_2.s_p := s_{p,c} + S$
  else
    $p_2.s_p := s_{p,c} - S$
  end if
else
  $p_2.s_p := s_{p,c}$
  if $dir_p = 1$ **then**
    $p_1.s_p := s_{p,c} + S$
  else
    $p_1.s_p := s_{p,c} - S$
  end if
end if

Nach diesem Verfahren steht die Position eines Ports p_i fest. Wenn p_i eine Generalisierung $p_i.Generalization = v_g.p_g$ besitzt (v_g ist dann vom Typ ***Module***), dann wird $p_g.s_p := p_i.s_p$ gesetzt. Dadurch werden die Port-Positionen von Modulen bereits bei der geometrischen Instanzierung auf der Ebene RN berechnet.

Initialisierung der Höhenkurve:
Die Initialisierung der Höhenkurve besteht aus der Berechnung von $v.hp.invert$, $v.hp.offset$ und $v.hp.h_o$. Wenn v_{caller} vom Typ ***Area*** ist, dann ist $v_{caller}.H$ das Höhenoffset, mit dem v and v_{caller} anschließt. Die Richtung der Parametrisierung muss invertiert werden, wenn die geometrische Instanzierung entlang wachsender Streckenmeter zu v gelangte ($dir_p = 1$) und v mit dem Ende, also $v.p_2$ an v_{caller} anschließt. Bei der geometrischen Instanzierung entlang abnehmender Streckenmeter wird invertiert, wenn $v.p_1$ an v_{caller} anschließt.
Wenn v_{caller} vom Typ ***Course*** ist, ist $v.hp.h_o$ die Höhe, die der Port p_{caller} von v_{caller} hat. Die Parametrisierung wird invertiert, wenn die beiden Knoten über $v.p_i, v_{caller}.p_i$ verbunden sind (also "Beginn" mit "Beginn" oder "Ende" mit "Ende") und die Höhenkurve von v_{caller} nicht invertiert ist. Analog muss die Parametrisierung von $v.hp$ ebenfalls invertiert werden, wenn die Knoten über $v.p_i, v_{caller}.p_j, i \neq j$ verbunden sind, die Höhenkurve von v_{caller} jedoch invertiert ist.
Dies und die Bestimmung von $v.hp.offset$ zeigt folgender Code-Ausschnitt:

if kein v_{caller} **then**
  // v ist erster Knoten des Updates im ersten Simulationsschritt
  $v.hp.invert := 0$
  $v.hp.h_o := 0$
else if $v_{caller} \triangleright$ ***Area*** **then**
  $v.hp.h_o := v_{caller}.h$
  if $dir_p = 1$ **then**
    $v.hp.invert := (p_{from} = v.p_2)$
  else
    $v.hp.invert := (p_{from} = v.p_1)$
  end if
else
  if $p_{caller} = v_{caller}.p_1$ **then**
    // v_{caller} schließt mit Beginn an
    $v.hp.h_o := v_{caller}.hp.h(0)$
  else
    // v_{caller} schließt mit Ende an
    $v.hp.h_o := v_{caller}.hp.h(v_{caller}.hp.S_c)$
  end if
end if
if $p_{from} = v.p_1$ **then**
  // Höhenoffset bezieht sich auf Beginn

$v.hp.offset := 0$
else
// Höhenoffset bezieht sich auf Ende
$v.hp.offset := v.hp.S_c$
end if

Geometrische Instanzierung von Verkehrsknoten

Für $v \triangleright \textbf{\textit{Area}}$ sieht $v.DoGeoInst$ so aus:

Algorithmus 5.5 $\textbf{\textit{Area}} :: DoGeoInst(p_{from}, v_{caller}, p_{caller}, dir_p)$

```
1:  // Prop. Streckenmeter der Ports berechnen
2:  for all p ∈ Ports do
3:      p.s_p := p_caller.s_p
4:      Prop. Streckenmeter der Generalisierung von p (falls vorhanden) festlegen.
5:  end for
6:
7:  // Ermittlung des Höhenoffsets
8:  if p_caller = v_caller.p_1 then
9:      H := v_caller.hp.h(0)
10: else
11:     H := v_caller.hp.h(v_caller.hp.S_c)
12: end if
```

Zeile 3: Wie in Abschnitt 5.1.3 erläutert, wird im modifizierten Sichtbarkeitsbereich die Ausdehnung von Verkehrsknoten vernachlässigt. Dies wird erreicht, indem die propagierten Streckenmeter aller Ports des Verkehrsknotens auf den Wert $p_{caller}.s_p$ gesetzt werden.

Zeile 4: Wenn p eine Generalisierung $p.Generalization = v_g.p_g$ besitzt (v_g ist dann vom Typ ***Module***), dann wird $p_g.s_p := p.s_p$ gesetzt. Dadurch werden die Port-Positionen von Modulen bereits bei der geometrischen Instanzierung auf der Ebene RN berechnet.

Zeile 8 - 12: Das Höhenoffset $v.H$ ist die Höhe, auf der $v_{caller}.p_{caller}$ liegt. Entsprechend wird in Zeile 9 bzw. 11 die Höhe in diesem Punkt abgefragt.

5.1.6. Ebene Modules

Port-Selektion

Im Falle $v \triangleright \textbf{\textit{Module}}$ wird zunächst die Verfeinerung des Ports, dessen aktive Kante bestimmt werden soll, betrachtet. Bei Modulen gibt es pro Port nur eine

Verfeinerung (v_r, p_r). Hierbei gilt $v_r \triangleright \boldsymbol{Course}$ oder $v_r \triangleright \boldsymbol{Area}$. Da die geometrische Instanzierung als erstes auf der Ebene RN durchgeführt wird, ist die aktive Kante von $v_r.p_r$ bereits ermittelt, etwa $v_r.p_r.ActiveEdges = \{v'_r.p'_r\}$. Die Generalisierung (v', r') von $v'_r.p'_r$ ist dann die aktive Kante von $v.p$. In den Zeilen 6 und 7 des folgenden Algorithmus wird die aktive Kante auch in "Gegenrichtung" gesetzt.

Algorithmus 5.6 $\boldsymbol{ModulePort}$ :: $SelectActiveEdges(T)$

```
1:  if T_update < T − 1 then
2:    if Refinements ≠ ∅ then
3:      Bezeichnung: Refinements = {v_r.p_r}
4:      Bezeichnung: v_r.p_r.ActiveEdges = {v'_r.p'_r}
5:      Bezeichnung: v'_r.p'_r.Generalization = {v'.p'}
6:      ActiveEdges := {v'.p'}
7:      v'.p'.ActiveEdges := {v.p}
8:      v'.p'.T_update := T
9:    else
10:     ActiveEdges := ∅
11:   end if
12: end if
13: T_update := T
```

Geometrische Instanzierung

Bei Modulen m müssen in der Methode $m.DoGeoInst$ die propagierten Streckenmeter *nicht* berechnet werden, da sie bereits bei der geometrischen Instanzierung auf der Ebene RN festgelegt wurden (vgl. im vorhergehenden Abschnitt die Methoden $\boldsymbol{Course}$ :: $DoGeoInst$ und $\boldsymbol{Area}$ :: $DoGeoInst$).

5.1.7. Ebene LaneCells

Selektion der aktiven Kanten

Die Bestimmung der aktiven Kanten von Knoten der Ebene LaneCells nutzt ebenfalls die Tatsache, dass die aktiven Kanten auf der Ebene RN bereits ermittelt sind: Es sei $v \triangleright \boldsymbol{LaneCell}$ und p ein Port von v. Zwei Fälle sind zu unterscheiden:

Fall 1: $v.p.Generalization = \emptyset$. Dies bedeutet, dass der Port $v.p$ nicht mit einem

Port der übergeordneten Straße bzw. des übergeordneten Verkehrsknotens zusammenfällt. Als aktive Kante kommt also nur die einzige in $v.p.Edges$ enthaltene in Frage.

Fall 2: $v.p.Generalization = \{v_g.p_g\}$. Angenommen, (v'_g, p'_g) ist die bereits ermittelte aktive Kante von $v_g.p_g$ und $v'_g.p'_g.Refinements = \{v'.p'\}$. Dann gilt:

$$\begin{aligned} v.p.ActiveEdges &= \{v'.p'\} \\ v'.p'.ActiveEdges &= \{v.p\} \end{aligned}$$

Dies führt zu folgendem Algorithmus:

Algorithmus 5.7 ***LaneCellPort*** :: $SelectActiveEdges(T)$

```
1:  if T_update < T − 1 then
2:    if Edges ≠ ∅ then
3:      if Generalization = ∅ then
4:        Bezeichnung: Edges = {v'.p'}
5:        ActiveEdges := {v'.p'}
6:        v'.p'.ActiveEdges := {v.p}
7:        v'.p'.T_update := T
8:      else
9:        Bezeichnung: Generalization = {v_g.p_g}
10:       Bezeichnung: v_g.p_g.ActiveEdges = {v'_g.p'_g}
11:       Bezeichnung: v'_g.p'_g.Refinements = {v'.p'}
12:       ActiveEdges := {v'.p'}
13:       v'.p'.ActiveEdges := {v.p}
14:       v'.p'.T_update := T
15:     end if
16:   else
17:     ActiveEdges := ∅
18:   end if
19: end if
20: T_update := T
```

Geometrische Instanzierung der Spurzellen von Straßen

Tritt ein Knoten v vom Typ ***CourseLaneCell*** neu in den Sichtbarkeitsbereich ein, dann muss berechnet werden, wie die Referenzkurve von v zu drehen und

zu verschieben ist, damit die von ihr abgeleiteten Spuren glatt an die des aufrufenden Knotens v_{caller} anschließen. Hierzu muß unterschieden werden, ob v_{caller} vom Typ ***AreaLaneCell*** ist oder ebenfalls vom Typ ***CourseLaneCell***. Außerdem werden, wie bei den Knoten der anderen Ebenen auch, die propagierten Streckenmeter der Ports ermittelt:

Algorithmus 5.8 ***CourseLaneCell*** :: $DoGeoInst(p_{from}, v_{caller}, p_{caller}, dir_p)$

1: Prop. Streckenmeter der Ports berechnen.
2:
3: **if** kein v_{caller} **then**
4: $refcurve.p_o := (0, 0)$
5: $refcurve.\alpha_0 := 0$
6: $refcurve.invert := 0$
7: **else if** $v_{caller} \triangleright$ ***CourseLaneCell*** **then**
8: **if** $p_{caller} = v_{caller}.p_1$ **then**
9: $refcurve.p_o := v_{caller}.refcurve.p(0)$
10: $refcurve.\alpha_o := v_{caller}.refcurve.\alpha(0)$
11: **else**
12: $refcurve.p_o := v_{caller}.refcurve.p(v_{caller}.refcurve.S)$
13: $refcurve.\alpha_o := v_{caller}.refcurve.\alpha(v_{caller}.refcurve.S)$
14: **end if**
15: $SamePort := [(p_{from} = v.p_1) \wedge (p_{caller} = v_{caller}.p_1)] \vee [(p_{from} = v.p_2) \wedge (p_{caller} = v_{caller}.p_2)]$
16: **if** $SamePort$ **then**
17: $refcurve.\alpha_o := refcurve.\alpha_o + \pi$
18: **end if**
19: $refcurve.invert := (SamePort \wedge v_{caller}.refcurve.invert = 0) \vee (\overline{SamePort} \wedge v_{caller}.refcurve.invert = 1)$
20: **else**
21: Bezeichnung: $v_{caller}.p_{caller}.Generalization = \{v^g_{caller}.p^g_{caller}\}$
22: $refcurve.p_o := R(v^g_{caller}.\varphi) \cdot p^g_{caller}.r + v^g_{caller}.v$
23: $refcurve.\alpha_o := p^g_{caller}.\rho + v^g_{caller}.\varphi$
24: **if** $p_{from} = v.p_2$ **then**
25: $refcurve.\alpha_o := refcurve.\alpha_o + \pi$
26: **end if**
27: $refcurve.invert := [(dir_p = 0) \wedge (p_{from} = v.p_1)] \vee [(dir_p = 1) \wedge (p_{from} = v.p_2)]$
28: **end if**
29:
30: **if** $p_{from} = v.p_1$ **then**
31: $refcurve.offset := 0$
32: **else**
33: $refcurve.offset := refcurve.S$
34: **end if**

Zeile 1: Die Berechnung der propagierten Streckenmeter der Ports erfolgt analog zu der in Algorithmus 5.4: In $s_{p,c}$ wird der propagierte Streckenmeter von p_{caller} gespeichert. Es kann sein, dass v_{caller} (und damit p_{caller}) nicht existiert, nämlich wenn v der erste Knoten der geometrischen Instanzierung im ersten Simulationsschritt ist. In diesem Fall wird $s_{p,c}$ auf 0 gesetzt und damit der propagierte Streckenmeter verankert. Der Port p_{from} hat denselben propagierten Streckenmeter wie p_{caller}. Der andere Port von v liegt um die Strecke $v.refcurve.S$ von p_{from} entfernt. Ob der Port in Richtung zunehmender oder abnehmender propagierter Streckenmeter liegt, wird durch das Argument dir_p von $DoGeoInst$ festgelegt. Der Pseudo-Code für Zeile 1 lautet:

if kein v_{caller} **then**
  $s_{p,c} := 0$
else
  $s_{p,c} := v_{caller}.p_{caller}.s_p$
end if
if $p_{from} = p_1$ **then**
  $p_1.s_p := s_{p,c}$
  if $dir_p = 1$ **then**
    $p_2.s_p := s_{p,c} + refcurve.S$
  else
    $p_2.s_p := s_{p,c} - refcurve.S$
  end if
else
  $p_2.s_p := s_{p,c}$
  if $dir_p = 1$ **then**
    $p_1.s_p := s_{p,c} + refcurve.S$
  else
    $p_1.s_p := s_{p,c} - refcurve.S$
  end if
end if

Zeile 3 - 6: Wenn v die Spurzelle ist, auf deren Port $v.p_1$ der Fahrer beim Start der Simulation aufgesetzt wird, dann beginnt die durch $v.refcurve$ modellierte Straße im Ursprung mit Tangentenwinkel 0.

Zeile 7 - 18 (Überblick): Dieser Abschnitt des Algorithmus behandelt den Fall, dass der aufrufende Knoten vom Typ ***CourseLaneCell*** ist. Es werden die Transformationsparameter $v.refcurve.p_o$, $v.refcurve.\alpha_o$ berech-

net. Diese Werte sind unabhängig von $v.refcurve.offset$, d.h. unabhängig davon, auf welches Ende von $v.refcurve$ sich die Transformation bezieht. Die Festlegung von $v.refcurve.offset$ erfolgt gemeinsam für $v \triangleright$ ***CourseLaneCell*** und $v \triangleright$ ***AreaLaneCell*** in den Zeilen 30-34. Die Berechnung von $v.refcurve.\alpha_o$ wird in zwei Schritte aufgeteilt: Zunächst wird der Tangentenwinkel von $v_{caller}.refcurve$ an dem Port ermittelt, mit dem v_{caller} an v anschließt (Zeile 8 - 16). Falls p_{caller} und p_{from} beide Beginn-Ports oder beide Ende-Ports sind, muss $v.refcurve$ um 180 Grad weiter gedreht werden, damit sie glatt an $v_{caller}.refcurve$ anschließt (Zeile 16 - 18).

Zeile 8 - 10: Dieser Abschnitt behandelt den Fall, dass v_{caller} mit dem Port Beginn an v anschließt. Die Referenzkurve von v muss also an die Stelle verschoben werden, an der $v_{caller}.p_1$ liegt. Die erste Komponente der Drehung ist der Tangentenwinkel von $v_{caller}.refcurve$ an der Stelle von $v_{caller}.refcurve.p_1$.

Zeile 11 - 14: v_{caller} ist über den Port Ende mit v verbunden. Entsprechend ist $v.refcurve.p_o$ die Position dieses Ports und $v.refcurve.\alpha_o$ der Tangentenwinkel von $v_{caller}.refcurve$ in diesem Port.

Zeile 15: Die Boole'sche Variable $SamePort$ ist wahr, wenn p_{from} und p_{caller} beide Beginn-Ports oder beide Ende-Ports sind.

Zeile 16 - 18: Falls $SamePort$ wahr ist, wird $v_{caller}.refcurve$ um zusätzliche 180 Grad gedreht.

Zeile 19: Über $refcurve.invert$ wird sichergestellt, dass sich die Durchlaufrichtung der Referenzkurven von LaneCell zu LaneCell nicht ändert.

Zeile 20 - 28 (Überblick): v_{caller} ist vom Typ ***AreaLaneCell***.

Zeile 21: Es gilt: $v^g_{caller} \triangleright$ ***Area***.

Zeile 22: Die Referenzkurve $v.refcurve$ muss an die Position des Ports $v^g_{caller}.p^g_{caller}$ anschließen. Diese Position wird berechnet, indem der Referenzpunkt $p^g_{caller}.r$, der im Entwurfskoordinatensystem gegeben ist, wie die gesamte Area um den Winkel $v^g_{caller}.\varphi$ gedreht und um $v^g_{caller}.v$ verschoben wird. Die Drehung erfolgt durch die Matrix R.

Zeile 23 - 26: Im Entwurfskoordinatensystem ist $p^g_{caller}.\rho$ der Tangentenwinkel einer Spur, die von diesem Port ausgeht. Die Area v^g_{caller} ist um den Winkel

$v^g_{caller}.\varphi$ gedreht. Wenn v mit dem Port $v.p_1$ an p^g_{caller} anschließt, die Referenzkurve also von diesem Port ausgeht, muss $v.refcurve$ mit dem Winkel $p^g_{caller}.\rho + v^g_{caller}.\varphi$ anschließen. Wenn $v.refcurve$ in den Port p^g_{caller} einläuft, also $p_{from} = v.p_2$ ist, dann muss der Winkel um π korrigiert werden (Zeile 25).

Zeile 27: Die Durchlaufrichtung von $v.refcurve$ wird so festgelegt: Wenn das Update von v entlang zunehmender propagierter Streckenmeter aufgerufen wurde ($dir_p = 1$), dann wird $v.refcurve$ von p_{caller} aus gesehen von 0 bis $v.refcurve.S$ durchlaufen. Folglich muss invertiert werden, wenn v mit $v.p_2$ an p_{caller} anschließt. Bei einem Update mit $dir_p = 0$ verläuft $v.refcurve$, wieder von p_{caller} aus gesehen, von $v.refcurve.S$ bis 0. Invertiert wird also, wenn v mit $v.p_1$ and p_{caller} anschließt.

Zeile 30 - 34: Die im vorausgehenden Code berechneten Werte $v.refcurve.p_o$ und $v.refcurve.\alpha_o$ beziehen sich auf den Parameterwert 0, wenn die geometrische Instanzierung vom Port Beginn aus aufgerufen wurde. Andernfalls gelten diese Werte an der Stelle $v.refcurve.S$.

Geometrische Instanzierung der Spurzellen von Verkehrsknoten

Neben den propagierten Streckenmetern der Ports (wie bei allen Knotentypen) wird in der Methode $v.DoGeoInst$ von Knoten $v \triangleright$ ***AreaLaneCell*** berechnet, wie der Verkehrsknoten $v.parent$ gedreht und verschoben werden muss, damit die Spuren von v_{caller} und v glatt aneinander schließen. Obwohl diese Transformationsparameter Elemente des Verkehrsknotens sind, werden sie auf der Ebene LaneCells berechnet. Der Grund dafür ist, dass die zu ihrer Berechnung nötigen Informationen (Tangentenwinkel und Position von $v_{caller}.p_{caller}$) erst auf dieser Ebene zur Verfügung stehen. Zu beachten ist, dass mit diesen Transformationsparametern wegen Definition 4.43 (S. 121) auch alle Spuren von $v.parent$ gedreht und verschoben werden.

Algorithmus 5.9 ***AreaLaneCell*** :: $DoGeoInst(p_{from}, v_{caller}, p_{caller}, dir_p)$

```
1:  prop. Streckenmeter der Ports berechnen.
2:
3:  if v_caller ▷ CourseLaneCell then
4:      if p_caller = v_caller.p_1 then
5:          x := v_caller.refcurve.p(0)
6:          ξ := v_caller.refcurve.α(0)
7:      else
8:          x := v_caller.refcurve.p(v_caller.refcurve.S)
9:          ξ := v_caller.refcurve.α(v_caller.refcurve.S)
10:     end if
11:     Bezeichnung: v.p_from.Generalization = {v_g.p_g}
12:     parent.φ := ξ − p_g.ρ
13:     if p_caller = v_caller.p_2 then
14:         parent.φ := parent.φ + π
15:     end if
16:     parent.v := x − R(parent.φ) · p_g.r
17: end if
```

Zeile 1: In Abschnitt 5.1.3 wurde die Annahme gemacht, dass Verkehrsknoten im modifizierten Sichtbarkeitsbereich keine Ausdehnung haben. Dies wird erreicht, indem alle propagierten Streckenmeter der Ports von Spurzellen auf den Wert $p_{caller}.s_p$ gesetzt werden:

for all $p \in Ports$ **do**
  $p.s_p := p_{caller}.s_p$
end for

Zeile 3: Für $v_{caller} \triangleright$ ***AreaLaneCell*** gibt es zwei Möglichkeiten: Wenn $v_{caller}.parent = v.parent$ ist, dann wurden die Transformationsparameter v und φ des übergeordneten Verkehrsknotens $v.parent$ in einem früheren Update-Schritt bereits berechnet. Der andere Fall $v_{caller}.parent \neq v.parent$ kann unter der Einschränkung, dass keine zwei Verkehrsknoten direkt miteinander verknüpft werden dürfen, nicht auftreten. Deswegen sind die Transformationsparameter des übergeordneten Verkehrsknotens $v.parent$ nur zu berechnen, wenn die geometrische Instanzierung von v von einer ***CourseLaneCell*** aus aufgerufen wurde.

Zeile 4-10: In x und ξ werden die Position und der Tangentenwinkel gespeichert, mit denen v_{caller} an v anschließt. Wenn p_{caller} der Beginn-Port ist,

dann werden diese Werte an Position 0 der Referenzkurve von v_{caller} abgefragt, andernfalls an der Position $v_{caller}.refcurve.S$.

Zeile 11: Es ist $v_g = v.parent$.

Zeile 12-14: Im Entwurfskoordinatensystem haben von $v_g.p_g$ ausgehende Spuren in diesem Port den Winkel $v_g.p_g.\rho$. Wenn v_{caller} mit Beginn an v anschließt, muss v_g um die Differenz zwischen dem Tangentenwinkel ξ von v_{caller} und $v_g.p_g.\rho$ gedreht werden. Dieser Winkel muss um π korrigiert werden, wenn v_{caller} mit Ende anschließt.

Zeile 16: Der Punkt $v_g.p_g.r$ ist die Position des Ports $v_g.p_g$ im Entwurfskoordinatensystem. Dieser Punkt wird zunächst um den in den Zeilen 12 - 14 ermittelten Winkel gedreht (R ist die Rotationsmatrix aus Satz B.1). Der Verkehrsknoten $v.parent$ muss um die Differenz zwischen der Position x von $v_{caller}.p_{caller}$ und der gedrehten Portposition verschoben werden.

5.1.8. Ebene Lanes

Selektion der aktiven Kanten

Die Bestimmung der aktiven Kanten von Ports p der Ebene Lanes nutzt die bereits ermittelten aktiven Kanten aus der übergeordneten Ebene LaneCells. In der Generalisierung $p.Generalization = \{v_g.p_g\}$ ist also $v_g.p_g.ActiveEdges = \{v'_g.p'_g\}$ bereits bekannt. Dann wird diejenige Verfeinerung $v'.p' \in v'_g.p'_g.Refinements$ gesucht, für die $v'.p' \in p.Edges$ gilt. Dies ist genau die aktive Kante von p.

Algorithmus 5.10 ***LanePort*** :: $SelectActiveEdges(T)$

if $T_{update} < T - 1$ **then**
  if $Edges \neq \emptyset$ **then**
    Bezeichnung: $Generalization = \{v_g.p_g\}$
    Bezeichnung: $v_g.p_g.ActiveEdges = \{v'_g.p'_g\}$
    Finde $v'.p' \in v'_g.p'_g.Refinements$ mit $v'.p' \in p.Edges$
    $ActiveEdges := \{v'.p'\}$
    $v'.p'.ActiveEdges := \{v.p\}$
    $v'.p'.T_{update} := T$
  else
    $ActiveEdges := \emptyset$
  end if
end if
$T_{update} := T$

Geometrische Instanzierung der Spuren von Straßen

Beim Eintritt eines Knotens $v \triangleright$ ***CourseLane*** in den Sichtbarkeitsbereich sind drei Dinge zu erledigen: Die propagierten Streckenmeter der Ports werden berechnet, die Fahrtrichtung der Spur wird festgelegt und es wird festgestellt, in welche Richtung die Referenzkurve der Spur zu durchlaufen ist.

Algorithmus 5.11 ***CourseLane*** :: $DoGeoInst(p_{from}, v_{caller}, p_{caller}, dir_p)$

```
1: prop. Streckenmeter der Ports berechnen.
2:
3: if v_caller then
4:    LaneInfo.dir := v_caller.LaneInfo.dir
5: end if
6:
7: curve.invert bestimmen.
```

Zeile 1: Da ein Port einer Spur immer als Generalisierung einen Port einer Spurzelle hat, können die propagierten Streckenmeter dieser übergeordneten Ports direkt übernommen werden.

Zeile 3-5: Wenn es eine Spur v_{caller} gibt, von der aus die geometrische Instanzierung von v aufgerufen wurde, dann wird deren Fahrtrichtung übernommen. Wenn kein v_{caller} existiert, wenn also $v.parent$ die Spurzelle ist, in

der der Fahrer beim Start der Simulation aufgesetzt wird, dann bleibt die Fahrtrichtung unverändert diejenige, die der Anwender gemäß Grammatik 3.21 (S. 58) für diese Spur angegeben hat. Die Fahrtrichtung wird also beim Aufsetzen "verankert" und dann von Spur zu Spur weitergereicht. Damit haben miteinander verknüpfte Spuren immer die gleiche Fahrtrichtung.

Zeile 7: Der Parameter $v.curve.invert = 1$ zeigt an, dass die Referenzkurve $v.parent.refcurve$ bei der Berechnung von $v.curve.p$, $v.curve.\alpha$ und $v.curve.\kappa$ in umgekehrte Richtung, d.h. von $v.parent.refcurve.S$ nach 0 durchlaufen werden muss. Dies ist der Fall, wenn in Algorithmus 5.8 (S. 158) die Durchlaufrichtung der Referenzkurve bereits invertiert wurde ($v.parent.refcurve.invert = 1$) und die Spur v die Richtung $v.LaneInfo.dir = onwards$ hat. Ebenso ist $v.curve.invert = 1$ zu setzen, wenn die Referenzkurve in normaler Richtung durchlaufen wird, die Spur v jedoch die Richtung $towards$ hat. Es gilt also:

$$\begin{aligned} v.curve.invert \;:=\; & (v.parent.refcurve.invert = 1 \wedge v.LaneInfo.dir = onwards) \vee \\ & (v.parent.refcurve.invert = 0 \wedge v.LaneInfo.dir = towards) \end{aligned}$$

Geometrische Instanzierung der Spuren von Verkehrsknoten

Die Methode $v.DoGeoInst$ von Knoten $v \triangleright \textbf{\textit{Lane}}$ muss die propagierten Streckenmeter der Ports von v berechnen. Wie bei den Spuren von Straßen gilt: Da ein Port einer Spur immer als Generalisierung einen Port einer Spurzelle hat, können die propagierten Streckenmeter dieser übergeordneten Ports direkt übernommen werden. Außerdem wird in $v.DoGeoInst$ die Fahrtrichtung der aufrufenden Spur v_{caller} übernommen:

$$v.LaneInfo.dir = v_{caller}.LaneInfo.dir$$

5.2. Schnittstelle

5.2.1. Überblick

Die meisten Softwaremodule einer Fahrsimulation benötigen Informationen aus der Datenbasis, wie folgende Beispiele zeigen:

Physikalisches Fahrzeugmodell: Dieses Modul simuliert die Dynamik eines realen Fahrzeugs. Aus den Bedieneingaben des Fahrers (Gas, Bremse, Lenkung) sowie Informationen über die Fahrbahnoberfläche wird der Zustand (Position, Geschwindigkeit, Beschleunigungen, Motordrehzahl etc.) des Fahrzeugs im nächsten Simulationsschritt berechnet. Die Informationen über die Fahrbahnoberfläche werden aus der Datenbasis erfragt.

Verkehrssimulation: Andere Fahrzeuge bewegen sich im Straßennetzwerk. Jedes davon benötigt aus der Datenbasis also Informationen über den Verlauf der Spuren, deren Breite und Richtung.

Assistenzsysteme: In der Realität beziehen Assistenzsysteme über Sensoren Daten über die Umwelt. In der Simulation werden diese Sensoren simuliert. Alle Daten, die das Assistenzsystem über die Straße benötigt, werden aus der Datenbasis bezogen.

Visualisierung: Die Visualisierung muss in jedem Simulationsschritt die Umgebung des Fahrers darstellen. Sie benötigt daher Informationen, welche Streckenabschnitte gerade sichtbar sind.

Dieser Abschnitt beschreibt die Schnittstellen, die die Datenbasis anderen Softwaremodulen bietet. Es wird zwischen zwei Typen von Informationen unterschieden. Abschnitt 5.2.2 stellt dar, wie die geometrische Instanzierung verfolgt werden kann. Damit sind Abfragen möglich, welche Elemente der Datenbasis (Spuren, Spurzellen, Strecken oder Module) im aktuellen Simulationsschritt neu in den Sichtbarkeitsbereich gekommen sind, welche sichtbar sind und welche aus dem Sichtbarkeitsbereich herausgefallen sind. Die in Abschnitt 5.2.3 beschriebenen Schnittstellen stellen Daten wie Koordinaten, Tangentenwinkel, Steigungen und Spurbreiten an bestimmten Punkten von Spuren, die sich im Sichtbarkeitsbereich des Fahrers befinden, zur Verfügung.

5.2.2. Information über die geometrische Instanzierung

Die Knoten v eines jeden Graphen der verschiedenen Modellebenen können in Bezug auf die geometrische Instanzierung in vier Mengen eingeordnet werden:

M_1 : v schließt an einen Knoten v' an, der gerade sichtbar ist. Der Port p' von v', über den v verbunden ist, ist jedoch noch nicht sichtbar. Je nach Route des Fahrers könnte v' in einem der nächsten Simulationsschritte geometrisch instanziert werden, nämlich, wenn sich der Fahrer p' nähert.

M_2 : v wurde in diesem Simulationsschritt geometrisch instanziert.

M_3 : v ist momentan sichtbar für den Fahrer. Aus $v \in M_2$ folgt stets $v \in M_3$.

M_4 : v ist in diesem Simulationsschritt aus dem Sichtbarkeitsbereich des Fahrers herausgefallen.

Objekte des Typs ***Map*** speichern diese Mengen für jede Modellebene:

Definition 5.5 (Erweiterung von ***Map***)
Zusätzlich zu den bisher genannten Elementen enthält ein Knoten $map \triangleright$ ***Map*** folgende Mengen:

1. $M_1^m, M_2^m, M_3^m, M_4^m$, jeweils mit Elementen des Typs ***Module***.
2. $M_1^{rn}, M_2^{rn}, M_3^{rn}, M_4^{rn}$, jeweils mit Elementen des Typs ***RNNode***.
3. $M_1^{lc}, M_2^{lc}, M_3^{lc}, M_4^{lc}$, jeweils mit Elementen des Typs ***LaneCell***.
4. $M_1^l, M_2^l, M_3^l, M_4^l$, jeweils mit Elementen des Typs ***Lane***. •

Für ein einfaches Straßennetzwerk zeigt Abb. 5.7 die Belegung der Mengen $M_1^{rn}, M_2^{rn}, M_3^{rn}, M_4^{rn}$ in zwei aufeinanderfolgenden Simulationsschritten.

Die Mengen werden in jedem Simulationsschritt bei der Traversierung der Graphen durch die Methode ***Node*** :: *Update* aufgebaut. Damit die Knoten entsprechend ihrer Ebene in die richtige Menge eingefügt werden, implementieren einzelne Knotentypen vier rein virtuelle Methoden von ***Node***: Die Methode $v.InsertM1()$ fügt den Knoten v in folgende Menge der Karte map ein:

- $map.M_1^m$, falls $v \triangleright$ ***Module***
- $map.M_1^{rn}$, falls $v \triangleright$ ***RNNode***
- $map.M_1^{lc}$, falls $v \triangleright$ ***LaneCell***
- $map.M_1^l$, falls $v \triangleright$ ***Lane***

Analog verfahren die Methoden $v.InsertM2()$, $v.InsertM3()$ und $v.InsertM4()$. Die Methoden $v.InsertXX()$ werden an folgenden Stellen von $v.Update(\ldots)$ (Algorithmus 5.2, S. 140) aufgerufen:

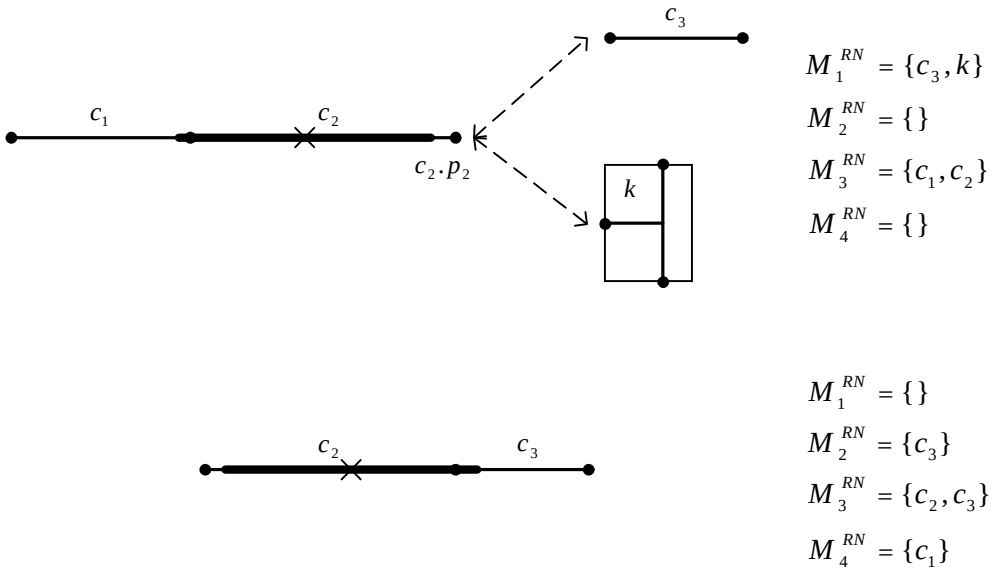

Abbildung 5.7.: Der obere Teil zeigt die Belegung der Mengen $M_1^{rn}, \ldots M_4^{rn}$ im Simulationsschritt T. Der Fahrer befindet sich an der mit 'x' markierten Position. Sein Sichtbarkeitsbereich ist fett eingezeichnet. Die Strecken c_3 und k sind über eine bedingte Verknüpfung mit $c_2.p_2$ verbunden. Der untere Teil zeigt die Belegung der Mengen im Zeitschritt $T + 1$. Der Fahrer hat sich ein Stück weiter bewegt. Über die bedingte Verknüpfung wurde c_3 selektiert.

Zeile 9: Aufruf von $v.InsertM4()$. An dieser Stelle fällt ein Knoten aus dem Teilgraphen heraus, weil er im aktuellen Zeitschritt nicht mehr im Sichtbarkeitsbereich ist, aber im letzten Zeitschritt noch darin war.

Zeile 11: Aufruf von $v.InsertM1()$. Der Knoten ist weder in diesem Simulationsschritt sichtbar, noch war er es im letzten. Allerdings schließt er an einen Knoten an, der sich gerade im Sichtbarkeitsbereich befindet.

Zeile 16: Aufruf von $v.InsertM2()$. Der Knoten war im letzen Simulationsschritt noch nicht sichtbar, in diesem ist er es aber.

Zeile 19: Aufruf von $v.InsertM3()$. Der Knoten liegt in diesem Simulationsschritt im Teilgraphen.

Die folgenden Beispiele zeigen, wie die Softwaremodule "Verkehrssimulation" und "Visualisierung" Informationen über die geometrische Instanzierung nutzen.

Die Anzahl der von der **Verkehrssimulation** gleichzeitig simulierten Fahrzeuge ist aus Gründen der Performanz beschränkt. Für die meisten Situationen, selbst für solche, die dichten Verkehr um den Fahrer herum benötigen, sind 50 Fahrzeuge ausreichend. Sobald ein Verkehrsteilnehmer den Sichtbarkeitsbereich des Fahrers verlässt, kann das "frei werdende" Fahrzeug an einer anderen Stelle des Sichtbarkeitsbereich wieder auftauchen. Es gibt zwei Möglichkeiten, wie ein Verkehrsteilnehmer aus dem Sichtbarkeitsbereich herausfallen kann: Entweder er bewegt sich selbst zu weit vom Fahrer weg oder der Fahrer entfernt sich von ihm, sodass der Knoten, auf dem er sich gerade befindet, den Sichtbarkeitsbereich verlässt. Letzteres erfragt die Verkehrssimulation in jedem Simulationsschritt über die Menge $map.M_4^l$ der aktuellen Karte map. Alle Verkehrsteilnehmer, die sich auf einer der Spuren aus dieser Menge befinden, sind nicht mehr sichtbar.

In der **Visualisierung** werden grafische Objekte (d.h. Bäume, Häuser, Felder, Seen, ...), die zur Landschaft links und rechts einer Straße c gehören, über Koordinaten (lc, s, d, β) positioniert[8]. Dabei ist lc eine Spurzelle von c, also $lc \in c.ChildNodes$, s eine Stelle auf der Referenzkurve $lc.refcurve$. Das Objekt befindet sich in einem Abstand d senkrecht zur Referenzkurve an der Stelle s und zwar links davon, wenn $d < 0$ ist. Das Objekt wird

[8] Diese kurvilinearen Koordinaten kommen mit $d = 0$ auch bei Streckenevents zum Einsatz.

um einen Winkel β zur Tangente der Referenzkurve gedreht. Diese Art der Positionierung ist in Abbildung 5.8 dargestellt.

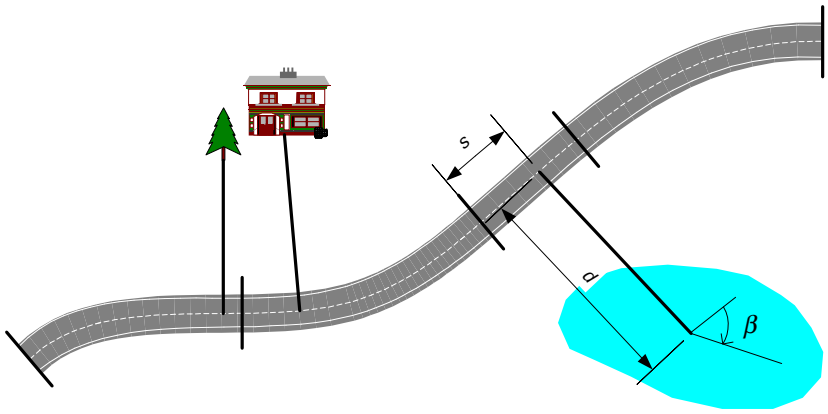

Abbildung 5.8.: Grafische Objekte in der Landschaft von Straßen, werden an einer Stelle s der Referenzkurve einer Spurzelle in einem Abstand d positioniert. Sie können zusätzlich um einen Winkel β zur Tangente der Referenzkurve gedreht werden.

Die Mengen $m.M_2^{lc}$ und $m.M_4^{lc}$ können für eine effiziente Verwaltung der grafischen Objekte benutzt werden: In jedem Simulationsschritt fügt die Visualisierung die Objekte, die zu den Spurzellen $lc \in m.M_2^{lc}$ mit $lc \triangleright \boldsymbol{CourseLaneCell}$ gehören, ihrer Liste der darzustellenden Objekte hinzu. Die Objekte der Spurzellen $lc \in m.M_4^{lc}$ mit $lc \triangleright \boldsymbol{CourseLaneCell}$ können aus dieser Liste entfernt werden. So ist sichergestellt, dass nur Objekte dargestellt werden, die für den Fahrer auch sichtbar sind.

5.2.3. Spurbezogene Informationen

Grundstruktur

Spurbezogene Informationen können über Objekte des Grundtyps ***Cursor*** erfragt werden. Ein Objekt $c \triangleright \boldsymbol{Cursor}$ repräsentiert einen Punkt auf einer *sichtbaren* Spur l der Modellebene LANES. Über Varianten einer Methode $c.Setup$ kann c auf einen bestimmten Punkt auf einer Spur gesetzt werden. Mittels Varianten der Methode $c.Move$ wird der Punkt bewegt. Jedes Objekt $c \triangleright \boldsymbol{Cursor}$ sammelt die Strecken-Events, die es bei einer solchen Bewegung "überfahren"

hat. Ferner speichern Objekte des Typs ***Cursor*** die Karte $map \triangleright \textbf{\textit{Map}}$, auf deren Sraßennetzwerk sie sich beziehen. Zusammenfassend ist ***Cursor*** wie folgt definiert:

Definition 5.6 (***Cursor***)
Ein Objekt $c \triangleright \textbf{\textit{Cursor}}$ enthält folgende Elemente:

1. Ein Objekt $map \triangleright \textbf{\textit{Map}}$.
2. Eine Spur $l \triangleright \textbf{\textit{Lane}}$. Es muss gelten:
$$l \in map.M_3^l$$
(Die Spur l muss sichtbar sein)
3. Eine Zahl $s \in [0, l.curve.S]$, die Position auf der Kurve, die den Verlauf von l beschreibt.
4. Eine Menge
$$Events = \{e_1, \ldots, e_n\}$$
von Strecken-Events. Die Elemente e_i sind dabei vom Typ ***Event***. •

Die Softwaremodule, die spurbezogene Informationen aus der Datenbasis benötigen, arbeiten mit zwei von diesem Grundtyp abgeleiteten Typen.

PrimaryCursor: Durch einen primären Cursor wird der Fahrer selbst repräsentiert. Folglich gibt es nur *ein* Objekt pc dieses Typs. Es befindet sich beim Start der Simulation am Beginn einer Spur in $pc.map.SetupPoints$. Während der Simulation bewegt sich der Fahrer frei durch die Datenbasis. Das Objekt pc repräsentiert den Punkt im Graphen der Spuren, der der aktuellen Position des Fahrers am nächsten ist.

SecondaryCursor: Softwaremodule können beliebig viele Objekte dieses Typs erzeugen. Sie werden entweder direkt auf einer Spur platziert oder an der Stelle des Fahrers aufgesetzt. Sekundäre Cursor können entlang von Spuren bewegt werden.

Die Methoden *Setup* und *Move* dieser Typen werden in den anschließenden Abschnitten beschrieben. In beiden abgeleiteten Typen werden zwei Hilfs-Methoden benötigt, die deswegen im Grundtyp ***Cursor*** implementiert werden. Diese Methoden werden im folgenden vorgestellt.

Die erste Hilfs-Methode eines Objekts $c \triangleright$ ***Cursor*** ist $c.CheckEvents(l, s_1, s_2)$. Sie fügt der Menge $c.Events$ alle Strecken-Events e_i aus $l.Events = \{e_1, \ldots, e_n\}$ hinzu, für die $s_1 \leq e_i.s \leq s_2$ gilt. Folglich muss für die Parameter dieser Methode $s_1, s_2 \in [0, l.curve.S]$ und $s_1 < s_2$ sein. Auf die genaue Wiedergabe der Implementation wird verzichtet.

Die zweite Methode, die sowohl von Objekten des Typs ***PrimaryCursor*** als auch von Objekten des Typs ***SecondaryCursor*** benötigt wird, ist $c.FindLaneS(x, y)$. Sie bekommt die Koordinaten eines beliebigen Punktes in der xy-Ebene übergeben und liefert ein Paar $(l^\star, s^\star)$. Der Punkt $l^\star.curve.p(s^\star)$ ist derjenige Punkt im sichtbaren Teilgraphen der Modellebene LANES, der dem Punkt (x, y) am nächsten ist. Gesucht wird dabei in allen Spuren, die sich im Sichtbarkeitsbereich befinden. Die Menge L der zu durchsuchenden Spuren ist folglich so definiert:

$$L := c.map.M_3^l \tag{5.1}$$

Für jede dieser Spuren $l_i \in L$ wird eine Position $s_i \in [0, l_i.curve.S]$ bestimmt, so dass der Abstand

$$d_i := ||(x, y) - l_i.curve.p(s_i)||$$

minimal ist. Wenn j der Index des kleinsten dieser Abstände ist, dann wird $l^\star := l_j$ und $s^\star := s_j$ gesetzt.

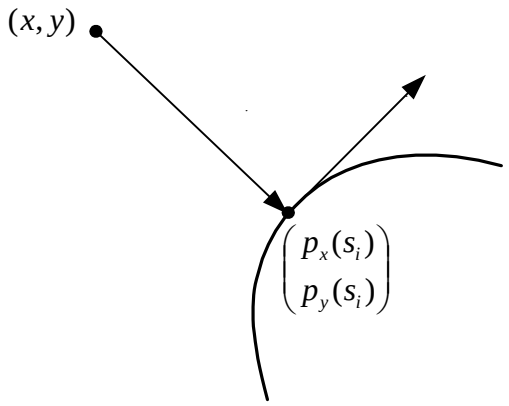

Abbildung 5.9.: An der Position s_i auf der Spur l_i (fett) ist der Abstand zum Punkt (x, y) minimal.

Es bleibt noch zu erläutern, wie zu einer gegebenen Spur l_i die Position s_i bestimmt wird, die dem Punkt (x, y) am nächsten ist: Der Abstand von (x, y) zu $l_i.curve.p(s_i) =: (p_x(s_i), p_y(s_i))$ ist minimal, wenn der Tangentenvektor

$(p'_x(s_i), p'_y(s_i))$ senkrecht auf $(x - p_x(s_i), y - p_y(s_i))$ steht (Abb. 5.9), wenn also gilt:

$$< (p'_x(s_i), p'_y(s_i)), (x - p_x(s_i), y - p_y(s_i)) > \quad = 0 \tag{5.2}$$

Die im Skalarprodukt auftretenden Ableitungen der Parameterkurven sind bekannt und leicht zu berechnen. Aufgrund der schnellen Konvergenz wird die Nullstelle dieser Gleichung iterativ mit dem Newton-Verfahren bestimmt. Im Iterationsschritt $k + 1$ hat das Verfahren folgende Struktur:

$$s_i^{(k+1)} = s_i^{(k)} - \frac{N(s_i^{(k)})}{N'(s_i^{(k)})}$$

mit

$$N(s) := p'_x(s) \cdot (x - p_x(s)) + p'_y(s) \cdot (y - p_y(s))$$

Das Verfahren erlaubt einige Optimierungen:

- Wenn $l_i.curve$ eine Gerade ist, dann lässt sich Gleichung 5.2 explizit nach s_i auflösen, das Newton-Verfahren muss nicht angewendet werden.
- Bei Spuren von Straßen reicht es, den Wert s_i einmal pro Spurzelle auf der Referenzkurve zu suchen. Die Spur l_i kann anhand des Querschnittsprofils ermittelt werden.
- Unter der Annahme, dass der Punkt $l^\star.curve.p(s^\star)$ in der Nähe der aktuellen Position des Cursors $c.l.curve.p(s)$ liegt, reicht es, in der aktuellen Spurzelle des Cursors, also in allen Spuren von $lc := c.l.parent$, sowie in den unmittelbar an lc anschließenden Spurzellen zu suchen. Angenommen, mit lc sind die Spurzellen $lc_1, \ldots, lc_n$ verbunden. Die Menge L aus Gleichung 5.1 enthält dann alle Spuren von $lc, lc_1, \ldots, lc_n$, also:

$$L := lc.ChildNodes \cup lc_1.ChildNodes \cup \ldots \cup lc_n.ChildNodes$$

Repräsentation des Fahrers als primärer Cursor

Ein Objekt $pc \triangleright \boldsymbol{PrimaryCursor}$ repräsentiert den Fahrer. Die in Algorithmus 5.1 (S. 138) skizzierte Steuerung der geometrischen Instanzierung erfolgt in den Methoden $pc.Setup$ und $pc.Move$. Der primäre Cursor zählt die Simulationsschritte T:

Definition 5.7 (***PrimaryCursor***)
Der Typ ***PrimaryCursor*** ist vom Typ ***Cursor*** abgeleitet. Zusätzlich zu den Elementen des Typs ***Cursor*** enthält ein Objekt $pc \triangleright \boldsymbol{Cursor}$ eine Zahl $T \in \mathbb{N}_0$, den aktuellen Simulationsschritt. •

Über die Methode

$$pc.Setup(name)$$

wird der primäre Cursor an der Stelle $l_i.curve.p(0)$ einer Spur aus $pc.map.SetupPoints$ aufgesetzt. Der Aufsetzpunkt wird über den Namen adressiert, den der Anwender gem. Grammatik 3.22 (S. 63) angibt. Es wird also derjenige Aufsetzpunkt $(name_i, l_i) \in pc.map.SetupPoints$ gewählt, für den $name_i = name$ gilt. In diesem Punkt wird der Ursprung eines ortsfestes Koordinatensystem verankert. Die x-Achse dieses Systems zeigt in Richtung der Tangente von $l_i.curve.p(0)$. Die restlichen Achsen verlaufen wie in Abbildung 5.10 dargestellt. Das Verfahren ist in Algorithmus 5.12 skizziert.

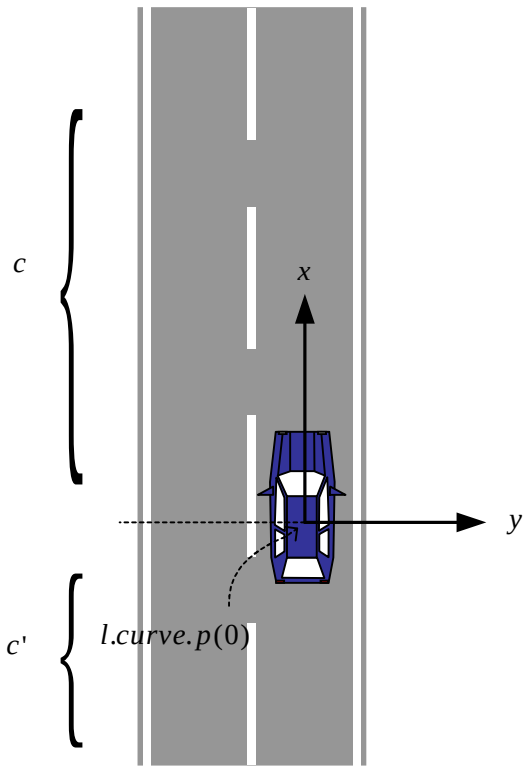

Abbildung 5.10.: Beim Start der Simulation wird der Fahrer auf eine bestimmte Spur l_i am Beginn einer Straße c aufgesetzt. Dadurch wird ein Koordinatensystem festgelegt, dessen Ursprung sich im Punkt $l_i.curve.p(0)$ befindet. Die z-Achse zeigt in dieser Darstellung aus dem Bild heraus auf den Betrachter.

Algorithmus 5.12 ***PrimaryCursor*** :: $Setup(name)$

Wähle $(name_i, l_i) \in map.SetupPoints$ mit $name_i = name$
$l := l_i,\ s := 0$
$T := 0$
Führe geom. Instanzierung auf der Ebene RN aus.
Führe geom. Instanzierung auf der Ebene MODULES aus.
Führe geom. Instanzierung auf der Ebene LANECELLS aus.
Führe geom. Instanzierung auf der Ebene LANES aus.

Die Ausführung der geometrischen Instanzierung in der Methode $pc.Setup$ wird beispielhaft an der Ebene RN erläutert. Zunächst wird die Straße c, zu der die Aufsetz-Spur $pc.l$ gehört, bestimmt:

$$c := pc.l.parent.parent$$

Danach wird $c.Update$ mit folgenden Werten für die Argumente aufgerufen:

T: Als aktueller Simulationsschritt wird $pc.T = 0$ übergeben.

$S_{p,max}$: Dieser Parameter stellt die obere Grenze für den Sichtbarkeitsbereich dar. In der Praxis hat sich ein Wert von 2000 m als sinnvoll erwiesen.

$s_p^\star$: Die geometrische Instanzierung wird vom propagierten Streckenmeter 0 aus gestartet.

d_p: Von $s_p^\star$ aus ist noch der volle Bereich $S_{p,max}$ sichtbar.

dir_p: Da es sich bei c um den Knoten handelt, auf dem sich der Fahrer gerade befindet, spielt dieser Parameter keine Rolle.

p_{from}: Die geometrische Instanzierung wird vom Port $c.p_0$ aus gestartet.

v_{caller}, p_{caller}: Es gibt keinen aufrufenden Knoten und entsprechend keinen Port, von dem aus $c.Update$ aufgerufen wurde.

Nachdem einmaligen Aufsetzen beim Start der Simulation via ***PrimaryCursor*** :: $Setup$ wird in jedem Simulationsschritt die Methode ***PrimaryCursor*** :: $Move$ aufgerufen, um die Datenbasis mit der aktuellen Position des Fahrers abzugleichen. Der Ablauf dieser Methode ist in Algorithmus 5.13 dargestellt.

Algorithmus 5.13 ***PrimaryCursor*** :: $Move(x, y)$

1: $(l^\star, s^\star) := FindLaneS(x, y)$
2: Speichere in $Events$ alle überfahrenen Strecken-Events.
3: $l := l^\star$, $s := s^\star$
4: $T := T + 1$
5: Führe geom. Instanzierung auf der Ebene RN aus.
6: Führe geom. Instanzierung auf der Ebene MODULES aus.
7: Führe geom. Instanzierung auf der Ebene LANECELLS aus.
8: Führe geom. Instanzierung auf der Ebene LANES aus.

Zeile 1: Der Fahrer befindet sich, projiziert auf die xy-Ebene, an der Stelle (x, y) in dem beim Aufsetzen festgelegten Koordinatensystem. Zu dieser Position ist der Punkt im Graphen der Ebene LANES zu bestimmen, der dem Fahrer am nächsten ist. Hierzu wird die im vorhergehenden Abschnitt beschriebene Hilfs-Methode ***Cursor*** :: $FindLaneS$ benutzt. Eine der vorgestellten Optimierungen dieses Verfahren geht davon aus, dass der Fahrer in einem Simulationsschritt höchstens in eine Spurzelle wechselt, die mit der aus dem letzten Simulationsschritt verbunden ist. Bei üblichen Simulationsfrequenzen von mindestens 30 Hz ist diese Annahme gerechtfertigt: Selbst wenn der Fahrer sich mit 300 km/h bewegt, legt er dann in einem Schritt lediglich eine Strecke von 2,8 m zurück.

Zeile 2: Nur wenn der Fahrer keinen Spurwechsel vorgenommen hat, d.h. wenn $pc.l = l^\star$ ist oder $pc.l$ mit $l^\star$ direkt verbunden ist, werden Strecken-Events aufgesammelt. Hierzu wird die im vorhergehenden Abschnitt beschriebene Hilfs-Methode ***Cursor*** :: $CheckEvents$ verwendet.

In Abbildung 5.11 sind zwei Beispiele für das Aufsammeln von Events zu sehen. Der Fahrer hat sich in einem Simulationsschritt von der Position $pc.s$ (Spur $pc.l$) zur Position $s^\star$ auf der Spur $l^\star$ bewegt. Im oberen Teil der Abbildung ist $pc.l = l^\star$. Deswegen werden in $pc.Events$ die Strecken-Events e_1 und e_2 aufgenommen. Im unteren Teil hat der Fahrer die Spur gewechselt, $pc.l$ ist nicht mit $l^\star$ verbunden. Deswegen wird keines der Strecken-Events $e_1, \ldots, e_4$ der Menge $pc.Events$ hinzugefügt.

Zeile 3: Die neu ermittelte Position des Fahrers wird übernommen.

Zeile 4-8: Die geometrische Instanzierung für den aktuellen Simulationsschritt wird durchgeführt. Dies wird wieder am Beispiel der Ebene RN darge-

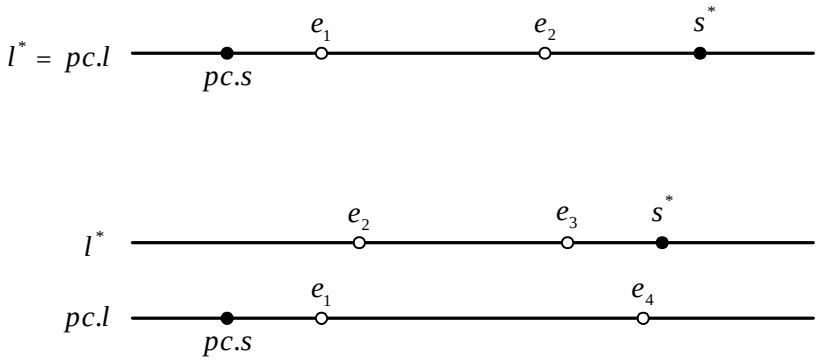

Abbildung 5.11.: Events des primären Cursors

stellt. Es sei $v := pc.l.parent.parent$ die Strecke, auf der sich der Fahrer befindet. Falls $v \triangleright \textbf{\textit{Course}}$ ist, wird die Methode $v.Update$ mit folgenden Werten für ihre Argumente aufgerufen:

T: Als aktueller Simulationsschritt wird $pc.T$ übergeben.

$S_{p,max}$: Die obere Grenze für den Sichtbarkeitsbereich wird auf 2 km gesetzt.

$s_p^\star$: Der propagierte Streckenmeter, von dem aus die geometrische Instanzierung in diesem Simulationsschritt gestartet wird, kann aus $pc.s$ und den propagierten Streckenmetern der Ports $v.p_1$ und $v.p_2$ berechnet werden.

d_p: Von $s_p^\star$ aus ist noch der volle Bereich $S_{p,max}$ sichtbar.

dir_p, p_{from}: Da es sich bei v um den Knoten handelt, auf dem sich der Fahrer gerade befindet, spielen diese Parameter keine Rolle.

v_{caller}, p_{caller}: Es gibt keinen aufrufenden Knoten und entsprechend keinen Port, von dem aus $v.Update$ aufgerufen wurde.

Da der Knoten v im Falle $v \triangleright \textbf{\textit{Area}}$ im modifizierten Sichtbarkeitsbereich keine Ausdehnung hat, wird der propagierte Streckenmeter $s_p^\star$ auf den eines beliebigen Ports von v gesetzt (alle Ports von v haben den gleichen propagierten Streckenmeter). Die restlichen Argumente von $v.Update$ entsprechen den oben aufgeführten.

Die folgenden Beispiele zeigen, wie Softwaremodule die Informationen des primären Cursor nutzen.

Neben den Eingaben des Fahrers (Lenkwinkel, Gas- und Bremspedal, Kupplung, Gangschaltung) benötigt ein **physikalisches Fahrzeugmodell** in jedem Simulationsschritt die Steigung der Fahrbahn in Richtung der Fahrzeuglängs- und Querachse. Diese Werte q_l und q_q lassen sich mithilfe des primären Cursors pc wie folgt ermitteln. Angenommen, das Fahrzeug ist bezüglich der z-Achse des ortsfesten Koordinatensystems der Datenbasis um den Winkel ψ gedreht (Abb. 5.12). Dann beträgt der Winkel, den es mit der Bahntangente an seiner aktuellen Position $pc.l.curve.p(pc.s)$ einnimmt

$$\Delta\psi = \psi - pc.l.curve.\alpha(pc.l.s)$$

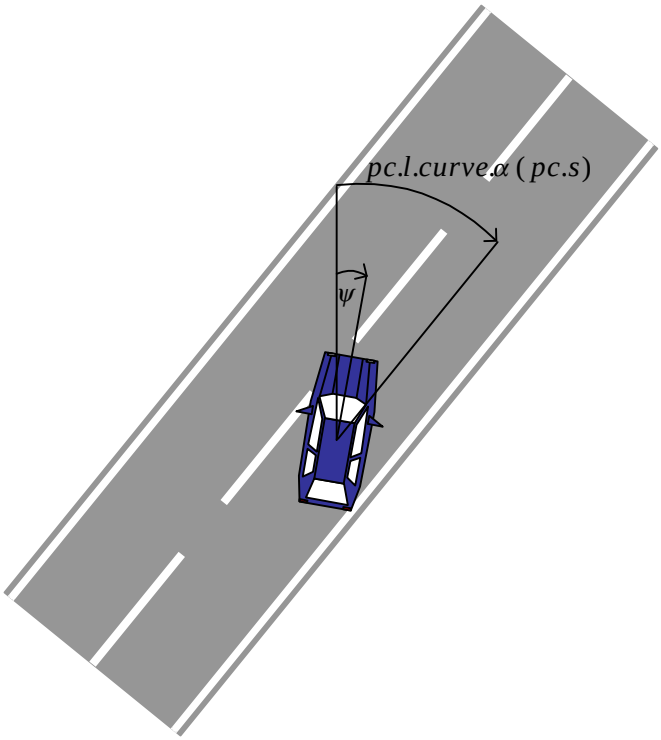

Abbildung 5.12.: Das Fahrzeug ist um den Winkel ψ bzgl. der z-Achse gedreht, der Bahntangentenwinkel an der aktuellen Position beträgt $pc.l.curve.\alpha(pc.s)$.

Die Steigung entlang der Spur ist durch $pc.l.h'(pc.s)$ gegeben. Dieser Wert wird wie folgt in die Komponenten q_l und q_q aufgeteilt:

$$\begin{aligned} q_l &= pc.l.h'(pc.s) \cdot \cos\Delta\psi \\ q_q &= pc.l.h'(pc.s) \cdot \sin\Delta\psi \end{aligned}$$

Strecken-Events kommen in der **Navigationsaufgabe** zum Einsatz[9]. Die Menge $pc.Events = \{e_1, \ldots, e_n\}$ des primären Cursors enthält die Strecken-Events, die der Fahrer seit dem letzten Simulationsschritt überfahren hat. Das Softwaremodul, das das einfache Navigationssystem implementiert, durchsucht diese Menge in jedem Schritt nach Elementen mit

$$e_i.swmod = \text{"navi"}$$

Da der Versuchsleiter diese Events in relativ großen Abständen platziert hat (vgl. Beispiel auf Seite 62), erfüllt pro Schritt höchstens eines diese Bedingung. Das Navigationssystem zeigt dann auf dem Display den Pfeil an, der in $e_i.args$ angegeben ist.

Sekundäre Cursor

Objekte sc des Typs ***SecondaryCursor*** können von Softwaremodulen benutzt werden, um Informationen an beliebigen Stellen von Spuren im Sichtbarkeitsbereich abzufragen. Zur anfänglichen Positionierung dienen drei Versionen der Methode $sc.Setup$:

$$sc.Setup(c)$$

Der sekundäre Cursor wird auf dieselbe Stelle platziert, an der sich ein anderer Cursor $c \triangleright$ ***Cursor*** befindet. In der Methode wird lediglich Spur und Parameter von c übernommen:

$$sc.l := c.l$$
$$sc.s := c.s$$

Über

$$sc.Setup(l_{setup}, s_{setup})$$

kann sc auf eine beliebige Spur $l_{setup} \triangleright$ ***Lane*** im Sichtbarkeitsbereich gesetzt werden. Folglich muss gelten:

$$l_{setup} \in sc.map.M_3^l$$

und

$$s_{setup} \in [0, l_{setup}.curve.S]$$

[9] Die Navigationsaufgabe ist auf Seite 19 erklärt.

Die Implementation sieht so aus:

$$sc.l := l_{setup}$$
$$sc.s := s_{setup}$$

Die dritte Variante

$$sc.Setup(x, y)$$

sucht denjenigen Punkt im sichtbaren Teilgraphen der Modellebene LANES, der den übergebenen ortsfesten Koordinaten (x, y) am nächsten ist. Hierzu wird die Methode $\boldsymbol{Cursor} :: FindLaneS$ benutzt:

$$(sc.l, sc.s) := FindLaneS(x, y)$$

Es handelt sich hierbei um eine Umrechnung der ortsfesten Koordinaten (x, y) in kurvilineare Koordinaten.

Nach dem das Objekt sc über eine der drei Varianten von $sc.Setup$ platziert ist, kann es über die Methode

$$sc.Move(\Delta s, heading)$$

bewegt werden. Diese Bewegung erfolgt um eine Strecke Δs von der aktuellen Position aus, und zwar in Fahrtrichtung der aktuellen Spur $sc.l$, wenn $\Delta s > 0$ ist, bzw. entgegen der Fahrtrichtung, wenn $\Delta s < 0$ ist. Algorithmus 5.14 zeigt die Implementation.
Die gesamte zurückzulegende Strecke Δs wird in Abschnitte aufgeteilt: In jedem Schleifendurchlauf erfolgt die Bewegung innerhalb *einer* Spur. Der Algorithmus benutzt intern folgende Variablen:

$stop$: Ein boolescher, Wert, der anzeigt, dass die repeat-Schleife beendet werden soll.

d : Gibt die Strecke an, die in jedem Schleifendurchlauf zurückgelegt wird.

s_{new} : Hilfsvariable, in der in jedem Schleifendurchlauf die neue Position auf der aktuellen Spur gespeichert wird.

l' : An l anschließende Spur, auf der die Bewegung im nächsten Schleifendurchlauf fortgesetzt wird.

Algorithmus 5.14 ***SecondaryCursor*** :: $Move(\Delta s, heading)$

```
1:  stop := false
2:  repeat
3:    if Δs < 0 then
4:      d := max{−s, Δs}
5:    else
6:      d := min{l.curve.S − s, Δs}
7:    end if
8:    s_new := s + d
9:    Δs := Δs − d
10:   Events aufsammeln
11:   s := s_new
12:   if Δs = 0 then
13:     stop := true
14:   else
15:     an l anschließende Spur l' mittels heading selektieren
16:     if l' exisitiert then
17:       l := l'
18:       s := Startposition auf l'
19:     else
20:       stop := true
21:     end if
22:   end if
23: until stop = true
```

Einige Kommentare zum Ablauf:

Zeile 4: Bei einer Bewegung entgegen der Fahrtrichtung der Spur darf in einem Schleifendurchlauf maximal die Strecke bis zum Spurbeginn zurückgelegt werden.

Zeile 6: Bei einer Bewegung in Fahrtrichtung der Spur darf in einem Schleifendurchlauf maximal die Strecke bis zum Spurende zurückgelegt werden.

Zeile 8: Die neue Position auf der aktuellen Spur wird berechnet.

Zeile 9: In den nächsten Schleifendurchläufen ist noch die Strecke $\Delta s - d$ zurückzulegen.

Zeile 10,11: Die Strecken-Events, die auf dem zurückgelegten Abschnitt zwischen s und s_{new} liegen, werden mittels der Methode $Cursor :: CheckEvents$ aufgesammelt. Danach kann die neue Position übernommen werden.

Zeile 13: Wenn bereits die gesamte Strecke Δs zurückgelegt wurde, kann die Schleife abgebrochen werden.

Zeile 15-21: Bei der Bewegung in Zeile 8 wurde das Ende oder der Beginn der aktuellen Spur l erreicht, es wurde aber noch nicht die gesamte Strecke zurückgelegt. Deswegen muss im nächsten Schleifendurchlauf die Bewegung auf einer Spur fortgesetzt werden, die über einen Port $l.p_i$ mit l verbunden ist und im Sichtbarkeitsbereich liegt. In Verkehrsknoten kann es vorkommen, dass mehrere Spuren $l_1, \ldots, l_n$ an $l.p_i$ anschließen. Aus diesen wird l' mittels dem den Algorithmus übergeben Parameter

$$heading \in \{straight, left, right\}$$

ausgewählt. Dabei wird die erste Spur l_i genommen, für die

$$heading \in l_i.LaneInfo.heading$$

gilt. Falls keine Spur l_i diese Bedingung erfüllt, werden ersatzweise, je nach dem Wert von $heading$, folgende Spuren genommen:

- $heading = straight$: Es wird zunächst eine Spur mit $right \in l_i.LaneInfo.heading$ gesucht. Falls nicht vorhanden wird l_1 genommen.

- $heading = right$: Es wird zunächst eine Spur mit $straight \in l_i.LaneInfo.heading$ gesucht. Falls nicht vorhanden wird l_1 genommen.
- $heading = left$: Wie $heading = right$

Abbildung 5.13 zeigt einen Ablauf von $\boldsymbol{SecondaryCursor} :: Move$ an einem einfachen Beispiel. Die Ausgangssituation ist im obersten Teil der Abbildung dargestellt. Der Cursor sc befindet sich auf der Spur l_1 an Position $s = 50$ (kleines Kreuz). Die Bewegung wird über den Aufruf $sc.Move(250, straight)$ durchgeführt. Nach dem ersten Schleifendurchlauf von Algorithmus 5.14, dargestellt im zweiten Teilbild von oben, hat der Cursor das Ende von l_1 erreicht. Es ist noch eine Strecke von $\Delta s = 200$ m zurückzulegen. Der Cursor bewegt sich im nächsten Durchgang über die gesamte Spur l_2 (drittes Teilbild von oben). Für den nächsten Durchlauf muss eine der Spuren l_3 oder l_4 selektiert werden. Angenommen, die Spuren haben folgende Richtungsangaben:

$$\begin{aligned} l_3.LaneInfo.heading &= \{left\} \\ l_4.LaneInfo.heading &= \{right\} \end{aligned}$$

Da beim Aufruf $straight$ übergeben wurde, wird die Bewegung auf l_4 fortgesetzt, wo sie an der Position $s = 60$ auch abgeschlossen ist.

Der Einsatz von Objekten des Typs $\boldsymbol{SecondaryCursor}$ soll anhand von drei Beispielen illustriert werden, einem Assistenzsystem für die Querführung, der physikalischen Fahrzeugsimulation und der Verkehrssimulation.

Eine Version eines Assistenzsystems, das den Fahrer bei der **Querführung des Fahrzeugs** unterstützt, basiert auf dem sog. "Aim Point Error" Modell (vgl. [30]). Es ermittelt die Abweichung des Fahrzeugs von der Spurmitte Δy, den Winkel $\Delta\psi$, den das Fahrzeug mit der Tangente der Spur einnimmt, sowie die Krümmung κ_{AP} der Spur in einem Punkt, der dem Fahrzeug eine bestimmte Strecke $T \cdot v$ vorausliegt. Dieser Vorausschaupunkt wird als "Aim Point" bezeichnet. Bei T handelt es sich dabei um eine Zeitkonstante (üblicherweise $T \approx 1.5$ Sekunden). v ist die aktuelle Geschwindigkeit. In der Realität werden diese Daten aus Sensoren im Fahrzeug sowie aus einer Bildverarbeitung von Videoaufnahmen der Fahrbahnmarkierungen gewonnen. Aus ihnen wird die Abweichung von der Spurmitte Δy_{AP} geschätzt, die das Fahrzeug bei konstantem Lenkradwinkel und konstanter Geschwindigkeit im Vorausschaupunkt haben wird. In Abhängigkeit von Δy_{AP} führt das System

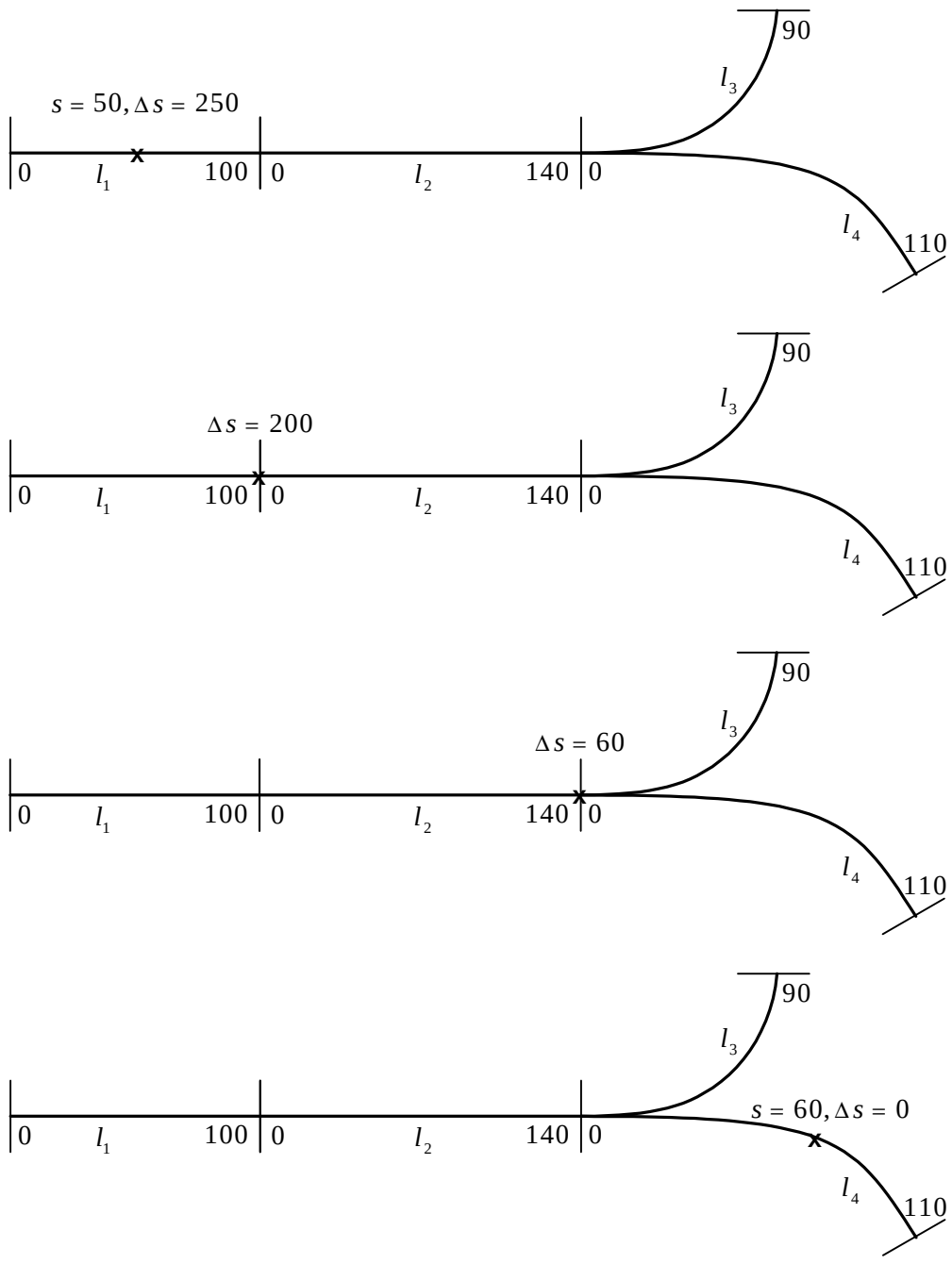

Abbildung 5.13.: Bewegung eines sekundären Cursors *sc* mittels *sc.Move* (Erläuterungen im Text).

über einen an der Lenksäule befestigten Motor korrigierende Lenkeingriffe aus. Das System lenkt nach rechts, wenn $\Delta y_{AP} < 0$ ist, und nach links, wenn $\Delta y_{AP} > 0$ ist. Der Eingriff ist umso stärker, je größer $|\Delta y_{AP}|$ ist[10].

Der Aim Point Error setzt sich aus drei Komponenten zusammen:

$$\Delta y_{AP} = \Delta y + \Delta y_{\Delta\psi} + \Delta y_{\kappa_{AP}}$$

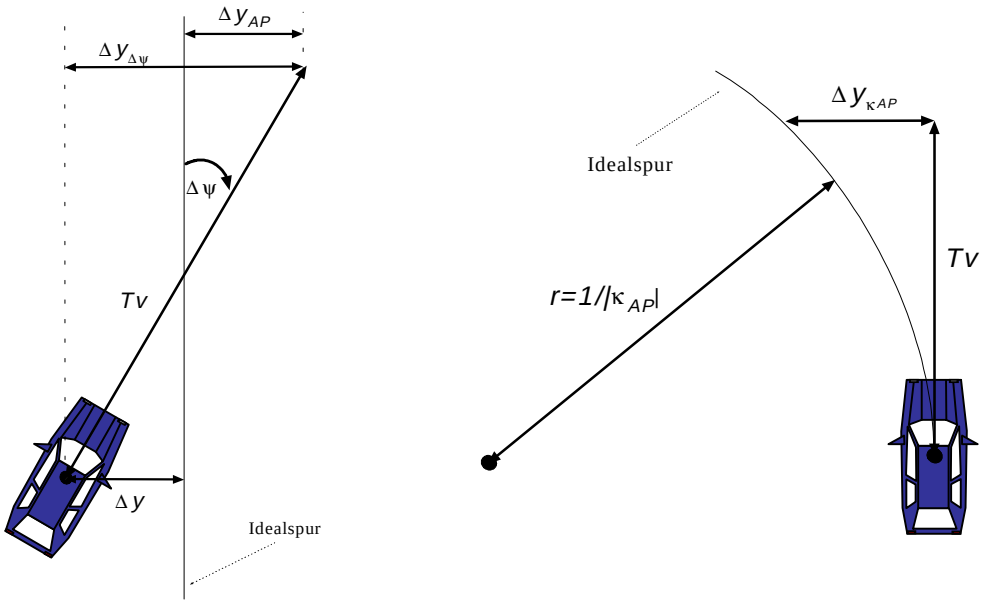

Abbildung 5.14.: Berechnung des Aim Point Errors Δy_{AP} (Erläuterungen im Text).

Der Einfluss der beiden ersten Komponenten ist aus Abbildung 5.14 (linker Teil) zu entnehmen. Zur Berechnung von $\Delta y_{\Delta\psi}$ wird ein gerader Verlauf der Idealspur angenommen:

$$\Delta y_{\Delta\psi} = T \cdot v \cdot \sin \Delta\psi$$

Zur Bestimmung von $\Delta y_{\kappa_{AP}}$ sei $\Delta y = 0$ und $\Delta\psi = 0$. Die Idealspur mit Krümmung κ_{AP} im Vorausschaupunkt, wird, wie in Abbildung 5.14 (rechter

[10] Bei diesem System handelt es sich nicht um einen Autopiloten. Der Fahrer muss die Hände am Lenkrad haben. Er wird lediglich beim Lenken unterstützt.

Teil) dargestellt, als Kreisbogen angenommen. Es ergibt sich:

$$\Delta y_{\kappa_{AP}} = -\text{sgn}\left(\kappa_{AP}\right) \cdot \left(\frac{1}{|\kappa_{AP}|} - \sqrt{\frac{1}{\kappa_{AP}^2} - (T \cdot v)^2} \right)$$

Das Softwaremodul, das dieses System simuliert, muss die Sensorik des realen Systems nachbilden. Es bezieht die Werte Δy, $\Delta\psi$ und κ_{AP} über den primären Cursor pc und einen sekundären Cursor sc:

Δy : Bei gegebener Fahrzeugposition (x, y) gilt:

$$|\Delta y| = ||(x, y) - pc.l.curve.p(pc.s)||$$

Das Vorzeichen von Δy kann durch einen Vergleich der Richtung des Vektors $(x, y) - pc.l.curve.p(s)$ mit der des Normalenvektors in $pc.l.curve.p(s)$ ermittelt werden.

$\Delta\psi$: Wird mithilfe von pc berechnet. Dies ist im Beispiel auf Seite 178 erläutert.

κ_{AP} : In jedem Simulationsschritt wird der sekundäre Cursor mittels

$$sc.Setup(pc)$$

auf die Stelle des primären Cursors gesetzt. Von dort aus wird er durch

$$sc.Move(T \cdot v, straight)$$

zum Vorausschaupunkt bewegt[11]. Danach kann die Krümmung abgefragt werden:

$$\kappa_{AP} = sc.l.curve.\kappa(sc.s)$$

In der Realität wird das System abgeschaltet, sobald der Sensor aufgrund von Problemen in der Bildverarbeitung die benötigten Werte nicht genau genug bestimmen kann. Dies ist z.B. in Verkehrsknoten der Fall. In der Simulation wird deswegen das System deaktiviert, wenn $pc.l \triangleright$ ***AreaLane*** und/oder $sc.l \triangleright$ ***AreaLane*** gilt.

Am Beispiel der **physikalischen Fahrzeugsimulation** auf Seite 178 wurde gezeigt, wie aus dem primären Cursor pc die Steigung der Fahrbahn in Fahrzeuglängs- und Querrichtung gewonnen wird. Detailliertere Modelle von Fahrzeugen erlauben als Eingabe zusätzlich die Höhe, auf der jedes der vier

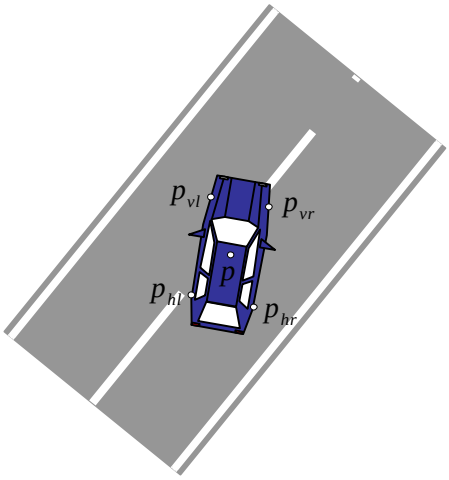

Abbildung 5.15.: Der Fahrzeugschwerpunkt befindet sich an der Position p, die vier Räder an den Positionen $p_{vl}, p_{vr}, p_{hl}, p_{hr}$.

Räder die Fahrbahnoberfläche berührt (vgl. z.B. [35]). Abbildung 5.15 zeigt die Situation.
In der xy-Ebene befindet sich der Schwerpunkt des Fahrzeugs an der Position

$$p = pc.l.curve.p(pc.s)$$

Mit d_v wird der Abstand der Vorderachse zum Schwerpunkt bezeichnet, mit ψ der Winkel, den das Fahrzeug um die z-Achse des ortsfesten Koordinatensystems gedreht ist. Die Mitte der Vorderachse hat die Position

$$p_v = p + d_v \cdot \begin{pmatrix} \cos\psi \\ \sin\psi \end{pmatrix}$$

Analog gilt für die Mitte der Hinterachse

$$p_h = p - d_h \cdot \begin{pmatrix} \cos\psi \\ \sin\psi \end{pmatrix}$$

Über die Normale

$$n = \begin{pmatrix} -\sin\psi \\ \cos\psi \end{pmatrix}$$

[11] Es wird davon ausgegangen, dass sich das Fahrzeug in Richtung der aktuellen Spur bewegt, dass also $\operatorname{sgn}(\cos\Delta\psi) > 0$ ist.

zur Fahrzeuglängsachse sowie die Fahrzeugbreite b lässt sich jetzt die Positon p_{vl} des linken Vorderrads, p_{vr} des rechten Vorderrads usw. berechnen:

$$\begin{aligned} p_{vl} &= p_v - \frac{b}{2} \cdot n \\ p_{vr} &= p_v + \frac{b}{2} \cdot n \\ p_{hl} &= p_h - \frac{b}{2} \cdot n \\ p_{hr} &= p_h + \frac{b}{2} \cdot n \end{aligned}$$

Mit der Methode $sc.Setup(x, y)$ eines sekundären Cursors sc wird zu jedem dieser Punkte die näheste Spur und der näheste Punkt darauf gesucht. Das entsprechende Rad liegt dann auf der Spurhöhe in diesem Punkt. Schreibt man beispielsweise

$$p_{vl} = \begin{pmatrix} x_{vl} \\ y_{vl} \end{pmatrix}$$

dann ergibt sich nach dem Aufruf von $sc.Setup(x_{vl}, y_{vl})$ für die Höhe h_{vl} in diesem Punkt:

$$h_{vl} = sc.l.h(sc.s)$$

Sekundäre Cursor werden vom Softwaremodul **Verkehrssimulation** zur Simulation anderer Verkehrsteilnehmer verwendet. Jeder simulierte Verkehrsteilnehmer ist im Straßennetzwerk durch einen sekundären Cursor sc repräsentiert. Über die Methode $sc.Setup(l_{setup}, s_{setup})$ kann er an einer bestimmten Stelle im Sichtbarkeitsbereich aufgesetzt werden. Die Bewegung entlang von Spuren erfolgt über die Methode $sc.Move(\Delta s, heading)$. Die Parameter Δs und $heading$ werden dabei von verschiedenen Verhaltensmodellen berechnet (vgl. [21]). Möchte der Verkehrsteilnehmer beispielsweise am nächsten Verkehrsknoten rechts abbiegen, dann wird $heading = right$ gesetzt. Sobald die aktuelle Spur diese Richtung tatsächlich hat, also

$$right \in sc.l.LaneInfo.heading$$

beginnt der Abbiegevorgang. Aufwändigere Modelle beobachten zur Verhaltensplanung die Fahrbahn, die dem simulierten Fahrzeug ein Stück vorausliegt, z.B. um beim Abbiegen die Geschwindigkeit rechtzeitig zu reduzieren und zu blinken. Solche Vorausschaupunkte können, wie bereits im Beispiel der Querführungsassistenz auf Seite 183 gezeigt, ebenfalls durch sekundäre Cursor realisiert werden.

Durch Strecken-Events, die in $sc.Events$ nach jeder Bewegung aufgesammelt werden, kann der Anwender die Verkehrsteilnehmer spezielle Aktionen ausführen lassen. Damit hat er die Möglichkeit, gezielt das Verhalten einzugreifen. So wird die Generierung von reproduzierbaren Verkehrssituationen unterstützt.

6. Ergebnisse und nächste Schritte

Im Gegensatz zu Fahrten im realen Verkehr bietet die Fahrsimulation den entscheidenden Vorteil, dass sich die Situationen, in die Fahrer gebracht werden, weitgehend kontrollieren lassen. Um daraus Erkenntnisse für die Realität abzuleiten, muss das Verkehrsumfeld in der Simulation möglichst detailgetreu nachgebildet werden. Grundlage hierfür ist ein Modell des Straßennetzwerks, in dem sich die zu untersuchenden Verkehrssituationen abspielen. Üblicherweise wird in Fahrsimulatoren ein real existierendes oder ein fiktives Straßennetzwerk nachgebildet. Diese Repräsentation ist global geometrisch konsistent, d.h. das Straßennetzwerk kann als Ganzes in einer maßstabsgetreuen Karte wiedergegeben werden. Diese Vorgehensweise hat drei Nachteile:

- Die Länge des befahrbaren Straßennetzes ist aufgrund der Datenmengen, die zu seiner geometrischen Repräsentation benötigt werden, beschränkt. Dies erschwert die Durchführung von Langzeitversuchen.
- Die Abfolge von Situationen ist schwer planbar, da sie im Wesentlichen von der Route abhängt, die der Fahrer durch das Straßennetzwerk nimmt.
- Es ist kaum möglich, das Auftreten von Situationen und damit von Streckenstücken, in denen sich diese Situationen abspielen, an bestimmte Bedingungen (z.B. solche, die den Zustand des Fahrers betreffen) zu knüpfen.

In der vorliegenden Arbeit wurde eine Repräsentation des Straßennetzwerks erarbeitet, die die genannten Nachteile nicht hat. Die Grundidee ist dabei, dass der Fahrer - bedingt durch die beschränkte Auflösung des Sichtsystems des Fahrsimulators - zu keinem Zeitpunkt der Simulation das gesamte Straßennetzwerk überblicken kann. Folglich kann die Abfolge von Streckenstücken außerhalb des Sichtbarkeitsbereichs verändert werden, ohne dass dies vom Fahrer bemerkt wird. Das Straßennetzwerk ist also lokal geometrisch konsistent, eine maßstabsgetreue Karte kann nur von dem Ausschnitt des Netzwerks

erstellt werden, der sich in einem bestimmten Simulationsschritt im Sichtbarkeitsbereich des Fahrers befindet. Das Straßennetzwerk als Ganzes ist auf einer topologischen Ebene modelliert.

Die durch dieses Konzept erzielten Ergebnisse werden anhand der Aufgabenstellung in Kapitel 2 verdeutlicht.

Generierung von reproduzierbaren Situationen
Damit bei jeder Fahrt Ereignisse in bestimmten Streckenabschnitten an genau spezifizierten Orten eintreten, wurden sog. Strecken-Events eingeführt. Ein Strecken-Event ist eine unsichtbare Markierung auf einer Spur eines Streckenstücks. Wird es überfahren, und zwar - je nach Typ des Events - vom Fahrer selbst oder von einem anderen Objekt, das sich im Straßennetzwerk bewegt, dann wird ein bestimmtes Ereignis ausgelöst.
Der zweite Mechanismus zur Generierung von reproduzierbaren Situationen sind sog. bedingte Streckenverknüpfungen. Die Abfolge von Streckenstücken wird dabei von Bedingungen abhängig gemacht, die an den Zustand des Fahrers oder den Zustand von Softwaremodulen der Simulation gestellt werden. Jedesmal, wenn während der Simulation eine solche Verknüpfung in den Sichtbarkeitsbereich des Fahrers gelangt, werden die Bedingungen ausgewertet und das nachfolgende Streckenstück selektiert.

Kontrolle des Versuchsablaufs
Die Abfolge von Situationen und damit die Abfolge bestimmter Streckenstücke darf durch die Wahlmöglichkeiten, die ein Fahrer beispielsweise an einer Kreuzung hat, nicht gestört werden. Dies wird durch die Repräsentation des Straßennetzwerks auf einer topologischen Ebene und der Veränderbarkeit der Streckenfolge während der Simulation gewährleistet. Biegt der Fahrer an einer Kreuzung falsch ab, so kann er trotzdem zur geplanten Folgesituation geführt werden, indem alle Ausgänge der Kreuzung mit demselben Streckenstück verknüpft werden. Die Auflösung solcher u.U. geometrisch inkonsistenten Verknüpfungen erfolgt während der Simulation durch die sog. geometrische Instanzierung (vgl. Abschnitt 5.1).

Einfacher Entwurf von Straßennetzwerken
Damit Versuchsleiter auf die jeweilige Fragestellung zugeschnittene Straßennetzwerke und Situationen selbst definieren könnnen, wurde eine Skriptsprache entwickelt (vgl. Kapitel 3). Die Planung und Organisation von Versuchen sowie deren spätere Auswertung und die Wiederverwendbarkeit von einzelnen Situationen wird von dieser Skriptsprache durch entsprechende syntaktische

Konstrukte erleichtert. Ferner wird der Versuchsleiter dabei unterstützt, die Streckennetze gemäß den in Deutschland gültigen Richtlinien zum Straßenbau (vgl. [36]) zu spezifizieren.

Bereitstellung von Informationen

Die meisten Softwaremodule einer Fahrsimulation (z.B. die Simulation anderer Verkehrsteilnehmer oder Sensoren von Assistenzsystemen) benötigen Daten über das Straßennetzwerk. Hierzu wurden Schnittstellen entworfen, über die Informationen zur geometrischen Instanzierung (vgl. Abschnitt 5.2.2) und zu geometrischen Daten der Fahrspuren im Sichtbarkeitsbereich des Fahrers (vgl. Abschnitt 5.2.3) effizient erfragt werden können.

Erweiterbarkeit

Die Repräsentation des Straßennetzwerks wurde auf fünf Ebenen aufgeteilt und nach Methoden der objektorientierten Modellierung entworfen. Dadurch lassen sich neue Modellierungsaspekte leicht und konsistent einfügen.

Abbildung 6.1.: Die Landschaft auf der linken Seite der Autobahn hat den Typ "bewirtschaftet", die auf der rechten Seite den Typ "bewaldet". Auf der Standspur fährt ein LKW der Verkehrssimulation.

Beim Einsatz in Fahrversuchen am IZVW hat sich das vorgestellte Konzept bewährt. Jedoch ist in der Anwendung bereits Bedarf nach einigen Erweiterungen entstanden:

Landschaft Zwei bereits implementierte Erweiterungen sind in [29] dargestellt. Zum einen werden dort Strecken um Landschaftsoberflächen mit natürlich aussehenden Höhenprofilen ergänzt. Zum anderen werden die Landschaften links und rechts von Straßen automatisch generiert. Der Anwender wählt hierzu in der Skriptsprache für jede Seite einer Straße einen Landschaftstypen (z.B. "bewirtschaftet", "bewaldet" oder "bewohnt") und beeinflusst dessen grobes Aussehen über einige wenige Paramter. Jedesmal, wenn während der Simulation eine Straße neu in den Sichtbarkeitsbereich des Fahrers gelangt, wird aus diesen Parametern ein Höhenprofil generiert und die für den gewählten Landschaftstyp charakteristischen grafischen Objekte verteilt (vgl. Abbildung 6.1).

Kollisionserkennung Den Softwaremodulen der Simulation sollen Informationen über Kollisionen zwischen bewegten Objekten untereinander sowie zwischen bewegten Objekten und statischen Strukturen (Pfosten, Leitplanken, Bäume etc.) zur Verfügung gestellt werden.

Verkehrsregeln Die für die Verkehrssimulation relevanten Verkehrsregeln sollen klassifiziert und derart repräsentiert werden, dass jeder Verkehrsteilnehmer zu jedem Zeitpunkt der Simulation die für ihn gerade gültigen abfragen kann.

Verkehrsknoten Der Entwurf von Verkehrsknoten soll vereinfacht werden. Dem Anwender muss ermöglicht werden, einen generischen Verkehrsknoten (z.B. T-Kreuzung, X-Kreuzung, Übergang zwischen unterschiedlichen Querschnittsprofilen) zu wählen und anhand einiger weniger Parameter zu spezifizieren. Hieraus wird ein konkreter Verkehrsknoten automatisch erzeugt.

Das vorgestellte Modell bildet die Grundlage für solche Erweiterungen und soll schrittweise um zusätzliche Komponenten ergänzt werden.

Zusammenfassung

Die zentrale Idee in der vorliegenden Arbeit ist, dass das Verkehrsumfeld einer Fahrsimulation nur aus der Sicht des Fahrers realistisch sein muss. Hierfür ist in jedem Simulationsschritt ein beschränkter Ausschnitt des Straßennetzwerks, nämlich der im Wahrnehmungsbereich des Fahrers, geometrisch konsistent darzustellen. Aus der Sicht eines Versuchsleiters ist das Straßennetzwerk

topologisch repräsentiert. Es kann außerhalb des Wahrnehmungsbereichs des Fahrers während der Simulation gezielt verändert werden. Die daraus resultierenden Möglichkeiten, Verkehrssituationen zu kontrollieren, erweitern den Einsatzbereich von Fahrsimulation als Methode der verkehrswissenschaftlichen Forschung.

Literaturverzeichnis

[1] R.W. Allen, T.J. Rosenthal, et al.: A Low Cost PC Based Driving Simulator for Prototyping and Hardware-In-The-Loop Applications, SAE Technical Paper Series 980222, Society of Automotive Engineers, 1998

[2] R.W. Allen, T.J. Rosenthal, et al.: A Scenario Definition Language for Developing Driving Simulator Courses, Proc. of the Driving simulation Conference, Sophia Antipolis, France, 2001

[3] B.E. Artz: An analytical road segment terrain database for driving simulation, Proc. of the Driving Simulation Conference, Sophia Antipolis, France, 1995

[4] A.C. Bailey: Advancement in logical road network design for the Leeds Driving Simulator, Proc. of the Driving simulation Conference, Paris, France, 2000

[5] S. Bayarri, M. Fernandez, M. Perez: Virtual reality for driving simulation, Communications of the ACM, 39(5):72-76, 1996

[6] G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modelling Language User Guide, Addison Wesly Longman, Boston, 1999

[7] T. Breiner: Freie Konzeption zukünftiger Fahrsimulatoren, Tagungsband 7. Workshop Sichtsysteme - Visualisierung in der Simulationstechnik, S. 15 ff., Wuppertal, 2001

[8] I.N. Bronstein, K.A. Semendjajew: Taschenbuch der Mathematik, B.G. Teubner Verlagsgesellschaft, Stuttgart, Leipzig, 1991

[9] S. Buld, H.-P. Krüger, S. Hoffmann, A. Kaussner, H. Tietze, I. Totzke: Wirkungen von Assistenz und Automation auf Fahrerzustand und Fahrsicherheit, veröffentlichter Abschlussbericht Projekt EMPHASIS (BMBF Förderkennzeichen 19 S 9812 7), Würzburg, 2002

[10] O. Carles, S. Espié: Database generation system for road application, Proc. of the Driving Simulation Conference, Paris, France 1999

[11] O. Carles, S. Espié: Multi-level environments modelling for road simulations, Proc. of the Driving Simulation Conference, Paris, France, 2000

[12] M. Desrochers: Database Structures, in: Tagungsband 6. Workshop Sichtsysteme - Visualisierung in der Simulationstechnik, Shaker Verlag, Aachen, 1999

[13] S. Donikian: VUEMS: A virtual urban environment modeling system, in: Computer Graphics International '97, 1997

[14] G. Farin: Kurven und Flächen im Computer Aided Geometric Desing, Vieweg und Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994

[15] J. Feng, Q. Peng: Speed control in path motion, Proc. of the 4th International Conference on Computer-Aided Design and Computer Graphics, Wuhan, China, 1996

[16] E. Frieling, K. Sonntag: Lehrbuch Arbeitspsychologie, 2. Auflage, S. 139 ff., Verlag Hans Huber, Göttingen, 1999

[17] T.D. Gillespie: Fundamentals of Vehicle Dynamics, Society of Automotive Engineers, Warrendale, 1992

[18] A. Göllö, A. Deshpande, P. Hingorani, P. Varaiya: Smartdb: An object oriented simulation framework for intelligent vehicles and highway systems, in: Fifth Annual Conference on AI, Simulation and Planning in High Autonomy Systems, pp. 244-250, IEEE Computer Society Press, 1994

[19] H. Grezlikowski, V. Schill: With Low Cost to High End - a PC approach at the DaimlerChrysler Driving Simulator, Proc. of the Driving Simulation Conference, Paris, France, 2000

[20] P. Grant, B. Artz, L. Cathey, J. Greenberg: Hardware in the Loop Torque on Demand (TOD), Proc. of the Driving Simulation Conference, Paris, France, 2000

[21] M. Grein, H-P. Krüger, H. Noltemeier: A framework for ambient Traffic in the IZVW driving simulator, Proc. of the Driving Simulation Conference, Paris, France, 2002

[22] M. Grein, A. Kaussner, H.-P. Krüger, H. Noltemeier: A flexible application framework for distributed real time systems with applications in PC based driving simulators, Proc. of the Driving Simulation Conference, Sophia Antipolis, France, 2001

[23] V. Hargutt: Eyelid Movements and their Predictive Value for Fatigue Stages, 3rd International Conference of Psychophysiology in Ergonomics, San Diego, 2000

[24] A. Kaussner, C. Mark, H.-P. Krueger, H. Noltemeier: Generic creation of landscapes and modelling of complex parts of road networks, Proc. of the Driving Simulation Conference, Paris, France, 2002

[25] M.G. Lenne, P. Dietze, G. Rumbold, J.R. Redman, T.J. Triggs: Treatments for Opiate Dependence and Driving Ability, Proc. of the Road Safety Rese-

arch, Policing, and Education Conference, Wellington, New Zealand, 1998, Volume I, pp. 36-40

[26] J.P. Löwenau, J.H. Bernasch, H.G. Rieker, P.J.Th. Venhovens, J.P. Huber, W. Huhn: Adaptive Light Control - a New Light Concept Controlled by Vehicle Dynamics and Navigation, SAE Technical Paper Series 980007, Society of Automotive Engineers, 1998

[27] J.P. Löwenau, M.H. Strobl, J.H. Bernasch, F.M. Reich, A.H. Rummel: Evaluation of Adaptive Light Control in the BMW Driving Simulator, Proc. of the Driving Simulation Conference, Sophia Antipolis, France, 2001

[28] T.D. Marcotte, R.K. Heaton, et al.: The impact of HIV-related neuropsychological dysfunction on driving behavior, Journal of the International Neuropsychological Society 5(7): 579-592, 1999

[29] C. Mark: Online Generierung und Visualisierung von dynamischen Szenerien in der Fahrsimulation, Diplomarbeit, Universität Würzburg, 2002

[30] K. Naab: Heading Control - Ein System zur Fahrerunterstützung bei der Spurhaltung, Veranstaltungsunterlagen Fahrerassistenzsysteme, Haus der Technik e.V., Essen, 1998

[31] M.A. Nahon, L.D. Reid: Simulator motion-drive algorithms : a designer's perspective, Journal of Guidance 3(2), pp. 356-362, 1989

[32] H. Natzschka: Straßenbau - Entwurf und Bautechnik, B.G. Teubner, Stuttgart, 1997

[33] S. Nordmark, H. Jansson, G. Palmkvist: ABS-braking in an emergency situation - a simulator study utilising hardware-in-the-loop, Proc. of the Driving Simulation Conference, Paris, France, 1999

[34] Y. E. Papelis, S. Bahauddin: Logical modeling of roadway environment to support real-time simulation of autonomous traffic, in: SIVE95: The First Workshop on Simulation and Interaction in Virtual Environments, pp. 62-71, Department of Computer Science, University of Iowa, 1995

[35] K. Popp, W. Schiehlen: Fahrzeugdynamik, Eine Einführung in die Dynamik des Systems Fahrzeug - Fahrweg, B.G. Teubner, Stuttgart, 1993

[36] Richtlinien für die Anlage von Straßen, Teil: L (Linienführung), Forschungsgesellschaft für Straßen- und Verkehrswesen, Bonn - Bad Godesberg, 1995

[37] Richtlinien für die Anlage von Straßen, Teil: Q (Querschnitte), Forschungsgesellschaft für Straßen- und Verkehrswesen, Köln, 1996

[38] G. Rill: Simulation von Kraftfahrzeugen, Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994

[39] P. Schulze: Die constrained Delauny Triangulierung als Basis der Generierung hochdetaillierter urbanisierter Geländedatenbasen für Simulationszwecke, in: Tagungsband 7. Workshop Sichtsysteme - Visualisierung in der Simulationstechnik, Shaker Verlag, Aachen, 2001

[40] D. Stewart: A platform with six degrees of freedom, Proceedings of the Institution of Mechanical Engineers, v. 180, Part 1, No. 15, pp. 371-368, 1965/66

[41] R. Sukthankar: Situation Awareness for Tactical Driving, PhD thesis, The Robotics Institute, Carnegie Mellon University, 1997

[42] H.T. Uitermark, A.B.M. Vogels: Semantic and geometric aspects of integrating road networks, Proc. of the 2nd International Conference on Interoperating Geographic Information Systems, Zürich, 1999

[43] S. Volz, M. Grossmann, N. Hönle, D. Nicklas, T. Schwarz: Integration mehrfach repräsentierter Straßenverkehrsdaten für eine föderierte Navigation, it + ti Informationstechnik und Technische Informatik 44 (2000) 5, Oldenbourg Verlag, München, 2000

[44] V. Wünsch: Differentialgeometrie. Kurven und Flächen, B.G. Teubner, Stuttgart, 1997

A. Ergänzungen zur formalen Beschreibung

A.1. Beschreibung einer T-Kreuzung

In der **Navigationsaufgabe**[1] wird eine T-Kreuzung eingesetzt. An diesem Beispiel soll die in Abschnitt 3.4.2 (S. 50 ff.) vorgestellte formale Beschreibung von Verkehrsknoten verdeutlicht werden. Die T-Kreuzung, in der der Fahrer trotz uneindeutiger Navigationshinweise links abbiegen muss, ist in Abbildung 3.6 (S. 52) und - etwas schematischer - in Abbildung A.1 zu sehen.
Zu beachten ist, dass die Abbildung nicht maßstabsgetreu ist, was bedeutet, daß die Größenangabe für das Rechteck sowie die Koordinaten für die Beginn-Punkte von Spuren nicht aus der Abbildung entnommen sind. In diesem Beispiel wird - der Übersichtlichkeit halber - nur die Beschreibung der schwarz eingezeichneten Spuren illustriert.

Die Kennung der Schablone[2], die Größe des Rechtecks und die drei Ports werden wie folgt definiert (Größenangabe und Positionen der Ports in Meter):

```
id = 1234;
Größe = (107, 104);

Port1.Seite = links;
Port1.Querschnittsprofil = Bundesstraße;
Port1.Position = -2.5;

Port2.Seite = rechts;
```

[1] Die Navigationsaufgabe ist auf Seite 19 erklärt. Das Streckennetz dieser Situation ist auf Seite 38 beschrieben.
[2] Für die Zwecke dieses Beispiels ist die Kennung nicht von Bedeutung.

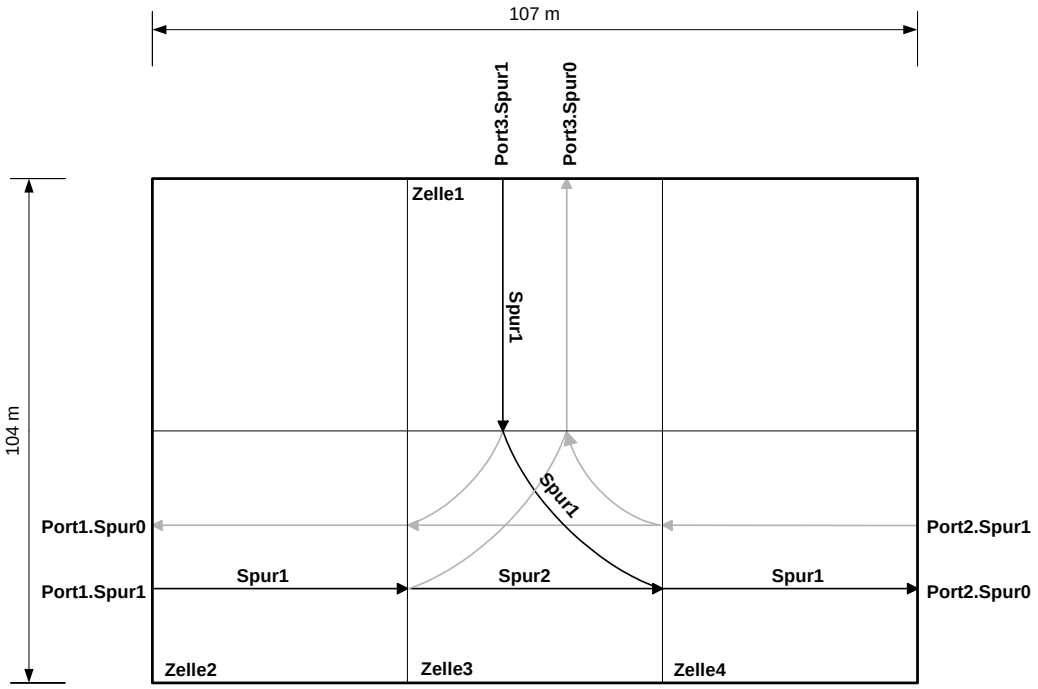

Abbildung A.1.: Schematische Darstellung einer T-Kreuzung (nicht maßstabsgetreu)

```
Port2.Querschnittprofil = Bundesstraße;
Port2.Position = -2.5;

Port3.Seite = oben;
Port3.Querschnittprofil = Bundesstraße;
Port3.Position = 0;
```

Der Querschnitt Bundesstraße beschreibt eine Straße mit je einer Fahrspur pro Richtung. Jede Fahrspur hat eine Breite von 3.5 m. Die Kreuzung ist in vier Spurzellen (Zelle1 bis Zelle4) aufgeteilt. Zelle1 führt von Port3 zum Innenbereich der Kreuzung. Die schwarz eingezeichnete Spur Zelle1.Spur1 ist eine Gerade:

```
SpurZelle Zelle1
{
    Gerade Spur1
    {
        Beginn = Port3.Spur1;
        Richtung = geradeaus;
    };
    ...
};
```

Da Zelle1.Spur1 an Port3.Spur1 beginnt, entfällt die Angabe von BreiteBeginn und HöheBeginn, die entsprechenden Werte sind durch den Querschnitt des Ports festgelegt.

Im Innenbereich Zelle3 werden exemplarisch zwei Spuren (Zelle3.Spur1 und Zelle3.Spur2) beschrieben (auch hier alle Angaben in Meter):

```
SpurZelle Zelle3
{
    Bogen Spur1
    {
        Beginn = (-1.75, 19.0);
        BreiteBeginn = 3.5;
        HöheBeginn = 0;
        Richtung = links;
    };
    Gerade Spur2
```

```
    {
        Beginn = (-22.5, -4.25);
        BreiteBeginn = 3.5;
        HöheBeginn = 0;
        Richtung = geradeaus;
    };
    ...
};
```

Keine Spur der Kreuzung ändert die Breite, deswegen wird in `Zelle3.Spur1` und in `Zelle3.Spur2` jeweils die Breite am Beginn auf 3.5 m gesetzt. Auch die Höhe aller Spuren bleibt auf 0.

Für `Zelle2` und `Zelle4` sieht die Beschreibung so aus:

```
SpurZelle Zelle2
{
    Gerade Spur1
    {
        Beginn = Port1.Spur1;
        Richtung = geradeaus;
    };
    ...
};

SpurZelle Zelle4
{
    Gerade Spur1
    {
        Beginn = (22.5, -4.25);
        BreiteBeginn = 3.5;
        HöheBeginn = 0;
        Richtung = geradeaus;
    };
    ...
};
```

Nach der Definition der Spurzellen und der darin enthaltenen Spuren werden die Spuren verknüpft. Die Reihenfolge sowie die Angabe, welche Spur in einer Verknüpfung links oder rechts aufgeführt ist, spielt dabei keine Rolle:

```
SpurVerknüpfungen =
{
    Zelle1.Spur1.Ende <-> Zelle3.Spur1.Beginn,
    Zelle3.Spur1.Ende <-> Zelle4.Spur1.Beginn,
    Zelle3.Spur2.Beginn <-> Zelle2.Spur1.Ende,
    Zelle3.Spur2.Ende <-> Zelle4.Spur1.Beginn,
    ...
};
```

Auf ähnliche Weise werden die in Abbildung A.1 grau eingezeichneten Spuren definiert und verknüpft.

A.2. Strecken-Events

Auf die folgenden Grammatiken wird vom Abschnitt 3.4.4 (61) aus verwiesen.

Grammatik A.1 Events in Schablonen von Straßen

```
<Events Schablone Straße> →
    Events =
    {
        (,)<Event Schablone Straße>
    };
<Event Schablone Straße> →
    ( <Pos. Straße>, <Event Argumente (→ A.2, S.208)> )
<Pos. Straße> →
    <Spurindex>,
    <Distanz Spurbeginn>
<Distanz Spurbeginn> →
    <Zahl >= 0> m |
    <Zahl aus [0, 100]> %
```

Grammatik A.2 Argumente von Strecken-Events

```
<Event Argumente> →
    <Name Software Modul>,
    (,)<Event Parameter>
<Event Parameter> →
    <Zahl> |
    <Bezeichner>
```

Grammatik A.3 Events in Schablonen von Verkehrsknoten

```
<Events Schablone Verkehrsknoten> →
    Events =
    {
        (,)<Event Schablone Verkehrsknoten>
    };
<Event Schablone Verkehrsknoten> →
    ( <Pos. Verkehrsknoten>, <Event Argumente (→ A.2, S.208)> )
<Pos. Verkehrsknoten> →
    <Name Spurzelle>.<Name Spur>,
    <Distanz Spurbeginn (→ A.1, S.207)>
```

Grammatik A.4 Events von Strecken-Instanzen

```
<Events von Strecken-Instanzen> →
    Events =
    {
        (,)<Event einer Strecken-Instanz>
    };
<Event einer Strecken-Instanz> →
    <Event Instanz Straße> |
    <Event Instanz Verkehrsknoten>
<Event Instanz Straße> →
    (
        <Name Strecken-Inst.>,
        <Pos. Straße (→ A.1, S.207)>,
        <Event Argumente (→ A.2, S.208)>
    )
<Event Instanz Verkehrsknoten> →
    (
        <Name Strecken-Inst.>,
        <Pos. Verkehrsknoten (→ A.3, S.208)>,
        <Event Argumente (→ A.2, S.208)>
    )
```

B. Ergänzungen zur Modellierung

B.1. Ebene Parameterkurven

Der folgende Überblick über die Differentialgeometrie ebener Parameterkurven ist im Wesentlichen [44] entnommen.

Definition B.1 (Ebene Parameterkurve)
Eine ebene Parameterkurve ist eine Vektorfunktion $p : I \to \mathbb{R}^2$ eines Intervalls $I \subset \mathbb{R}$ in den $\mathbb{R}^2$. •

Eine ebene Parameterkurve p ist durch zwei Koordinatenfunktionen, $x(t)$ und $y(t)$ charakterisiert, besitzt also die Darstellung

$$p(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

mit $t \in I$. Die Variable t heißt Parameter der Kurve. In dieser Arbeit wird angenommen, daß p (und damit x und y) auf I genügend oft differenzierbar ist, d.h. mindestens $\in C^2(I)$ liegt. Üblicherweise wird zwischen der Parameterkurve und ihrer Spur

$$p(I) = \{p(t) | t \in I\} \subset \mathbb{R}^2$$

unterschieden. Wo keine Missverständnisse auftreten, wird im Folgenden mit dem Begriff "Parameterkurve" auch die Spur bezeichnet.

Durch eine Parametrisierung wird die Orientierung einer Parameterkurve festgelegt: Die Spur der Parameterkurve wird in Richtung wachsender Werte des Parameters durchlaufen.

Für eine gegebene Spur $p(I)$ einer Kurve gibt es unterschiedliche Parametrisierungen. Beispielsweise haben die beiden Kurven

$$p_1(t) = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}$$
$$p_2(t) = \begin{pmatrix} \cos 2t \\ \sin 2t \end{pmatrix}$$

mit $t \in [0, 2\pi]$ dieselbe Spur, nämlich den Einheitskreis in der Ebene mit Mittelpunkt im Ursprung.

Definition B.2 (Umparametrisierung, Parametertransformation)
Eine Kurve $p^\star : I^\star \to \mathbb{R}^2$ heißt Umparametrisierung einer Kurve $p : I \to \mathbb{R}^2$, wenn es eine differenzierbare Funktion $t : I^\star \to I$ gibt mit

$$t^\star \mapsto t = t(t^\star) \in I, t^\star \in I^\star$$

so dass für alle $t^\star \in I^\star$ gilt

$$\frac{dt}{dt^\star}(t^\star) \neq 0, p^\star(t^\star) = p(t(t^\star))$$

Eine Umparametrisierung heißt orientierungstreu bzw. orientierungsumkehrend, falls $\frac{dt}{dt^\star}(t^\star) > 0$ bzw. $\frac{dt}{dt^\star}(t^\star) < 0$ ist. •

Die Länge einer Kurve kann gemessen werden:

Definition B.3 (Bogenlänge)
Es sei $p : I \to \mathbb{R}^2$ eine Parameterkurve. Für $t_0 \in I$ ist die Bogenlänge des Kurvenstückes $p : [t_0, t] \to \mathbb{R}^2$ gegeben durch:

$$L_{t_0}(t) = \int_{t_0}^{t} ||p'(u)||du$$

Für $I = [a, b]$ ist $L_a(t)$ die Bogenlänge bis zum Parameterwert t und $L(p) := L_a(b)$ die Länge der Parameterkurve. •

In dieser Arbeit wird häufig eine spezielle Parametrisierung verwendet:

Definition B.4 (Parametrisierung nach der Bogenlänge)
Eine Parametrisierung $p(s)$ einer Kurve p heißt Parametrisierung nach der Bogenlänge (oder auch "natürliche Parametrisierung"), wenn für alle $s \in [0, b]$ gilt:

$$\int_0^s ||p'(u)||du = s$$

•

Bei $p(s)$ handelt es sich genau dann um eine Parametrisierung nach der Bogenlänge, wenn

$$||p'(s)|| = 1$$

für alle $s \in [0, b]$ gilt. Zu jeder Kurve gibt es eine orientierungstreue Umparametrisierung nach der Bogenlänge. In den meisten Fällen kann diese jedoch leider nicht explizit angegeben werden.

Definition B.5 (Krümmung, Krümmungskreis, Krümmungsmittelpunkt)
Es sei p eine ebene Parameterkurve und

$$p(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

eine Parametrisierung. Dann heißt die Zahl

$$\kappa(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{(x'^2(t) + y'^2(t))^{3/2}}$$

Krümmung von p in t. Wenn $\kappa(t) \neq 0$ ist, dann gibt es genau einen in der Ebene liegenden Kreis $K(t)$ mit dem Mittelpunkt

$$p_m(t) = p(t) + \frac{1}{\kappa(t)} n(t)$$

Dabei ist $n(t)$ die Normale zum Punkt $p(t)$. $K(t)$ heißt Krümmungskreis, $\frac{1}{\kappa(t)}$ Krümmungsradius und $p_m(t)$ Krümmungsmittelpunkt von p in t. •

B.2. Transformationen ebener Parameterkurven

Die Parameterkurven, die den Verlauf der Fahrbahnen einzelner Abschnitte des Straßennetzwerks beschreiben, werden durch eine Drehung und Verschiebung so transformiert, dass sich ein "glatter" Verlauf der gesamten Strecke ergibt. Abbildungen, die Punkte in der Ebene um einen bestimmten Winkel drehen, gehören einer speziellen Klasse von Matrizen an:

Definition B.6 (orthogonale Matrix)
Eine Matrix $A \in \mathbb{R}^{n \times n}$ heisst orthogonal, wenn gilt:

$$AA^T = A^T A = I$$ •

Satz B.1 Eine Matrix $R \in \mathbb{R}^{2\times 2}$, die eine Drehung um den Winkel φ bzgl. der Basis $\{e_1, e_2\}$ des $\mathbb{R}^2$ darstellt, ist orthogonal.

Beweis: Setzt man zur Abkürzung $s := \sin\varphi$ und $c := \cos\varphi$, dann besitzt die Matrix R folgende Darstellung:

$$R = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

Damit gilt:

$$RR^T = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} = \begin{pmatrix} c^2+s^2 & cs-sc \\ sc-cs & s^2+c^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

Durch analoge Rechnung erhält man $R^T R = I$. □

Der folgende Satz zeigt, daß Transformationen, die durch orthogonale Matrizen beschrieben werden, nicht "verzerren":

Satz B.2 Sei $A \in \mathbb{R}^{n\times n}$ eine orthogonale Matrix. Dann gilt:

$$\|Av\| = \|v\| \qquad \text{für alle} \quad v \in \mathbb{R}^n$$

(Durch orthogonale Matrizen bestimmte lineare Abbildungen sind längentreu.)

Beweis:

$$\begin{aligned} \|Av\| &= \sqrt{< Av, Av >} \\ &= \sqrt{(Av)^T(Av)} = \sqrt{v^T A^T A v} \\ &= \sqrt{v^T v} = \sqrt{< v, v >} \\ &= \|v\| \end{aligned}$$

□

Die Längentreue von Abbildungen durch orthogonale Matrizen erhält die Parametrisierung nach der Bogenlänge einer durch sie transformierten Kurve:

Satz B.3 Sei $p : [0, S] \to \mathbb{R}^2$ eine nach der Bogenlänge parametrisierte Kurve, $R \in \mathbb{R}^{2\times 2}$ eine orthogonale Matrix und $a \in \mathbb{R}^2$. Dann ist auch $q : [0, S] \to \mathbb{R}^2$ mit

$$q(s) = Rp(s) + a$$

eine nach der Bogenlänge parametrisierte Kurve.

Beweis: Die Behauptung, dass q ebenfalls nach der Bogenlänge parametrisiert ist, ist äquivalent zu $\|q'(s)\| = 1$.
Es gilt:

$$\begin{aligned} \|q'(s)\| &= \|\frac{d}{ds}(Rp(s) + a)\| \\ &= \|Rp'(s)\| \end{aligned}$$

Mit Satz B.2 folgt hieraus:

$$\|q'(s)\| = \|p'(s)\|$$

Gemäß Voraussetzung ist $p(s)$ nach der Bogenlänge parametrisiert und damit $\|p'(s)\| = 1$. Insgesamt gilt also:

$$\|q'(s)\| = 1$$

□

Die Krümmung einer Parameterkurve ist gegenüber Rotation und Translation invariant:

Satz B.4 Sei $p : \mathbb{R} \to \mathbb{R}^2$ eine Parameterkurve und $\kappa_p : \mathbb{R} \to \mathbb{R}$ die Krümmung von p. Mit der Rotationsmatrix $R \in \mathbb{R}^{2x2}$, die eine Drehung um den Winkel φ bzgl. der Basis $\{e_1, e_2\}$ des $\mathbb{R}^2$ darstellt, und $a \in \mathbb{R}^2$ wird die Parameterkurve

$$q = Rp + a$$

definiert. Wenn κ_q die Krümmung von q ist, dann gilt:

$$\kappa_p = \kappa_q$$

Beweis: Schreibt man (zur Abkürzung ohne Parameter)

$$p = \begin{pmatrix} x \\ y \end{pmatrix}$$

dann gilt gem. Definition B.5:

$$\kappa_p = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}}$$

Mit $s := \sin\varphi$ sowie $c := \cos\varphi$ hat die Rotationsmatrix folgende Form:

$$R = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

Für q ergibt sich damit:

$$q = \begin{pmatrix} cx - sy + a \\ sx + cy + a \end{pmatrix}$$

Die Krümmung κ_q von q wird gem. Definition B.5 berechnet:

$$\begin{aligned}
\kappa_q &= \frac{(cx - sy + a)'(sx + cy + a)'' - (sx + cy + a)'(cx - sy + a)''}{[(cx - sy + a)'^2 + (sx + cy + a)'^2]^{3/2}} \\
&= \frac{(cx' - sy')(sx'' + cy'') - (sx' + cy')(cx'' - sy'')}{[(cx' - sy')(cx' - sy') + (sx' + cy')(sx' + cy')]^{3/2}} \\
&= \frac{(c^2 + s^2)x'y'' - (s^2 + c^2)y'x''}{[(c^2 + s^2)x'^2 + (s^2 + c^2)y'^2]^{3/2}} \\
&= \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}} \\
&= \kappa_p
\end{aligned}$$

□

B.3. Straßen

Auf den folgenden Algorithmus, der die gerichteten Abstände $cl_i.curve.d_r$ von Spuren $cl_i \triangleright$ ***CourseLane*** aus den Spurinformationen $cl_i.LaneInfo$ berechnet, wird von Abschnitt 4.4.2 aus verwiesen. Er bekommt als Eingabe eine Spurzelle $clc \triangleright$ ***CourseLaneCell*** und macht von der Annahme Gebrauch, dass sich die Spuren von Straßen nicht überlappen und nahtlos aneinander schließen.

Algorithmus B.1 Berechnung der Abstände $cl_i.curve.d_r$, $cl_i \in clc.ChildNodes$, von Spurmitten zur Mitte einer Straße

$w := 0$
for all $cl_i \in clc.ChildNodes$ **do**
  $w_i := w$
  $w := w + cl_i.LaneInfo.w_1$
end for
for all $cl_i \in clc.ChildNodes$ **do**
  $cl_i.curve.d_r := w_i + \frac{cl_i.LaneInfo.w_1 - w}{2}$
end for

B.4. Verkehrsknoten

Der folgende Algorithmus ergänzt die Ausführungen in Abschnitt 4.5.5 (S. 130 ff.). Mit l wird eine beliebige Spur eines Verkehrsknotens a bezeichnet, unabhängig davon, in welcher Spurzelle sie sich befindet.

Algorithmus B.2 Bestimmung der Parameterkurven von Spuren l eines Verkehrsknotens $a \triangleright$ ***Area***

// 1. Schritt: Bestimme Parameterkurven der Geraden
for all l mit $l.curve \triangleright$ ***AStraightLine*** **do**
  if $l.p_2$ liegt an einem Port $a.p_k$ **then**
    Berechne $l.curve.p_2$ aus Port-Querschnitt.
  else
    Bezeichnung: $l.p_2$ ist mit $l_j.p_1$ verbunden.
    $l.curve.p_2 := l_j.curve.p_1$
  end if
end for
// 2. Schritt: Bestimme Kontrollpunkte der Bezier-Kurven
for all l mit $l.curve \triangleright$ ***ABezierCurve*** **do**
  if $l.p_1$ liegt an einem Port $a.p_k$ **then**
    Berechne Koord. p_1 und Tangentenwinkel α_1 aus Port-Querschnitt.
  else
    Bezeichnung: $l.p_1$ ist mit $l_j.p_2$ verbunden.
    $p_1 := l_j.curve.p(l_j.curve.S)$
    $\alpha_1 := l_j.curve.\alpha(l_j.curve.S)$
  end if
  if $l.p_2$ liegt an einem Port $a.p_k$ **then**
    Berechne Koord. p_2 und Tangentenwinkel α_2 aus Port-Querschnitt.
  else
    Bezeichnung: $l.p_2$ ist mit $l_j.p_1$ verbunden.
    $p_2 := l_j.curve.p(0)$
    $\alpha_2 := l_j.curve.\alpha(0)$
  end if
  Berechne Kontrollpunkte gemäß Abschnitt 4.5.3.
end for

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Angestellter am Interdisziplinären Zentrum für Verkehrswissenschaften an der Universität Würzburg (IZVW).

Besonderer Dank gilt Herrn Prof. Dr. Hartmut Noltemeier, der die Arbeit wissenschaftlich betreute und durch konstruktive Kritik vorantrieb. Für die Übernahme des Korreferats und die eingehende Durchsicht der Arbeit danke ich Herrn Prof. Dr. Jürgen Wolff von Gudenberg.

Ich möchte auch meinen Kollegen danken, die mich während der Erstellung der Arbeit durch die angenehme Arbeitsatmosphäre und die wertvollen Diskussionen unterstützt haben.

Mein herzlicher Dank geht an den Leiter des IZVW, Herrn Prof. Dr. Hans-Peter Krüger, der die Arbeit ermöglichte.