Anhang C

Programmdokumentation

Im Zuge dieser Diplomarbeit wurde ein neues Programmsystem zur Bildfolgenauswertung mit der Kurzbezeichnung `Motris` (**MO**del based **TR**acking in **I**mage **S**equences) entworfen und zu großen Teilen gemeinsam mit H. Dahlkamp und A. Ottlik implementiert. `Motris` soll die verschiedenen, in der Forschungsgruppe Kognitive Systeme am IAKS entwickelten Verfahren in einem einheitlichen Rahmenwerk zusammenfassen und den Zugriff über eine gemeinsame Schnittstelle präsentieren. Aus verschiedenen Gründen wurde als Programmiersprache `Java` gewählt. Zum einen kommen verschiedene Rechnerarchitekturen zum Einsatz, so dass die Plattformunabhängigkeit von Java sehr nützlich ist. Zum anderen werden Programme im Universitätsrahmen von einer Vielzahl Studenten bearbeitet, die meisten davon nur über einen kurzen Zeitraum hinweg. In einem solchen Szenario tragen Fehler in der Speicherverwaltung in nicht mehr zu vernachlässigendem Umfang zu der Menge der potentiellen Fehler bei. Solche Speicherverwaltungsfehler können in Java in der Regel nicht auftreten.

Im folgenden werden nur die Entwurfsüberlegungen für die wesentlichen Programmteile erläutert. Eine vollständige Dokumentation würde aufgrund der reinen Anzahl der entstandenen Quelltextzeilen den zeitlichen Rahmen dieser Arbeit sprengen. Es wird daher auf 4 wesentliche Punkte eingegangen:

- Konfigurations- und Kontrollschnittstelle;
- Arbeitsdatenhandhabung;
- Modelle;
- Verfolgung.

C.1 Die Konfigurations- und Kontrollschnittstelle

Eine wichtige Überlegung bei dem Entwurf des Programmgerüsts war die Art und Weise, wie Benutzer die im Programm befindlichen Daten und Algorithmen manipu-

lieren können. Auf der einen Seite soll Motris vollständig interaktiv bedienbar sein, mit der Möglichkeit, die Belegung aller wesentlichen Datenelemente abzurufen und zu verändern, auf der anderen Seite sollen die Startbedingungen für komplexe Experimente und die Ergebnisse abspeicherbar und jederzeit wiederaufrufbar sein.

Zu diesem Zweck wurde das Paket `Parameter` geschrieben, das beide Arten von Zugriffen über eine einheitliche Schnittstelle ermöglicht. Kern des Pakets `Parameter` ist die gleichnamige Klasse `Parameter`. Diese enthält ein Datum sowie Metainformationen über das Datum wie den Typ des Datums, einen in einem Parametersatz eindeutigen Bezeichner, einen für Menschen sinnhaften Namen und eine Beschreibung. `Parameter` treten in der Regel in Parametersammlungen auf, realisiert durch die Klasse `ParameterSet`. Diese enthält eine beliebige Anzahl `Parameter` sowie eine beliebige Anzahl an Untermengen.

Ein `ParameterSet` kann auf zwei Arten manipuliert werden: zum einen kann es an einen `ParameterViewController` übergeben werden, der den Inhalt des `ParameterSet` in einem Bildschirmfenster darstellt. Dort können die Werte von vorhandenen `Parametern` eingesehen und verändert werden. Bei Änderungen gibt der `ParameterViewController` ein verändertes `ParameterSet` an den Datensatz zurück, vom dem das ursprünglich erhaltene `ParameterSet` stammte.
Ausserdem können `ParameterSets` über den `ParameterXMLHandler` als Datei im XML-Format eingelesen oder geschrieben werden. Der Schreibvorgang kann über das Fenster des `ParameterViewController` ausgelöst werden.

Alle Klassen, die bei der Verfolgung oder bei der Berechnung von Zwischenergebnissen oder Arbeitsdaten eine Rolle spielen, implementieren die Schnittstelle (in der Java-Bedeutung) `ParameterizedObject`, in der die Methoden `getParameters` und `setParameters` deklariert sind. Über diese werden die konfigurierbaren Werte in Form eines `ParameterSet` angefordert bzw. gesetzt.

`Parameter` können die folgenden Typen enthalten:

- `Integer`
- `Double`
- `Boolean`
- `Color`
- `String`
- `Fileselection`
- `Dirselection`
- `Matrix`
- `FreeParameter`

C.2 Arbeitsdatenhandhabung

Ein wichtiger Punkt bei dem Entwurf von Motris war eine möglichst starke Entkoppelung von Datenquellen und Datensenken. Dabei werden unter Datenquellen Methoden oder Klassen verstanden, die Daten bereitstellen (berechnen oder laden), die von Algorithmen oder dem Benutzer (Datensenken) benötigt werden. Ein zentrales Datenrepositorium wurde eingeführt, um die Kentnis der vorkommenden Datentypen und ihrer Quellen von den Senken zu isolieren. Das Datenrepositorium kennt zu allen Datenarten die entsprechende Quelle und kann auf Anforderung von Datensenken die Berechnung eines Datums auslösen und dieses an die Datensenke weiterleiten. Zusätzlich besitzt es eine Zwischenspeicherfunktionalität. 'Teure' Berechnungen können somit zwischengespeichert werden um aufwändige Neuberechnungen einzusparen.

Das Datenrepositorium wird durch die Klasse `DataManager` implementiert. Aus Zeitgründen wurden alle bisher verwendeten Datenquellen fest einprogrammiert, es soll jedoch in Zukunft möglich sein, beliebige Methoden bzw. Klassen als Datenquelle zu registrieren. Der Zugriff auf zur Laufzeit registrierte Daten wird als `ParameterSet` formuliert werden. Die Zwischenspeicherung von Daten ist ebenfalls fest programmiert, es sollte jedoch möglich sein, auch diesen Prozess zu automatisieren.

C.3 Modelle

Modelle werden in Motris durch die Klasse `Actor` repräsentiert. Ein Modell wird durch eine Datei im XML-Format beschrieben. In der Datei stehen in einer für Menschen lesbaren Form die einzelnen Körperteile, angeordnet in einer Baumstruktur. Es ist auf einfache Weise möglich, die Modelle in Anzahl der Körperteile, Anzahl der freigegebenen Haltungsparameter sowie Startwerten für die Parameter an die Anforderungen anzupassen, ohne dass Änderungen im Quelltext des Programms oder Änderungen in der sonstigen Konfiguration der Experimente nötig sind. Die Klasse `Actor` bezieht ihre Funktionalität im wesentlichen aus zwei untergeordneten Klassen: `State` und `RigidModel`. Die Klasse `State` ist die Implementierung des in dieser Arbeit als $\boldsymbol{\xi}$ bezeichneten Zustandsvektors. Zusätzlich zu den Werten der freien Parameter und ihrer zeitlichen Ableitungen (beliebigen Grades) enthält er die Kovarianzmatrix des Zustandsvektors und Funktionen zum Laden und Speichern des Zustands zu verschiedenen Zeitpunkten. Ausserdem wurde das Prozessmodell in `State` integriert, da für die zeitliche Fortschreibung des Zustands Detailwissen über die interne Darstellung der in `State` gespeicherten Daten benötigt wird. Das Prozessmodell kann durch Klassenvererbung verändert werden, wie die Klasse `CarState` demonstriert, die in [Dahlkamp 03] verwendet wird. Die in `State` gespeicherten Werte sind `Parameter`, die entweder vom Typ `Double` oder vom Typ `FreeParameter` sein können. `FreeParameter` sind die einzigen Parameter, die durch Prädiktion des Prozessmodells oder Anpassung durch ein Kalman-Filter verändert werden. Alle vorkommenden `FreeParameter` sind auch in

`State` gespeichert.

Die Klasse `RigidModel` repräsentiert ein einzelnes, starres Körperteil. Es besteht aus einer Transformationsmatrix, die den Übergang vom lokalen Koordinatensystem in das übergeordnete beschreibt, und einer Repräsentation des von dem Körperteil eingenommenen Volumens. Die Transformationsmatrix ist vom Typ `DOFMatrix` (**D**egrees **O**f **F**reedom-Matrix) und enthält eine Reihe von `Parametern`. Diese können unter anderem vom Typ `FreeParameter` sein. In diesem Fall nehmen sie bei Veränderungen des dem übergeordneten `Actor` zugehörigen `State` andere Werte an. Durch Aufruf der Aktualisierungsmethode `update` wird die Matrix mit den derzeitigen Werten der `Parameter` nach der Rollen-Stampfen-Gieren ('Roll-Pitch-Yaw') Konvention neu berechnet. Zusätzlich kann die Klasse `DOFMatrix` die ersten und zweiten Ableitungen nach ihren Freiheitsgraden (`FreeParameter`) berechnen.

Das Volumen des `RigidModel` wird durch Objekte beschrieben, deren Klasse die Schnittstelle `VolumeRepresentation` implementiert. In der vorliegenden Arbeit wird dazu die Klasse `GeneralizedCone` verwendet, die das Volumen eines parametrisierten, verallgemeinerten Kegelstumpfes umfasst. Die derzeit einzige weitere Volumenrepräsentation ist die Klasse `CarVolume`, die das Fahrzeugmodell nach [Koller 92] implementiert. Die Volumenrepräsentationen werden ebenfalls durch `Parameter` konfiguriert. Durch die Änderung eines Schlüsselworts in der XML-Datei eines Modells können so zusätzlich zu den Haltungsparametern auch beliebige Formparameter in die Menge der freien Parameter mit aufgenommen und durch Schätzprozesse beeinflußt werden. Weiterhin können in allen Schätzprozessen die Volumenrepräsentationen ohne weitere Änderungen ausgetauscht werden.

C.4 Verfolgung

Der Verfolgungsprozess wird in Motris formell in drei Ebenen unterteilt. Auf der obersten Ebene wird Verfolgung als eine abwechselnde Verkettung von Prädiktion und Anpassung gesehen. Dieser Ebene entspricht die Klasse `Tracker`, die neben der Konfiguration abwechselnd Prädiktion und Anpassung auslöst und die Ergebnisse speichert. In der zweiten Ebene stehen die Prädiktion und der Anpassungsprozess. Die Prädiktion wird durch das Modell selber, genauer durch die Implementierung des Zustandsvektors, `State` vorgenommen. Der Anpassungsprozess wurde im Rahmen dieser Arbeit als eine Minimierungsaufgabe verstanden. Dementsprechend ist dem `Tracker` ein `Minimizer` untergeordnet, ein Quadratsummenminimierer, der eine durch die Schnittstelle `MinmizationProblem` formulierte Residuumsfunktion minimiert. An Minimierungsalgorithmen wurden die beiden Verfahren nach Gauss-Newton und Levenberg-Marquardt durch gleichnamige Klassen implementiert.