DySim

Ein Werkzeug zur Simulation mechanischer Starrkörpersysteme

Version 0.5

Arnim Henze
(henze@tat.physik.uni-tuebingen.de)

Institut für Astronomie und Astrophysik
Eberhard-Karls-Universität Tübingen

September 2001

Übersicht

1 Die Installation von DySim

Die Quellen von DySim sind in einem komprimierten Archiv `dysim-0.5.tgz` zusammengefasst, das auch die Quellen der zusätzlich verwendeten Bibliotheken enthält, sofern nicht deren Originalversionen verwendet wurden.

1.1 Unterstützte Plattformen

Bisher ist DySim lediglich unter Linux getestet und verwendet worden. Als Kompiler für die C++-Quellen diente der GNU-Kompiler `g++` Version 2.95.2 [1], für die numerischen FORTRAN 77 Quellen wurden der GNU-Kompiler `g77` Version 2.95.2 [1] bzw. der Portland-Group Kompiler `pgf77` Version 3.2-4 [11] verwendet.

Da bei der Programmierung von DySim darauf geachtet wurde, ausschließlich mit den ISO-Normen [2, 3] konformes C und C++ zu verwenden, sollte eine Übertragung auf andere Betriebssysteme unproblematisch sein.

Bemerkung: Bei `g77` traten Probleme mit den statischen Variablen der FORTRAN-Quellen auf. Die maximale Optimierungsstufe `-O3` des `g77` lieferte fehlerhafte Objekt-Dateien, es sei denn, es wurde gleichzeitig die Option `-fno-automatic` verwendet, mit der sämtliche lokale Variable der Subroutinen als statisch deklariert werden.

1.2 Zusätzlich verwendete Bibliotheken

Da DySim auf der Theorie von Legnani et al. [8, 9] beruht, wurde auch die von den Autoren zusammengestellte Bibliothek SPACELIB in C Version 2.1 [7] verwendet. Allerdings gibt es in der originalen SPACELIB eine Inkompatibilität zur C++ STL, da SPACELIB eine eigene Funktion `vector()` definiert. Diese Inkompatibilität wurde behoben, indem die Funktion `vector()` in allen Quellen der SPACELIB in `unittov` umbenannt wurde. Die auf diese Weise modifizierten Quellen werden beim Kompilieren in der Bibliothek `libspacelib.a` zusammengefasst.

Einige der in DySim verwendeten Integratoren stammen aus den Numerical Recipes in C Version 2.08 [10, 12]. Allerdings mussten kleinere Veränderungen an den Quellen vorgenommen werden, um entsprechende Schnittstellen schaffen zu können, mit denen es möglich ist, auch kompilierte FORTRAN 77 Quellen von anderen Integratoren in DySim einzubinden, und um zusätzliche Parameter an die Routinen übergeben zu können.

Das Animationsprogramm AniDySim basiert auf OpenGL, d.h. es ist die OpenGL- bzw. die MesaGL-Bibliothek erforderlich. Weiterhin wurde für Maus- und Tastaturinteraktionen sowie für rudimentäre Fensterprogrammierung GLUT Version 3.7 [4] verwendet. Für die Ausgabe der Animation in Form von TIFF-Bildern, wird `libtiff` von Sam Leffler et al. [6] benötigt.

1.3 Die Verzeichnisstruktur des Simulationssystems

Das DySim-Archiv der Programmquellen muss unter Linux in das Verzeichnis `${HOME}/` des Benutzers entpackt werden, und das Basisverzeichnis des Simulationssystems ist standardmäßig `BASEDIR = ${HOME}/DySim/`. Wenn DySim in einem anderen Verzeichnis betrieben werden soll, dann muss die Variable `BASEDIR` in der Datei `<BASEDIR>/make/linux.mak`

auf die entsprechend andere Position angepasst werden. In jedem Fall ist dafür zu sorgen, dass das Verzeichnis `<BASEDIR>/bin/` für die ausführbaren Programme im Suchpfad enthalten ist.

Nach dem Entpacken ergibt sich die folgende Verzeichnisstruktur im Basisverzeichnis `BASEDIR`, wobei die mit `(*)` gekennzeichneten Dateien erst beim Kompilieren erzeugt werden:

```
SpaceLib----lib------------libspacelib.a (*)
                           Makefile
                           linear.c
                           linear2.c
                           printm67.c
                           spaceli3.c
                           spaceli4.c
                           spacelib.c
            include--------linear.h
                           spacelib.h
animate-----Makefile
            anidysim.cc
            anidysim.h
            primitive.cc
            primitive.h
            utils.cc
            utils.h
bin---------anidysim (*)
            dysim    (*)
doc---------dysim_doc.pdf
            dysim_doc.ps
include-----basicelement.h
            body.h
            dysim.h
            dysim_force.h
            dysim_solver.h
            dysim_spclib.h
            force.h
            forcectrl.h
            frame.h
            joint.h
            model.h
            modeldefault.h
            parmblock.h
            rawobject.h
            singularctrl.h
            solver.h
            solverctrl.h
            string_parse.h
            system.h
            triad.h
            nrutil.h
```

```
lib---------libdysim.a    (*)
           libelements.a (*)
           libintegs.a   (*)
           libparse.a    (*)
           libusrfrc.a   (*)
           libusrout.a   (*)
           libusrrfkt.a  (*)
make--------linux.mak
model-------example--------box.dys
                           gyro.dys
                           pendulum.dys
                           pendulum3.dys
           example_udef---makefile
                           usrfrc.cc
                           usrout.cc
                           usrrfkt.cc
src---------Makefile
           body.cc
           dysim.cc
           dysim_aux.cc
           dysim_force.cc
           dysim_memory.cc
           dysim_output.cc
           dysim_solver.cc
           dysim_usrfrc.cc
           dysim_usrout.cc
           dysim_usrrfkt.cc
           force.cc
           frame.cc
           joint.cc
           model.cc
           parmblock.cc
           solver.cc
           string_parse.cc
           system.cc
           triad.cc
           de.f
           bsstep.c
           mmid.c
           nrutil.c
           odeint.c
           pzextr.c
           rk4.c
           rkck.c
           rkqs.c
```

Das Ausführen von `make` in den Unterverzeichnissen `src/` und `animate/` des Basisverzeichnisses erzeugt die erforderlichen Bibliotheken und die ausführbaren Programme für DySim bzw. AniDySim.

2 Aufruf der Programme von der Kommandozeile

Derzeit gibt es noch zwei unterschiedliche Programme für die Simulation und für die spätere grafische Animation eines Modells: DySim zur Simulation und AniDySim zur Animation. Allerdings verwenden beide Programm dieselbe Modellkonfigurationsdatei. Als Standardname für diese Datei erwarten DySim und AniDySim einen Dateinamen mit der Endung `.dys`, wobei diese Endung bei beiden Programmaufrufen optional ist und ggf. automatisch ergänzt wird.

Eine DySim-Simulation eines Modells mit der Modellbeschreibung in `modell.dys` wird mittels

```
dysim modell[.dys]
```

gestartet und die nachfolgende AniDySim-Animation der auf diese Weise erzeugten Animationsdaten mittels

```
anidysim [-s -h] modell[.dys]
```

.

Die Option `-h` beendet AniDySim nach der Ausgabe einer kurzen Hilfe über das Dateiformat sowie die Tastatur- und Mausfunktionen wieder, mit `-s` wird AniDySim ohne die OpenGL-Funktionalität des *Double Buffering* gestartet.

3 Die Modellierungselemente

Bei der nun folgenden Beschreibung der Konfigurationsparameter und der Syntax der Konfigurationsdatei sind stets die Standardwerte angegeben, die DySim bzw. AniDySim – unter Ausgabe einer Warnung – automatisch verwendet, wenn diese zwingenden erforderlichen Parameter fehlen. Bei mehrfach auftretenden Parametern gilt stets die erste Definition, unnötige Angaben werden stillschweigend ignoriert.

Die Nomenklatur `parameter = <A|B>;` bezeichnet die Wahlmöglichkeit für den Parameter `parameter` zwischen den Werten `A` *oder* `B`, wobei immer der erste Wert, `A`, der Standardwert ist. Die Schreibweise `parameter2 = <parameter1>;` bedeutet, dass als Standardwert für den Parameter `parameter2` der Wert von `parameter1` verwendet wird.

Es wurde grundsätzlich versucht, die Namen für die Modellierungelemente und die Konfigurationsparameter so selbsterklärend wie möglich zu wählen.

3.1 Der prinzipielle Aufbau der Konfigurationsdatei

Die Modell-Konfigurationsdatei ist in ihrer groben Struktur aus einzelnen Modellierungsblöcken zusammengesetzt:

```
BLOCKBEZEICHNUNG "NAME" {
  StringParameter = Wort;
   :
  SkalarParameter = 0.0;
   :
  VektorParameter = 0.0, 0.0, 0.0;
   :
}
```

Ein Block besteht dabei jeweils aus einer `BLOCKBEZEICHNUNG`, die den Typ des Blocks kennzeichnet. Danach folgt ein innerhalb aller Blöcke von diesem Typ einmalig zu vergebender `NAME` in Anführungszeichen, mit dem diesem speziellen Block im Modell ein eindeutiger Name gegeben werden muss, um ihn von anderen Blöcken aus referenzieren zu können. Auf die Einmaligkeit der Namen sollte der Benutzer derzeit noch selber achten.
Die Blockdaten selber stehen wie in C/C++ in geschwungenen Klammern `{...}`. Die Parameter werden jeweils mit der Syntax `Parameter = Wert;` belegt, wobei `Wert` je nach Typ des Parameters eine Zeichenkette, ein Skalar oder aber auch die durch Kommata getrennten Komponenten eines Vektors sein können. Jede Parameterbelegung ist mit einem Semikolon abzuschließen.
Es gelten die gleichen Kommentarzeichen wie in C++, d.h. mit `//` kann eine Zeile bis zum Zeilenende auskommentiert werden und durch ein beginnendes `/*` und ein beendendes `*/` ein ganzer Bereich. Sämtliche sog. *White Space* Zeichen, wie Leerzeichen, Tabulatoren und Zeilenschaltungen, können zur Verbesserung der Übersichtlichkeit beliebig eingefügt werden, da sie bei der Auswertung der Datei ignoriert werden.

Das Inertialsystem heißt grundsätzlich `world`, so dass kein anderes Modellierungselement so genannt werden darf. Das Inertialsystem wird wie ein normaler Körper mit unendlicher Masse behandelt, d.h. es kann selber Mutterkörper sein, und es können Triaden an ihm definiert werden, um daran Gelenke anzubringen oder Kräfte wirken zu lassen.

3.2 Allgemeine Einstellungen

Allgemeine Einstellungen werden in einem `Header`- und in einem `System`-Block vorgenommen. Beide Blöcke müssen in der Konfigurationsdatei genau einmal vorkommen, eventuelle Wiederholungen werden ignoriert.

Der `Header`-Block enthält allgemeine, die Modellierung und die Analyse betreffende Einstellungen:

```
Header "NAME" {
  ModelName = MODELLNAME;
  OutputType = <none|all>;
}
```

Mit `OutputType = none;` wird sämtliche Datenausgabe, bis auf die Ausgabe in die Protokoll-Datei `<MODELLNAME>.log` unterbunden.

Für den `Header`-Block sind folgende Erweiterung/Verbesserungen geplant:

1. Detailliertere Steuerbarkeit der Ausgabe einzelner Simulationsdaten in Dateien.
2. Globale Definierbarkeit von Parametern, die dann in der Konfigurationdatei zur Belegung der Konfigurationsparameter verwendet werden können.

Im `System`-Block werden allgemeine, das mechanische System betreffende Einstellungen vorgenommen:

```
System "NAME" {
  GravAccel = 0.0, 0.0, -9.8065;
  ScaleGrav = 1.0;
}
```

Momentan sind in diesem Block als einzige Größen der Vektor der Erdbeschleunigung `GravAccel` sowie ein Skalierungsparameter `ScaleGrav` für diesen Vektor anzugeben.

3.3 Der Löser

Zur Festlegung des Analysetyps und zur Konfiguration des Lösers dient der `Solver`-Block. Er muss ebenfalls genau einmal in der Konfiguration auftreten, wobei Wiederholungen auch hier ignoriert werden.

Auf dem derzeitigen Stand wird lediglich eine *dynamische* Analyse, d.h. die zeitliche Integration mechanischer Systeme unterstützt:

```
Solver "NAME" {
  Type = dynamic;
   :
}
```

3.3.1 Integratoren

Für die dynamische Analyse eines mechanischen Systems stehen unterschiedliche Integratoren zur Verfügung, die im `Solver`-Block ausgewählt werden können. Der Standard-Integrator in DySim ist der `de` von Shampine/Gordon [13].

Es ist zu bemerken, dass bei allen Integratoren kleine Modifikationen gegenüber deren Originalquellen vorgenommen wurden. Im wesentlichen wurde dabei die Parameterklammer des Aufrufs zur Berechnung der Zustandsgeschwindigkeiten um einen Ganzzahl-Indikator und einen Ganzzahl-Vektor erweitert. Der Indikator zeigt bei `de` und `derf` an, ob es sich beim momentanen Integrationsschritt um einen Prediktor- oder um einen Korrektorschritt handelt und das Feld enthält bei `derf` ggf. die Werte diskreter, binären Zustände.

Neben der Start- und der Endzeit, `Tstart` und `Tend`, muss bei allen zur Verfügung stehenden Integratoren mit `Tstep` das zeitliche Ausgabeintervall der Simulationsdaten festgelegt werden, und mit `TstepMax` die maximale Integrationsschrittweite, die der Integrator innerhalb des Ausgabeintervalls verwenden darf.

rk4

Als primitiver Integrator steht das Runge-Kutta-Verfahren vierter Ordnung, `rk4` (vgl. Numerical Recipes in C [12], S. 712 f.), zur Verfügung:

```
Solver "NAME" {
  Type = dynamic;
  Solver = rk4;
  Tstart = 0.0;
  Tend = 1.0;
  Tstep = 0.01;
  TstepMax = Tstep;
}
```

odeint

Als Integrator mit adaptiver Schrittweitensteuerung kann `odeint` (vgl. Numerical Recipes in C [12], Kap. 16.2–16.4) verwendet werden, der entweder mit einem Crash-Karp Runge-Kutta-Verfahren fünfter Ordnung, `Solver = odeint_rk;`, oder mit einem Bulirsch-Stoer-Verfahren, `Solver = odeint_bs;` betrieben werden kann:

```
Solver "NAME" {
  Type = dynamic;
  Solver = <odeint_rk|odeint_bs>;
  AbsErr = 1.e-9;
  RelErr = 1.e-6;
  Tstart = 0.0;
  Tend = 1.0;
  Tstep = 0.01;
  TstepMax = <Tstep>;
  TstepMin = 0.0;
  TstepIni = 0.1*<Tstep>;
}
```

Für `odeint` gibt es als zusätzliche Parameter die absolute Genauigkeit `AbsErr`, d.h. der kleinste von Null verschieden anzusehende Betrag einer Zahl, den maximalen relativen Fehler `RelErr`, die minimale Integrationsschrittweite `TstepMin` sowie die Anfangsschrittweite `TstepIni`.

de

Als Standardintegrator steht der SHAMPINE/GORDON-Integrator `de` [13] zur Verfügung, der auf einem modifizierten ADAMS-Prediktor-Korrektor-Verfahren mit lokaler Extrapolation beruht, das zur Erreichung des maximalen lokalen Fehlers selbstständig die Ordnung und die Schrittweite variiert und ggf. die Integrationsparameter anpasst, bzw. Steifheit des Gleichungssystems meldet:

```
Solver "NAME" {
  Type = dynamic;
  Solver = de;
  AbsErr = 1.e-9;
  RelErr = 1.e-6;
  Tstart = 0.0;
  Tend = 1.0;
  Tstep = 0.01;
  TstepMax = <Tstep>;
}
```

Da sich `de` selbstständig um die Anpassung der Parameter kümmert und ein selbststartender Integrator ist, sind die absolute Genauigkeit `AbsErr`, d.h. der kleinste von Null verschieden anzusehende Betrag einer Zahl, und der maximale relative Fehler `RelErr` als einzige zusätzliche Parameter anzugeben.

derf

Um mechanische Systeme simulieren zu können, bei denen Unstetigkeiten im Differentialgleichungssystem beispielsweise aufgrund von Kontaktkräften auftreten können, steht `derf` zur Verfügung, der mit Ausnahme der Zeile `Solver = derf;` durch den gleichen `Solver`-Block definiert wird der `de`. Tatsächlich *muss* `derf` bei der Verwendung mancher Modellierungselemente zwingend ausgewählt werden.

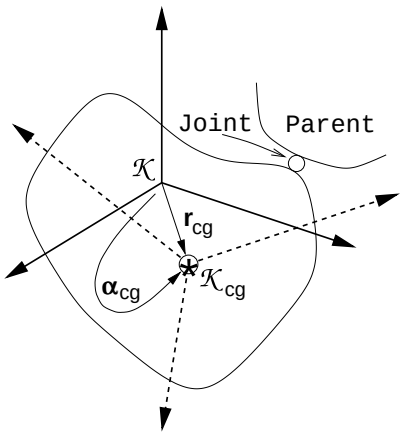
Bei `derf` handelt es sich um die Originalversion des `de`, die um eine Schalt-Funktionalität für diskrete, binäre Zustände erweitert wurde. Dabei erfolgt die Ermittlung der Schaltzeitpunkte durch die Nullstellendetektion von stetigen Schaltfunktionen, d.h. eine sog.

Root-Finder-Funktionalität (vgl. Abschnitt 7). Positive Werte der Schaltfunktion kennzeichnen den einen Wert des entsprechenden Zustands und negative den anderen. Ohne jegliche diskrete Zustände im Modell verhalten sich `de` und `derf` identisch.

3.4 Starrkörper

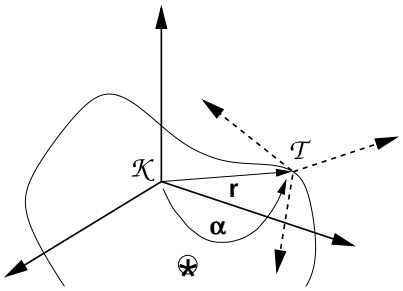
```
Body "NAME" {
  Parent = NAME-BODY-ATTACHED-TO;
  Joint = NAME-JOINT-ATTACHED-TO;
  Mass = 1.0;                                   ⟶  m
  Inertia = 1.0, 0.0, 0.0,                      ⟶  Θxx, Θxy, Θxz
                 1.0, 0.0,                      ⟶  Θyy, Θyz
                      1.0;                      ⟶  Θzz
  CGFramePosition = 0.0, 0.0, 0.0;              ⟶  r_cg
  CGFrameOrientation = 0.0, 0.0, 0.0;           ⟶  α_cg
  AngleType = <euler|cardan>;
  Degrees  = <rad|deg>;
}
```

Mit dem `Body`-Block wird ein starrer Körper definiert. Dazu muss der Name des Mutterkörpers, der auch das Inertialsystem `world` sein kann, und der Name des Gelenks, über das der Körper mit dem Mutterkörper verbunden ist, angegeben werden. Weiterhin muss die Masse m, der Trägheitstensor $\boldsymbol{\Theta}$ sowie die Lage $\boldsymbol{r}_{cg}$ und die Orientierung $\boldsymbol{\alpha}_{cg}$ des Schwerpunktsystems $\mathcal{K}_{cg}$ relativ zum körperfesten Koordinatensystem $\mathcal{K}$ angegeben werden. Die Beschreibung der Orientierung kann in Euler- oder in Kardan-Winkeln erfolgen. Das Schwerpunktsystem bezeichnet dasjenige Koordinatensystem, bzgl. dessen der Trägheitstensor definiert ist.



3.5 Koordinatensysteme

```
Triad "NAME" {
  Parent = BODYNAME;
  Position = 0.0, 0.0, 0.0;          ⟶  r
  Orientation = 0.0, 0.0, 0.0;       ⟶  α
  AngleType = <euler|cardan>;
  Degrees  = <rad|deg>;
}
```



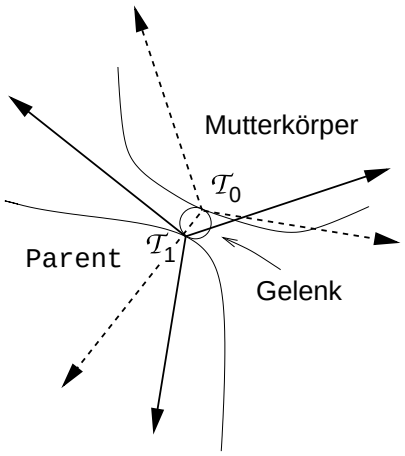
Auf jedem Körper oder im Inertialsystem können weitere Koordinatensysteme, sog. Triaden $\mathcal{T}$, definiert werden, um an ihnen beispielsweise Gelenke befestigen zu können oder Kräfte wirken zu lassen. Dabei beziehen sich die Lage $\boldsymbol{r}$ und die Orientierung $\boldsymbol{\alpha}$, die durch Euler- oder Kardan-Winkel parametrisiert werden kann, jeweils auf das körperfeste Koordinatensystem $\mathcal{K}$ von `Parent`, d.h. von demjenigen Körper, auf dem das Koordinatensystem definiert werden soll.

3.6 Gelenke

Die Modellierung in DySim erfolgt hierarchisch, d.h. vom Inertialsystem `world` ausgehend werden (verzweigte) kinematische Ketten in der Abfolge

Inertialsystem – Triade – Gelenk – Triade – Körper – Triade – Gelenk – Triade – Körper ...

gebildet. Jeder Körper wird also relativ zu einem Mutterkörper beschrieben und jedes Gelenk versieht genau einen Körper mit Freiheitsgraden. Aus diesem Grund muss für jedes Gelenk bei der Gelenkdefinition der Name des Körpers mit `Parent = NAME-BODY-ATTACHED;` angegeben werden, den das Gelenk mit dessen Mutterkörper verbinden soll, sowie die Namen der beiden Triaden $\mathcal{T}_0$ und $\mathcal{T}_1$, zwischen denen das Gelenk angebracht ist. Dabei kennzeichnet $\mathcal{T}_0$ die Triade auf dem Mutterkörper und $\mathcal{T}_1$ diejenige auf dem Körper `Parent`. Die (generalisierten) Koordinaten, die auf diese Weise definiert werden, sind die Lage und/oder die Orientierung von $\mathcal{T}_1$ relativ zu $\mathcal{T}_0$, die Koordinaten nehmen somit alle den Wert Null an, wenn die beiden Gelenktriaden aufeinander liegen.



3.6.1 Sechs-Freiheitsgrad-Gelenke

```
Free "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;                  ⟶  T0
  TriadTo = NAME-TO-TRIAD;                      ⟶  T1
  Qinit = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0;         ⟶  q0
  QDinit = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0;        ⟶  q̇0
  AngleType = <euler|cardan>;
  Degrees  = <rad|deg>;
}
```

Aufgrund der hierarchischen Definition des mechanischen Systems und der Philosophie von DySim, dass Gelenke benötigt werden, um Körper mit Freiheitsgraden zu versehen, benötigt selbst ein freier Körper ein Gelenk, das allerdings über drei Translations- sowie drei Rotationsfreiheitsgrade verfügt. Dabei kann für die Orientierung zwischen einer EULER- und einer KARDAN-Parametrisierung gewählt werden. Durch die beiden Vektoren $\boldsymbol{q}_0 = [q_x, q_y, q_z, q_{\alpha_1}, q_{\alpha_2}, q_{\alpha_3}]^T$ und $\dot{\boldsymbol{q}}_0 = [\dot{q}_x, \dot{q}_y, \dot{q}_z, \dot{q}_{\alpha_1}, \dot{q}_{\alpha_2}, \dot{q}_{\alpha_3}]^T$ werden die Anfangslage und die Anfangsgeschwindigkeit für den Körper `Parent` relativ zu seinem Mutterkörper festgelegt.

3.6.2 Klammergelenke

```
Bracket "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  T0
  TriadTo = NAME-TO-TRIAD;              ⟶  T1
}
```

Mit einem Klammergelenk können zwei Körper *starr* miteinander verkoppelt werden, so dass der Körper auf dem $\mathcal{T}_1$ liegt, relativ zu seinem Mutterkörper *keinen* Freiheitsgrad besitzt. Mit Hilfe dieses Gelenks kann die Bestimmung von Gesamtträgheitsmomenten zusammengesetzter Körper umgangen werden.

3.6.3 Scharniergelenke

```
Revolute "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  T_0
  TriadTo = NAME-TO-TRIAD;              ⟶  T_1
  JointAxis = 0.0, 0.0, 1.0;            ⟶  n
  JointPoint = 0.0, 0.0, 0.0;           ⟶  r_n
  Qinit = 0.0;                          ⟶  q_0
  QDinit = 0.0;                         ⟶  q̇_0
  Degrees  = <rad|deg>;
}
```

Ein Scharniergelenk versieht seinen Körper mit einem Rotationsfreiheitsgrad. Dazu müssen die Richtung der Gelenkachse $\boldsymbol{n} = [n_x, n_y, n_z]^T$ sowie ein Punkt $\boldsymbol{r}_n = [r_{n_x}, r_{n_y}, r_{n_z}]^T$, der auf dieser Achse liegen soll, in Koordinaten von $\mathcal{T}_0$ angegeben werden. q_0 und $\dot{q}_0$ legen den Anfangswinkel und die Anfangswinkelgeschindigkeit um die Gelenkachse fest.

3.6.4 Kardangelenke

```
Universal "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  T_0
  TriadTo = NAME-TO-TRIAD;              ⟶  T_1
  JointAxis1 = 1.0, 0.0, 0.0;           ⟶  n_1
  JointPoint1 = 0.0, 0.0, 0.0;          ⟶  r_n1
  JointAxis2 = 0.0, 1.0, 0.0;           ⟶  n_2
  JointPoint2 = 0.0, 0.0, 0.0;          ⟶  r_n2
  Qinit = 0.0, 0.0;                     ⟶  q_0
  QDinit = 0.0, 0.0;                    ⟶  q̇_0
  Degrees  = <rad|deg>;
}
```

Ein Kardangelenk versieht seinen Körper mit zwei Rotationsfreiheitsgraden. Dazu müssen in Koordinaten von $\mathcal{T}_0$ die beiden *linear unabhängigen* Gelenkachsen $\boldsymbol{n}_1 = [n_{1x}, n_{1y}, n_{1z}]^T$ und $\boldsymbol{n}_2 = [n_{2x}, n_{2y}, n_{2z}]^T$ definiert werden, sowie die beiden Punkte $\boldsymbol{r}_{n_1} = [r_{n_{1x}}, r_{n_{1y}}, r_{n_{1z}}]^T$ und $\boldsymbol{r}_{n_2} = [r_{n_{2x}}, r_{n_{2y}}, r_{n_{2z}}]^T$, durch die die Achsen jeweils verlaufen sollen. $\boldsymbol{q}_0 = [q_{1_0}, q_{2_0}]^T$ und $\dot{q}_0 = [\dot{q}_{1_0}, \dot{q}_{2_0}]^T$ legen die Anfangswinkel und die Anfangswinkelgeschindigkeiten um die Gelenkachsen fest.

3.6.5 Kugelgelenke

```
Spherical "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  $\mathcal{T}_0$
  TriadTo = NAME-TO-TRIAD;              ⟶  $\mathcal{T}_1$
  Qinit = 0.0, 0.0, 0.0;                ⟶  $\boldsymbol{q}_0$
  QDinit = 0.0, 0.0, 0.0;               ⟶  $\dot{\boldsymbol{q}}_0$
  AngleType = <euler|cardan>;
  Degrees  = <rad|deg>;
}
```

Ein Kugelgelenk erlaubt eine beliebige Orientierung zweier Körper relativ zueinander. Zur Parametrisierung dieser Orientierung kann zwischen der EULER- und der KARDAN-Darstellung gewählt werden, so dass mit $\boldsymbol{q}_0 = [q_{\alpha_{10}}, q_{\alpha_{20}}, q_{\alpha_{30}}]^T$ und $\dot{q}_0 = [\dot{q}_{\alpha_{10}}, \dot{q}_{\alpha_{20}}, \dot{q}_{\alpha_{30}}]^T$ die Anfangsorientierung und die Anfangswinkelgeschwindigkeit in EULER- oder KARDAN-Winkeln angegeben werden kann. Da bei einem Kugelgelenk in dieser Darstellung durch ein spezielles Winkelsystem nicht sichergestellt werden kann, dass die einzelnen Rotationsachsen stets linear unabhängig sind, überwacht DySim intern die Lage der Achsen und schaltet ggf. selbstständig auf das jeweils andere Winkelsystem um (vgl. dazu auch Abschnitt 6).

3.6.6 Prismagelenke

```
Prismatic "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  $\mathcal{T}_0$
  TriadTo = NAME-TO-TRIAD;              ⟶  $\mathcal{T}_1$
  JointAxis = 0.0, 0.0, 1.0;            ⟶  $\boldsymbol{n}$
  JointPoint = 0.0, 0.0, 0.0;           ⟶  $\boldsymbol{r}_n$
  Qinit = 0.0;                          ⟶  $q_0$
  QDinit = 0.0;                         ⟶  $\dot{q}_0$
}
```

Ein Prismagelenk erlaugt eine relative Translationsbewegung zweier Körper entlang einer Achse. Dazu müssen die Richtung der Gelenkachse $\boldsymbol{n} = [n_x, n_y, n_z]^T$ sowie ein Punkt $\boldsymbol{r}_n = [r_{n_x}, r_{n_y}, r_{n_z}]^T$, der auf dieser Achse liegen soll, in Koordinaten von $\mathcal{T}_0$ angegeben werden. q_0 und $\dot{q}_0$ legen die Anfangsverschiebung und die Anfangsgeschwindigkeit entlang der Gelenkachse fest.

3.6.7 2D-Translationsgelenke

```
BiPrismatic "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  $\mathcal{T}_0$
  TriadTo = NAME-TO-TRIAD;              ⟶  $\mathcal{T}_1$
  JointAxis1 = 1.0, 0.0, 0.0;           ⟶  $\boldsymbol{n}_1$
  JointPoint1 = 0.0, 0.0, 0.0;          ⟶  $\boldsymbol{r}_{n_1}$
  JointAxis2 = 0.0, 1.0, 0.0;           ⟶  $\boldsymbol{n}_2$
  JointPoint2 = 0.0, 0.0, 0.0;          ⟶  $\boldsymbol{r}_{n_2}$
  Qinit = 0.0, 0.0;                     ⟶  $\boldsymbol{q}_0$
  QDinit = 0.0, 0.0;                    ⟶  $\dot{\boldsymbol{q}}_0$
}
```

Ein 2D-Translationsgelenk erlaubt eine relative Translationsbewegung zweier Körper in einer Ebene, es versieht seinen Körper demnach mit zwei Freiheitsgraden. Dazu müssen in Koordinaten von $\mathcal{T}_0$ die beiden *linear unabhängigen* Achsen $\boldsymbol{n}_1 = [n_{1x}, n_{1y}, n_{1z}]^T$ und $\boldsymbol{n}_2 = [n_{2x}, n_{2y}, n_{2z}]^T$ angegeben werden, die die Ebene aufspannen, sowie die beiden Punkte $\boldsymbol{r}_{n_1} = [r_{n_{1x}}, r_{n_{1y}}, r_{n_{1z}}]^T$ und $\boldsymbol{r}_{n_2} = [r_{n_{2x}}, r_{n_{2y}}, r_{n_{2z}}]^T$, durch die die Achsen jeweils verlaufen sollen. $\boldsymbol{q}_0 = [q_{1_0}, q_{2_0}]^T$ und $\dot{q}_0 = [\dot{q}_{1_0}, \dot{q}_{2_0}]^T$ legen die Anfangslage und die Anfangsgeschindigkeit auf der Ebene fest.

3.6.8 3D-Translationsgelenke

```
TriPrismatic "NAME" {
  Parent = NAME-BODY-ATTACHED;
  TriadFrom = NAME-FROM-TRIAD;          ⟶  T_0
  TriadTo = NAME-TO-TRIAD;              ⟶  T_1
  Qinit = 0.0, 0.0, 0.0;                ⟶  q_0
  QDinit = 0.0, 0.0, 0.0;               ⟶  q̇_0
}
```

Ein 3D-Translationsgelenk versieht seinen Körper mit drei Translationsfreiheitsgraden, so dass er beliebig relativ zu seinem Mutterkörper verschoben, jedoch nicht verdreht werden kann. Die Anfangslage sowie die Anfangsgeschwindigkeit legen $\boldsymbol{q}_0 = [q_{x_0}, q_{y_0}, q_{z_0}]^T$ und $\dot{q}_0 = [\dot{q}_{x_0}, \dot{q}_{y_0}, \dot{q}_{z_0}]^T$ fest.

3.7 Kräfte

Zur Definition von Kräften müssen neben den individuellen Parametern für jedes Kraftelement lediglich die Namen der beiden Triaden $\mathcal{T}_0$ und $\mathcal{T}_0$ angegeben werden, an denen die jeweiligen Kräfte angreifen sollen. Über die eindeutige Zuordnung der Triaden zu Körpern steht damit auch fest, zwischen welchen Körpern die Kräfte jeweils wirken. Sämtliche Gleichungen geben die Kraft bzw. das Drehmoment an, wie sie bzw. es auf den Körper mit der Triade $\mathcal{T}_0$ wirkt.

3.7.1 Gedämpft-linearelastischer Faden

```
Spring "NAME" {
  TriadFrom = NAME-FROM-TRIAD;          ⟶  T_0
  TriadTo = NAME-TO-TRIAD;              ⟶  T_1
  ElasticLinear = 0.0;                  ⟶  κ
  DampingLinear = 0.0;                  ⟶  ρ
  ZeroLength = 0.0;                     ⟶  s_0
  OneSide = <bi|+|->;
}
```

Mit dem Fadenelement (entspannte Länge s_0) kann auf der Verbindungslinie $\boldsymbol{e}_s$ zwischen den Ursprüngen der Kraftangriffstriaden $\mathcal{T}_0$ und $\mathcal{T}_1$, an denen das Kraftelement befestigt ist, eine lineare, dissipative Kraft definiert werden,

$$\boldsymbol{f} = (\kappa\,(s - s_0) + \rho\,\dot{s})\;\boldsymbol{e}_s \quad ,$$

die für `OneSide = bi;` anziehend *und* abstoßend, für `OneSide = -;` nur anziehend und für `OneSide = +;` nur abstoßend wirkt.

3.7.2 Gedämpft-linearelastische Drehfeder

```
RSDA "NAME" {
  Joint1D = NAME-JOINT-ACTING-ON;
  ElasticLinear = 0.0;                    ⟶  κ
  DampingLinear = 0.0;                    ⟶  ρ
  ZeroAngle = 0.0;                        ⟶  φ0
  OneSide = <bi|+|->;
  Degrees  = <rad|deg>;
}
```

Mit der Drehfeder (entspannter Winkel φ_0) kann direkt in dem Scharniergelenk, dessen Name mittels `Joint1D = NAME-JOINT-ACTING-ON;` festgelegt wird, ein lineares, dissipatives Drehmoment definiert werden,

$$\tau_\varphi = \kappa\,(\varphi_0 - \varphi) - \rho\,\dot{\varphi} \quad ,$$

das für `OneSide = bi;` im negativen *und* positiven Drehsinn, für `OneSide = -;` nur im negativen und für `OneSide = +;` nur im positiven Drehsinn wirkt.

3.7.3 Gedämpft-linearelastische Translationsfeder

```
TSDA "NAME" {
  Joint1D = NAME-JOINT-ACTING-ON;
  ElasticLinear = 0.0;                    ⟶  κ
  DampingLinear = 0.0;                    ⟶  ρ
  ZeroLength = 0.0;                       ⟶  q0
  OneSide = <bi|+|->;
}
```

Mit der Translationsfeder (entspannte Länge q_0) kann direkt in dem Prismagelenk, dessen Name mittels `Joint1D = NAME-JOINT-ACTING-ON;` festgelegt wird, eine lineare, dissipative Kraft definiert werden,

$$f_q = \kappa\,(q_0 - q) - \rho\,\dot{q} \quad ,$$

die für `OneSide = bi;` anziehend *und* abstoßend, für `OneSide = -;` nur anziehend und für `OneSide = +;` nur abstoßend wirkt.

3.7.4 Einfaches, gedämpft-linearelastisches Kontaktelement

```
ContactSimple "NAME" {
  TriadFrom = NAME-FROM-TRIAD;            ⟶  T0
  TriadTo = NAME-TO-TRIAD;                ⟶  T1
  ElasticLinearZ = 0.0;                   ⟶  κ⊥
  DampingLinearZ = 0.0;                   ⟶  ρ⊥
  FrictionXY = 0.0;                       ⟶  ρ∥
}
```

Mit diesem einfachen Kontaktelement können auf rudimentäre Weise Kontakte zwischen Punkten und Flächen beschrieben werden (vgl. Abbildung 1). Dabei wird die Kontaktfläche durch die Triade $\mathcal{T}_0$ definiert: Die z-Komponente dieser Triade legt die Flächennormale fest, während sich die Fläche selber in der xy-Ebene des Koordinatensystems erstreckt. Zum Kontakt kann es kommen, wenn der senkrechte relative Abstand $\boldsymbol{r}_\perp$ des

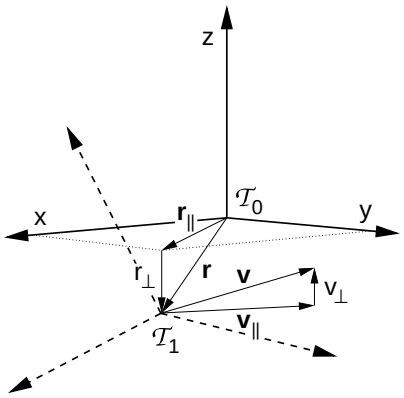


Abbildung 1: Die Definitionen in der Kontaktsituation. $\mathcal{T}_0$ legt die Kontaktfläche fest und der Ursprung von $\mathcal{T}_1$ den Kontaktpunkt.

Ursprungs von $\mathcal{T}_1$ zur xy-Ebene von $\mathcal{T}_0$ negativ wird. Die Kontaktdissipation ist proportional zur senkrechten Komponente $v_\perp$ der Relativgeschwindigkeit. Tangential zur Kontaktfläche wirkt lediglich eine Gleitreibungskraft, die proportional zur tangentialen Komponente $\boldsymbol{v}_\parallel$ der Relativgeschwindigkeit der Triaden ist. In Koordinaten der Triade $\mathcal{T}_0$ lautet die Kontaktkraft auf den Körper an dem $\mathcal{T}_0$ befestigt ist:

$$\boldsymbol{f} = \begin{pmatrix} f_\perp \\ \boldsymbol{f}_\parallel \end{pmatrix} = \begin{cases} \begin{pmatrix} \kappa_\perp\, r_\perp + \rho_\perp\, v_\perp \\ \rho_\parallel\, \boldsymbol{v}_\parallel \end{pmatrix} & : \text{solange } f_\perp < 0 \\ \mathbf{0} & : \text{sonst} \end{cases} .$$

Sobald die Vertikalkomponente der Kontaktkraft auf den $\mathcal{T}_0$-Körper nicht mehr negativ ist, besteht kein Kontakt mehr. Dies kann auch der Fall sein, wenn $r_\perp$ zwar negativ ist, sich aufgrund des Reibungsterms jedoch eine positive, d.h. anziehende Komponente $f_\perp$ ergibt.

3.7.5 Komplexes Kontaktelement

```
Contact "NAME" {
  TriadFrom = NAME-FROM-TRIAD;          ⟶  T_0
  TriadTo = NAME-TO-TRIAD;              ⟶  T_1
  ElasticLinearZ = 0.0;                 ⟶  κ_⊥
  DampingLinearZ = 0.0;                 ⟶  ρ_⊥^01
  DampingO2MixedZ = 0.0;                ⟶  ρ_⊥^11
  SlipDampingLinear = 0.0;              ⟶  σ
  SlipFrictionCoeff = 0.0;              ⟶  μ
  VSlipMin = 0.0;                       ⟶  v_c
  SlipFrictionCoeffRot = 0.0;           ⟶  μ_φ
  OmegaSlipMin = 0.0;                   ⟶  ω_c
  StickElasticLinear = 0.0;             ⟶  κ_∥
  StickDampingLinear = 0.0;             ⟶  ρ_∥
  StaticFrictionCoeff = 0.0;            ⟶  μ_0
  StickElasticLinearRot = 0.0;          ⟶  κ_φ
  StickDampingLinearRot = 0.0;          ⟶  ρ_φ
  StaticFrictionCoeffRot = 0.0;         ⟶  μ_0φ
  Degrees  = <rad|deg>;
}
```

Mit diesem Kontaktkraftelement kann der Kontakt zwischen einem Punkt im Ursprung der Kontakttriade $\mathcal{T}_1$ und einer Fläche, beschrieben durch die xy-Ebene der Kontakttriade $\mathcal{T}_0$, (vgl. Abbildung 1) detaillierter als mit dem einfachen Kontaktelement modelliert

werden. Um den immensen numerischen Aufwand bei der Integration der Bewegungsgleichungen zu umgehen, der aufgrund der Form des Kraft-/Drehmomentgesetzes bei großen Reibungskonstanten $\rho_{\parallel}$ oder bei COULOMB-Reibung immer dann auftritt, wenn die Relativgeschwindigkeit der Kontakttriaden sehr klein wird, umfasst dieses komplexe Kraftelement neben den Zuständen *Kontakt* und *kein Kontakt* in Richtung der Flächennormalen für die Bewegung tangential zur Fläche zwei weitere Zustände: *Gleiten* und *Haften*. Wenn bei sehr langsamem Gleiten in den Zustand des Haftens umgeschaltet wird, können somit der numerische Integrationsaufwand enorm reduziert und die realen Eigenschaften von Kontakten besser modelliert werden. Um auch flächige Kontakte mit diesem idealisierten Punkt-Flächen-Kontaktelement imitieren zu können, wirken sich die Zustände sowohl auf die Translationen als auch auf die Rotationen von $\mathcal{T}_1$ relativ zu $\mathcal{T}_0$ aus. Ohne flächennormale Reibungsdrehmomente und ohne das Haften der Orientierung bzgl. Verdrehungen um die Flächennormale wäre jede Rotation um den Kontaktpunkt vollkommen frei und $\mathcal{T}_1$ würde sich wie eine Nadelspitze auf der Kontaktfläche verhalten. Bei einem flächigen Kontakt, wie er in der Realität in der Regel vorkommt, können solche Torsionsmomente jedoch übertragen werden.

Die Schaltkriterien erläutert Abbildung 2: Solange die zur Fläche senkrechte Komponente des Abstandsvektors $\boldsymbol{r}$ von $\mathcal{T}_0$ nach $\mathcal{T}_1$ negativ ist und die senkrechte Komponente der Kontaktkraft abstoßend wirkt, besteht Kontakt. Nur dann wirkt am Kontaktpunkt eine Kontaktkraft $\boldsymbol{f} = [f_{\perp}, \boldsymbol{f}_{\parallel}]^T$ und ein Kontaktdrehmoment $\tau = [0, 0, \tau_{\perp}]^T$ zwischen den beiden Körpern, auf denen die Kontakttriaden liegen. Die tangentiale Komponente der Kontaktkraft hängt davon ab, ob sich das Kontaktelement im Gleit- oder im Haftzustand befindet. Ab dem Moment, in dem der Kontakt eintritt, liegt zunächst Gleiten vor, und es wirkt eine Gleitreibungskraft parallel und ein Gleitreibungsmoment senkrecht zur Kontaktfläche. Sobald sowohl der Betrag der Tangentialgeschwindigkeit $\boldsymbol{v}_{\parallel}$ von $\mathcal{T}_1$ relativ zu $\mathcal{T}_0$ unter eine Grenze v_c als auch die Normalkomponente $\omega_{\perp}$ von deren relativer Winkelgeschwindigkeit unter die Grenze ω_c sinkt, schaltet das Kraftelement in den Zustand des Haftens. In diesem Zustand wirken eine gedämpft-elastische Haftkraft parallel und ein gedämpft-elastisches Haftdrehmoment senkrecht zur Kontaktfläche. Die Kontakttriaden haften, so lange Kontakt besteht und sowohl der Betrag der wirkenden Haftkraft $\boldsymbol{f}_{\parallel}$ unterhalb einer Grenze f_c als auch der Betrag des Haftdrehmoments $\tau_{\perp}$ unterhalb der Grenze τ_c liegt. Wird mindestens eine dieser Bedingungen verletzt, so schaltet das Kontaktelement zurück in den Zustand des Gleitens.

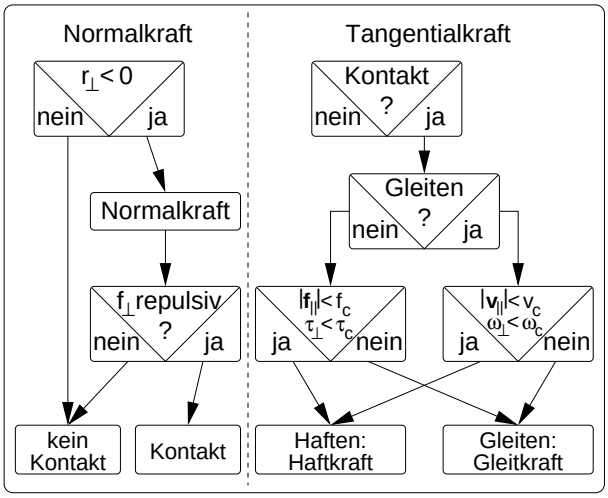


Abbildung 2: Zustandskriterien im komplexen Kontaktelement.

Die Grenzwerte für die maximale Haftung werden durch Haftreibungskoeffizienten parametrisiert:

$$\begin{aligned} f_c &= \mu_0 \, |f_{\perp}| \\ \tau_c &= \mu_{0\phi} \, |f_{\perp}| \quad . \end{aligned}$$

Die Normalkomponente der Kontaktkraft berechnet sich aus den Normalkomponenten $r_\perp$ bzw. $v_\perp$ des Abstandvektors bzw. der Relativgeschwindigkeit durch

$$f_\perp = \kappa_\perp \, r_\perp + \rho_\perp^{01} \, v_\perp - \rho_\perp^{11} \, r_\perp \, v_\perp \quad .$$

Die Gleitreibungskraft und das Gleitreibungsmoment werden aus der Normalkraft $f_\perp$ und der Tangentialkomponente $\boldsymbol{v}_\parallel$ der Relativgeschwindigkeit bzw. der Normalkomponente der relativen Winkelgeschwindigkeit $\omega_\perp$ bestimmt, d.h. die Reibungskraft kann eine geschwindigkeitsabhängige Geschwindigkeits- sowie eine COULOMB-Reibungskomponente und das Reibungsmoment eine modifizierte COULOMB-Reibungskomponente enthalten:

$$\begin{aligned} \boldsymbol{f}_\parallel &= \sigma \boldsymbol{v}_\parallel + \mu \, |f_\perp| \, \frac{\boldsymbol{v}_\parallel}{|\boldsymbol{v}_\parallel|} \\ \tau_\perp &= \mu_\phi \, |f_\perp| \, \omega_\perp \quad . \end{aligned}$$

Ab dem Zeitpunkt, an dem die Beträge der Tangentialgeschwindigkeit und der Normalkomponente der Winkelgeschwindigkeit die Grenzwerte v_c und ω_c unterschreiten, haftet $\mathcal{T}_1$ auf der Kontaktfläche. Dazu wird im Moment des Schaltens von Gleiten nach Haften eine Hafttriade $\overline{\mathcal{T}}$ an der Position von $\mathcal{T}_1$ auf der Kontaktfläche definiert und $\mathcal{T}_1$ gedämpft-elastisch daran gekoppelt. Dadurch sollen zum einen die xy-Koordinate des Ursprungs von $\mathcal{T}_1$ relativ zu $\mathcal{T}_0$ und die Orientierung von $\mathcal{T}_1$ bzgl. Verdrehungen um die Flächennormale fixiert werden können.

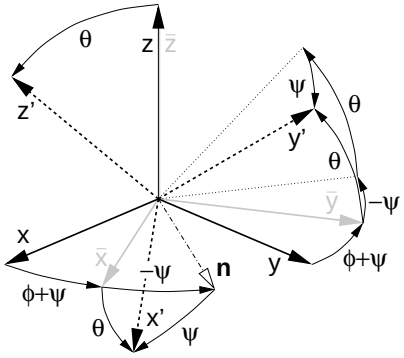


Abbildung 3: Beschreibung der Orientierung des gestrichenen Achsensystems $\mathcal{T}_1$ (gestrichelt) relativ zum ungestrichenen System $\mathcal{T}_0$ (durchgezogen) durch eine Kippung um die Achse $\boldsymbol{n}$ und eine Torsion um die $z/\bar{z}$-Achse mit Hilfe von EULER-Winkeln $\{\phi, \theta, \psi\}$. $\mathcal{T}_0$ legt mit seiner xy-Ebene die Kontaktfläche fest, in der die Kippachse liegt, und $\mathcal{T}_1$ den Kontaktpunkt. Auf diese Weise kann die Hafttriade $\overline{\mathcal{T}}$ (in grau) eindeutig definiert werden.

Während die Festlegung der Haftposition als die senkrechte Projektion des Ursprungs von $\mathcal{T}_1$ auf die Kontaktfläche zum Schaltzeitpunkt unproblematisch ist, ist für die Definition der Orientierung der Hafttriade eine geeignete Beschreibung nötig. Da im Zustand des Haftens lediglich gedämpft-elastische Torsionsmomente übertragbar sein sollen, jedoch keine Drehmomente um Achsen in der Kontaktfläche, wird die Orientierung von $\mathcal{T}_1$ relativ zu $\mathcal{T}_0$ in eine Rotation um die Flächennormale, d.h. eine Torsion, und eine Rotation um eine Achse in der Fläche, d.h. eine Kippung, aufgeteilt.

Dies geschieht, indem die EULER-Winkel $\{\phi, \theta, \psi\}$ bestimmt werden, die die Orientierung von $\mathcal{T}_0$ in diejenige von $\mathcal{T}_1$ überführen (vgl. Abbildung 3). Die Gesamttransformation $\mathcal{T}_0 \xrightarrow{D} \mathcal{T}_1$ kann nun in einen Torsionsanteil $\boldsymbol{D}_\perp$ und einen Kippanteil $\boldsymbol{D}_\parallel$ aufgeteilt werden,

$$\boldsymbol{D} = \boldsymbol{D}_z(\phi) \, \boldsymbol{D}_x(\theta) \, \boldsymbol{D}_z(\psi) = \underbrace{\boldsymbol{D}_z(\phi + \psi)}_{\boldsymbol{D}_\perp} \, \underbrace{\boldsymbol{D}_z(-\psi) \, \boldsymbol{D}_x(\theta) \, \boldsymbol{D}_z(\psi)}_{\boldsymbol{D}_\parallel} \quad ,$$

d.h. der Torsionswinkel ist $\xi = \phi + \psi$, der Kippwinkel beträgt θ und die Kippachse in der xy-Ebene von $\mathcal{T}_0$ ist $\boldsymbol{n} = \boldsymbol{D}_z(\phi) \, \boldsymbol{e}_x$.

Im Moment des Haftens kann die Hafttriade $\overline{\mathcal{T}}$ mit den Koordinatenachsen $\{\bar{x}, \bar{y}, \bar{z}\}$ somit eindeutig definiert werden: Sie ist um den Torsionswinkel $\bar{\xi} = \bar{\phi} + \bar{\psi}$ um die Flächennormale gegenüber $\mathcal{T}_0$ verdreht, so dass $\mathcal{T}_1$ aus ihr zu diesem Zeitpunkt lediglich durch eine Kippung entlang der Achse $\boldsymbol{n}$ hervorgeht.

Die tangentiale Kraft, die den Kontaktpunkt im Zustand des Haftens in der Fläche fixiert, wird aus der Auslenkung des Ursprungs von $\mathcal{T}_1$ gegenüber $\overline{\mathcal{T}}$ in der Kontaktfläche, $\boldsymbol{r}_\parallel - \bar{\boldsymbol{r}}_\parallel$, und der tangentialen Geschwindigkeit $\boldsymbol{v}_\parallel$ des Kontaktpunkts auf der Fläche bestimmt:

$$\boldsymbol{f}_\parallel = \kappa_\parallel \left(\boldsymbol{r}_\parallel - \bar{\boldsymbol{r}}_\parallel\right) + \rho_\parallel \, \boldsymbol{v}_\parallel \quad .$$

Bei Verdrehungen von $\mathcal{T}_1$ relativ zur Hafttriade wirkt zwischen den beiden Kontakttriaden ein rücktreibendes Haftdrehmoment um die Flächennormale:

$$\tau_\perp = \kappa_\phi \, \Delta\xi + \rho_\phi \, \omega_\perp \quad .$$

Dabei ist $\omega_\perp$ die zur Fläche senkrechte Komponente der Winkelgeschwindigkeit von $\mathcal{T}_1$ relativ zu $\mathcal{T}_0$, und $\Delta\xi$ ist der Torsionswinkel zwischen $\overline{\mathcal{T}}$ und $\mathcal{T}_1$.

Mit sehr großen Werten für die Grenze der translatorischen Relativgeschwindigkeit oder der relativen Winkelgeschwindigkeit zwischen den beiden Kontakttriaden kann der Zustand des Gleitens verhindert werden, so dass mit beginnendem Bodenkontakt sofort auch Haften eintritt. Umgekehrt deaktivieren $v_c = \omega_c = 0$ das Schalten zwischen Gleiten und Haften, so dass bei bestehendem Kontakt stets Gleiten vorliegt.

3.7.6 6D-Kraft-Drehmoment-Element

```
Bushing "NAME" {
  TriadFrom = NAME-FROM-TRIAD;                          ⟶  T_0
  TriadTo = NAME-TO-TRIAD;                              ⟶  T_1
  ElasticLinear = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,        ⟶  K
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0;
  DampingLinear = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,        ⟶  R
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0;
  AngleType = <cardan|euler>;
  Degrees = <rad|deg>;
}
```

Mit einem 6D-Kraft-Drehmoment-Element, einem sog. *Bushing*-Element, können zwei Körper über die Triaden $\mathcal{T}_0$ und $\mathcal{T}_1$ durch eine beliebige, lineare, gedämpft-elastische Kopplung miteinander verbunden werden. Dazu werden zwei 6×6 Matrizen, die Steifigkeitsmatrix $\boldsymbol{K}$ und die Dämpfungsmatrix $\boldsymbol{R}$ definiert, die eine beliebige lineare Verknüpfung der

Relativkoordinaten, d.h. der drei Translations- und der drei Winkelkoordinaten erlauben:

$$\begin{pmatrix} f_x \\ f_y \\ f_z \\ \tau_\alpha \\ \tau_\beta \\ \tau_\gamma \end{pmatrix} = K \begin{pmatrix} r_x \\ r_y \\ r_z \\ \alpha \\ \beta \\ \gamma \end{pmatrix} + R \begin{pmatrix} v_x \\ v_y \\ v_z \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{pmatrix} \quad .$$

Momentan wird lediglich die KARDAN-Repräsentation der Orientierung unterstützt. Die Geschwindigkeiten der KARDAN-Winkel lassen sich aus dem Vektor der Winkelgeschwindigkeit $\boldsymbol{\omega}$ ermitteln,

$$\begin{pmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{pmatrix} = \frac{1}{\cos\beta} \begin{pmatrix} \cos\beta & \sin\alpha\,\sin\beta & -\cos\alpha\,\sin\beta \\ 0 & \cos\alpha\,\cos\beta & \sin\alpha\,\cos\beta \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix} \boldsymbol{\omega} \quad ,$$

und die kartesischen Komponenten des Kopplungsdrehmoments lauten als Funktion der KARDAN-Komponenten

$$\boldsymbol{\tau} = \begin{pmatrix} 1 & 0 & \sin\beta \\ 0 & \cos\alpha & -\sin\alpha\,\cos\beta \\ 0 & \sin\alpha & \cos\alpha\,\cos\beta \end{pmatrix} \begin{pmatrix} \tau_\alpha \\ \tau_\beta \\ \tau_\gamma \end{pmatrix} \quad .$$

Aufgrund der Koordinatensingularität, die bei jedem Winkelsystem existiert und die bei KARDAN-Winkeln für $\beta = \pi/2$ auftritt, muss bei diesem Kraftelement darauf geachtet werden, dass die relativen Auslenkungen der Triaden $\mathcal{T}_0$ und $\mathcal{T}_1$ nicht zu groß werden.

3.8 Grafikprimitive

Zur Animation der simulierten Modellbewegung mit AniDySim können die einzelnen Körper durch `Primitive`-Blöcke in der Konfigurationsdatei mit Grafikprimitiven versehen werden. Dabei ist es möglich, jedem Segment beliebig viele unterschiedliche geometrische Objekte zuzuweisen, um so aus einfachen Primitiven komplexe Körpergeometrien zusammenzusetzen. Das Inertialsystem wird prinzipiell durch ein Dreibein mit roter x-, grüner y- und blauer z-Achse dargestellt, dem noch weitere Primitive hinzugefügt werden können.

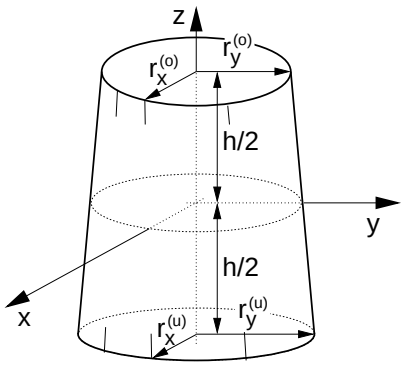
```
Primitive "NAME" {
  Parent = NAME-BODY-ATTACHED-TO;
  Position = 0.0, 0.0, 0.0;
  Cardan = 0.0, 0.0, 0.0;
  Degrees  = <rad|deg>;
  Color = 0.0, 0.0, 0.0, 0.0;
  :
}
```

Jedem geometrischen Objekt muss ein `Parent`, d.h. der Name des Körpers zugeordnet werden, den es visualisieren soll. Weiterhin sind die Position des Objektkoordinatensystems sowie seine Orientierung – parametrisiert durch KARDAN-Winkel in Radian oder Grad – relativ zum Körperkoordinatensystem anzugeben. Die Farbe des Objekts wird

durch vier Werte jeweils aus dem Intervall $[0, 1]$ festgelegt: Der erste Wert bestimmt den Rot-, der zweite den Grün- und der dritte den Blau-Anteil. Die vierte Komponente des Farbvektors skaliert die Intensität der durch die drei Farbanteile additiv erzeugten Mischfarbe. Alle grafischen Objekte sind rein kosmetischer Natur und haben keinerlei Einfluss auf die Trägheitseigenschaften, wie sie im jeweiligen `Body`-Block des zugeordneten physikalischen Körpers definiert werden. Der Typ eines geometrischen Objekts wird mit dem Schlüsselwort `Type` festgelegt, wobei dessen Form je nach Objekttyp mit weiteren Parametern genauer zu spezifizieren ist.

3.8.1 Zylinder

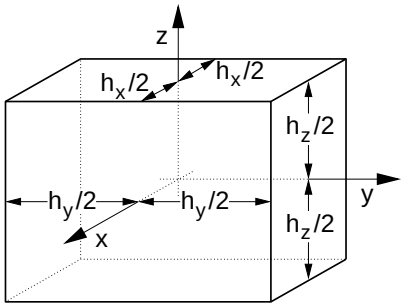
```
Primitive "NAME" {
  :
  Type = Cylinder;
  RadiusBottomX = 1.0;      ⟶  r_x^(o)
  RadiusBottomY = 1.0;      ⟶  r_y^(o)
  RadiusTopX = 1.0;         ⟶  r_x^(u)
  RadiusTopY = 1.0;         ⟶  r_y^(u)
  HeightZ = 1.0;            ⟶  h
  GridLong = 8;
}
```



Ein Zylinder der Höhe h mit elliptischen Stirnseiten, dessen Symmetrieachse auf der z-Achse und dessen geometrischer Schwerpunkt im Ursprung des Objektkoordinatensystems liegt, wird durch `Type = Cylinder;` ausgewählt. Die Radien der beiden elliptischen Flächen in x- bzw. in y-Richtung, $r_x^{(o)}$ und $r_y^{(o)}$ bzw. $r_x^{(u)}$ und $r_y^{(u)}$, können unabhängig festgelegt werden, und mit `GridLong` ist die Anzahl der Längenkreisunterteilungen, d.h. die Glattheit der Flächendarstellung wählbar.

3.8.2 Quader

```
Primitive "NAME" {
  :
  Type = Box;
  LengthX = 1.0;            ⟶  h_x
  LengthY = 1.0;            ⟶  h_y
  LengthZ = 1.0;            ⟶  h_z
}
```



Ein Quader mit den Seitenlängen h_x, h_y und h_z dessen geometrischer Schwerpunkt im Ursprung des Objektkoordinatensystems liegt und dessen Seitenflächen parallel zu den Achsenebenen liegen, wird durch `Type = Box;` ausgewählt.

3.8.3 Dreibein

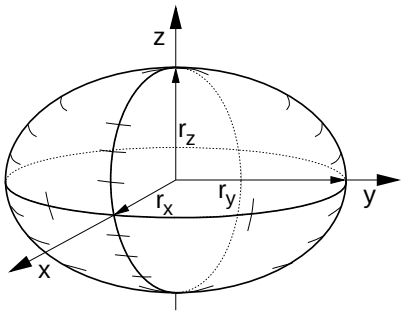
```
Primitive "NAME" {
  :
  Type = Axes;
  Length = 0.25;
  ColorX = <Color>;
```

```
  ColorY = <Color>;
}
```

Mit `Type = Axes;` kann das Objektkoordinatensystem mit einem Dreibein von beliebiger Achslänge visualisiert werden. Zusätzlich ist es möglich, die x- und die y-Achse individuell einzufärben.

3.8.4 Ellipsoid

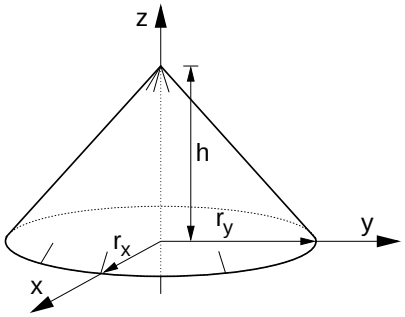
```
Primitive "NAME" {
  :
  Type = Sphere;
  RadiusX = 1.0;          ⟶  r_x
  RadiusY = 1.0;          ⟶  r_y
  RadiusZ = 1.0;          ⟶  r_z
  GridLong = 8;
  GridLati = 8;
}
```



Ein Ellipsoid mit den drei Radien r_x, r_y und r_z entlang der Achsen des Objektkoordinatensystems lässt sich mit `Type = Sphere;` erzeugen. Mit `GridLati` bzw. `GridLong` kann die Anzahl der Breiten- bzw. der Längenkreisunterteilungen, d.h. die Glattheit der Flächendarstellung ausgewählt werden.

3.8.5 Kegel

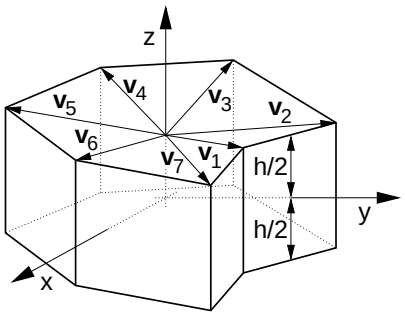
```
Primitive "NAME" {
  :
  Type = Cone;
  RadiusX = 1.0;          ⟶  r_x
  RadiusY = 1.0;          ⟶  r_y
  HeightZ = 1.0;          ⟶  h
  GridLong = 8;
}
```



Durch `Type = Cone;` wird ein Kegel der Höhe h mit einer Symmetrieachse entlang der z-Achse und einer elliptischen Grundfläche mit den Radien r_x und r_y in der xy-Ebene des Objektkoordinatensystems generiert. `GridLong` legt die Anzahl der Längenkreisunterteilungen, d.h. die Glattheit der Flächendarstellung fest.

3.8.6 Stanzpolynom

```
Primitive "NAME" {
  :
  Type = Extrude;
  HeightZ = 1.0;                          ⟶  h
  VertexList = { ... , ... , ... ,        ⟶  {v_i}
                 ... , ... , ... ,
                 :
                 ... , ... , ... };
```

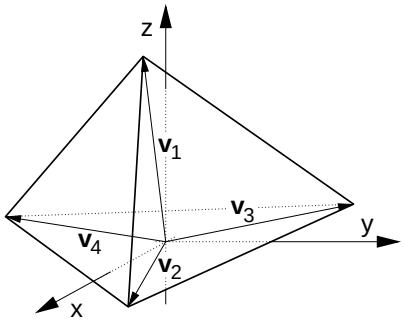


Ein Stanzpolynom, d.h. ein geometrisches Objekt mit der Grundfläche eines beliebigen, geschlossenen Polynoms und einer wählbaren Dicke h wird durch `Type = Extrude;` er-

zeugt. Dazu ist mit `VertexList` in geschwungenen Klammern `{}` eine Liste von Vertizes für die Eckpunkte des Polynoms anzugeben. Jeder Vertex umfasst drei Komponenten, so dass bei n Vertizes $3n$ durch Kommata getrennte Komponenten in der Liste angegeben werden müssen, die jeweils als Zahlentripel interpretiert werden. Der Benutzer muss selbstständig auf Konsistenz achten, d.h. es gibt keine Standard-Vertizes, so dass eine fehlende oder fehlerhafte `VertexList` zum Abbruch von AniDySim führt.
Die ersten beiden Komponenten jedes Vertex bezeichnen seine Position in der xy-Ebene des Objektkoordinatensystems und der dritte Wert wird momentan ignoriert. Es ist darauf zu achten, dass die Vertizes im mathematisch positiven Drehsinn um die z-Achse definiert werden, da ansonsten die Flächendarstellung fehlerhaft sein kann. Ihre Anzahl ist beliebig, allerdings sind weniger als drei nicht sinnvoll und sehr viele verlangsamen die Animation. Die räumliche Ausdehnung des Objekts geschieht in z-Richtung, wobei es sich um den gleichen Betrag oberhalb und unterhalb der xy-Ebene erstreckt.

3.8.7 Tetraeder

```
Primitive "NAME" {
  :
  Type = Tetraeder;
  Vertex1 = 0.0, 0.0, 0.0;        ⟶   v1
  Vertex2 = 1.0, 0.0, 0.0;        ⟶   v2
  Vertex3 = 0.0, 1.0, 0.0;        ⟶   v3
  Vertex4 = 0.0, 0.0, 1.0;        ⟶   v4
}
```



Mit `Type = Tetraeder;` wird durch die Angabe von vier Vertizes, d.h. von den Koordinaten der vier Eckpunkte im Objektkoordinatensystem ein Tetraeder erzeugt. Dabei ist darauf zu achten, dass die Vertizes $\boldsymbol{v}_2$, $\boldsymbol{v}_3$ und $\boldsymbol{v}_4$ im mathematisch positiven Drehsinn bzgl. $\boldsymbol{v}_1$ definiert werden, da es ansonsten zu einer fehlerhaften Flächendarstellung kommen kann. Der Boden des Tetraeders, d.h. das durch $\boldsymbol{v}_2$, $\boldsymbol{v}_3$ und $\boldsymbol{v}_4$ aufgespannte Dreieck wird nicht dargestellt.

4 Benutzerdefinierte Elemente

Bisher existieren lediglich rudimentäre Möglichkeiten, benutzerdefinierte Komponenten in DySim einzubinden. Allerdings stehen einem Programmierer nach einem Blick in die Quellen alle Eingriffsmöglichkeiten offen. Dazu sei bemerkt, dass es sehr sinnvoll ist, die Abschnitte 6 und 7 über die theoretischen Grundlagen und die Schalt-Funktionalität zu lesen, bevor benutzerdefinierte Routinen programmiert werden.

Um selbstprogrammierte Komponenten verwenden zu können, muss DySim selbstverständlich neu kompiliert werden. Dazu müssen die Dateien

```
makefile
usrfrc.cc
usrout.cc
usrrfkt.cc
```

in einem Verzeichnis in `<BASEDIR>/model/` vorliegen. Die drei Quelldateien enthalten die Standard-Funktionsrümpfe der Benutzerroutinen, die beliebig mit Quelltext ausgefüllt werden können.

Mit dem Ausführen von `make` in diesem Verzeichnis wird dann eine individuelle Version von DySim erzeugt, die anstelle der Standard-Funktionsrümpfe die Funktionsdefinitionen aus den drei Quelldateien enthält.

4.1 Benutzerdefinierte Kräfte

Zur Berechnung beliebiger benutzerdefinierter Kräfte und zu deren Anwendung auf das mechanische System steht eine Funktion zur Verfügung, die von DySim im Rahmen der internen Kraftberechnungen aufgerufen wird:

```
void UserForce( double t );  .
```

Die folgenden Daten werden in dieser Routine neben der aktuellen Simulationszeit `t` standardmäßig für die Berechnungen bereitgestellt:

`model` : Modellkonfigurationsdaten des Modells
- `MOo` : Vektor der 4×4 Lagematrizen der Körperkoordinatensysteme
- `WO` : Vektor der 4×4 Geschwindigkeitsmatrizen für jeden Freiheitsgrad
- `JO` : Vektor der 4×4 Trägheitsmatrizen der Teilkörper
- `PO` : Vektor der 4×4 Kraft-Drehmomentmatrizen der Teilkörper
- `FF` : Vektor der in Koordinatenrichtung aufgebrachten generalisierten Kräfte

`nrgb` : Anzahl der Teilkörper
`bdof` : Abbildung der Körperindizes auf die zugehörigen Indizes der Freiheitsgrade

Für das Verständnis der Modellkonfigurationsdaten im Objekt `model` sei auf die Klassendefinition im Quelltext hingewiesen.
Die 4×4 Matrizen liegen ausschließlich in Koordinaten des Inertialsystems vor und ihre Indizierung beginnt jeweils mit 1, d.h. die Lagematrix des ersten Körpers ist `MOo[1]`.
Der Vektor `WO` enthält in seinen Komponenten nicht nur die Geschwindigkeitsmatrizen der realen Körper, sondern auch die der masselosen Hilfskörper (vgl. Abschnitt 6), also gewissermaßen die Geschwindigkeitsmatrizen aller Freiheitsgrade. Um daraus nur diejenigen der realen Teilkörper zu extrahieren, muss die Index-Abbildung `bdof` verwendet werden. (`bdof[i]` enthält den Index des Freiheitsgrads, der direkt mit dem `i`-ten Körper verbunden ist.)
`PO` und `FF` werden in der Regel nur für die Funktionsaufrufe benötigt, mit denen die berechneten benutzerdefinierten Kräfte und Drehmomente auf die Körper aufgebracht werden. Dazu stehen DySim-Funktionen zur Verfügung, die sowohl die Kraftangriffspunkte, als auch die Kräfte und Drehmomente in Koordinaten des Inertialsystems erwarten.

Ein äußeres Drehmoment `tau` bzw. eine äußere Kraft `f` am Punkt `r` wird auf den Körper mit dem Index `rgbidx` durch die beiden Funktionen

```
void ApplyForce( double* f, double* r, int rgbidx, MAT4* PO );
void ApplyForce( double* tau, int rgbindx, MAT4* PO );
```

aufgebracht. Analog dienen die Funktionen

```
void ApplyForce( double* f, double* r0, int rgbidx0, double* r1, int rgbidx1,
                 MAT4* PO );
void ApplyForce( double* f, double* r0, int rgbidx0, double* r1, int rgbidx1,
                 double* tq, MAT4* PO );
void ApplyForce( double* tau, int rgbidx0, int rgbidx1, MAT4* PO );
```

dazu, zwischen den beiden Körpern mit den Indizes `rgbidx0` und `rgbidx1` eine Kraft an den entsprechenden Angriffspunkten und/oder ein Drehmoment anzubringen.

Mit der Funktion

```
void ApplyForce( double fq, int dofidx, double* FF );
```

kann eine generalisierte Kraft `fq` direkt auf den Freiheitsgrad mit dem Index `dofidx` ausgeübt werden.

4.2 Benutzerdefinierte Datenausgabe

Für eine benutzerdefinierte Datenausgabe sind drei Funktionen vorgesehen. Mit

```
void UserOutputCreate( const Model& model );
```

können einmalige Aktionen wie beispielsweise das Öffnen von Dateien oder das Bereitstellen von Speicher zu dem Zeitpunkt, wenn DySim die internen Dateien öffnet, durchgeführt werden. Als Funktionsparameter wird die gesamte Modellkonfiguration im Objekt `model` übergeben.

Für das Herausschreiben von Daten an jedem Datenausgabezeitpunkt dient die Funktion

```
void UserOutput( double t );
```

,

in der die gleichen Lage-, Geschwindigkeits- und Trägheitsmatrizen standardmäßig bereitgestellt werden wie in der benutzerdefinierten Kraftroutine.

Einmalige Aktionen wie beispielsweise das Schließen von Dateien oder die Freigabe von Speicher kurz vor Programmende können in der Funktion

```
void UserOutputCleanUp();
```

veranlasst werden.

4.3 Benutzerdefinierte diskrete Zustände

Um benutzerdefinierte diskrete Zustände mit der Schalt-Funktionalität (vgl. Abschnitt 7) des Integrators `derf` in DySim einführen und behandeln zu können, stehen ebenfalls drei Funktionen zur Verfügung.

Zunächst muss DySim mitgeteilt werden, wieviele Benutzerzustände benötigt werden. Dazu dient die Funktion

```
int UserNumberDState();
```

,

die lediglich die Anzahl der benötigten Zustände zurückliefern muss. Die Berechnung der Schaltfunktionswerte erfolgt in der Funktion

```
void UserRootFunc( const double t, const double *q, const double *qd,
                   const int nurfkt, const int *udstate, double *urval );
```

.

Dazu stehen die Simulationszeit `t`, die Zustandvariablen und deren Geschwindigkeiten, `q` und `qd` sowie die Anzahl und die Werte der Benutzerzustände, `nurfkt` und `udstate` zur Verfügung. Die berechneten Schaltfunktionswerte sind in dem Vektor `urval` abzulegen.

Der Schaltvorgang für die Benutzerzustände und die ggf. im Zusammenhang mit dem Schalten erforderlichen Berechnungen müssen in der Funktion

```
void UserSwitchDState( const double t, const double *q, const double *qd,
                       const int nurfkt, int *udstate, const int *nudstate );
```

festgelegt werden. Dazu stehen wiederum die Simulationszeit `t`, die Zustandvariablen und deren Geschwindigkeiten, `q` und `qd` sowie die Anzahl und die neuen Werte der Benutzerzustände, `nurfkt` und `nudstate` zur Verfügung. Jeder diskrete Zustand, dessen

entsprechender Wert in `nudstate` Null ist, muss unverändert bleiben. Von Null verschiedene Werte geben die jeweils neuen Zustandswerte an und müssen in `udstate` abgelegt werden.

5 Die Datenausgabe und die Animation

5.1 Die Simulationsdaten

Zur Auswertung der Simulationsergebnisse und für eine spätere Animation gibt DySim für jeden Ausgabezeitschritt (vgl. Abschnitt 3.3.1) ausgewählte Daten spaltenweise in Dateien aus. Dabei enthalten die Datenblöcke in der ersten Spalte stets die Zeit, und bei der Ausgabe von Vektoren oder Matrizen werden die Komponenten zeilenweise in die Spalten der Datei geschrieben:

$$t \ldots s \ldots \boldsymbol{r} \ldots D \ldots \;\rightarrow\; \begin{array}{c} \vdots \\ t_k \;\ldots s \ldots r_1\; r_2\; r_3 \ldots D_{11}\; D_{12}\; D_{13}\; D_{21}\; D_{22}\; D_{23}\; D_{31}\; D_{32}\; D_{33} \ldots \\ t_{k+1} \ldots s \ldots r_1\; r_2\; r_3 \ldots D_{11}\; D_{12}\; D_{13}\; D_{21}\; D_{22}\; D_{23}\; D_{31}\; D_{32}\; D_{33} \ldots \\ \vdots \end{array}$$

5.1.1 Die Protokolldatei

Die Protokolldatei `<MODELLNAME>.log` enthält neben der Start- und der Endzeit und den Anfangswerten der Integration die Meldungen über sämtliche DySim-internen Ereignisse, d.h. über die Zustandsänderungen des komplexen Kontaktkraftelements sowie über den Wechsel der Winkelkoordinaten bei 6-Freiheitsgrad- und Kugelgelenken. Weiterhin werden die Meldungen des Integrators protokolliert, die nicht zum Abbruch der Integration führen.

5.1.2 Die Ausgabe von Energie, Impuls und Drehimpuls

Die Ausgabe der Energie, der Schwerpunktslagen sowie des Impulses und des Drehimpulses erfolgt in die Datei `<MODELLNAME>.epl`. Der Datensatz umfasst jeweils 11 Werte für das Gesamtsystem und für jeden Körper. Somit berechnet sich die Spalte mit dem ersten Wert für den Körper n durch $(n-1)\,11+13$:

Spalte	Anzahl	Dateninhalt	Dimension
1	1	Zeit	1x1
	Gesamtsystem		
2	1	potentielle Gesamtenergie	1x1
3	1	kinetische Gesamtenergie	1x1
4–6	3	Gesamtschwerpunktslage	3x1
7–9	3	Gesamtimpuls	3x1
10–12	3	Gesamtdrehimpuls	3x1
	Körper 1		
13	1	potentielle Energie	1x1
14	1	kinetische Energie	1x1

Spalte	Anzahl	Dateninhalt	Dimension
15–17	3	Schwerpunktslage	3x1
18–20	3	Impuls	3x1
21–23	3	Drehimpuls	3x1
	Körper 2		
24	1	potentielle Energie	1x1
⋮	⋮	⋮	⋮
	Körper n		
$11n+2$	1	potentielle Energie	1x1
⋮	⋮	⋮	⋮

5.1.3 Die Ausgabe der Kräfte

Die Ausgabe der Kräfte erfolgt in die Datei `<MODELLNAME>.frc`. Auf dem derzeitigen Stand werden lediglich die Daten des komplexen Kontaktelements (vgl. Abschnitt 3.7.5) ausgegeben. Der Datensatz umfasst jeweils 16 Werte für jedes Kontaktelement. Somit berechnet sich die Spalte des ersten Wertes für das Element n durch $(n-1)\,16+2$:

Spalte	Anzahl	Dateninhalt	Dimension
1	1	Zeit	1x1
	Kontakt 1		
2–4	3	Auslenkung $\boldsymbol{r}$	3x1
5–7	3	Relativgeschwindigkeit $\boldsymbol{v}$	3x1
8	1	Kontaktindikator	1x1
9	1	Haftindikator	1x1
10, 11	2	Haftposition $\bar{\boldsymbol{r}}_{\parallel}$	2x1
12–14	3	Kontaktkraft $\boldsymbol{f}$	3x1
15–17	3	Kontaktdrehmoment $\boldsymbol{\tau}$	3x1
	Kontakt 2		
18–20	3	Auslenkung $\boldsymbol{r}$	3x1
⋮	⋮	⋮	⋮
	Kontakt n		
$16n-14$	3	Auslenkung $\boldsymbol{r}$	3x1
⋮	⋮	⋮	⋮

5.1.4 Die Ausgabe von Lagen, Geschwindigkeiten und Beschleunigungen

Die Ausgabe der Positionen, Orientierungen, Geschwindigkeiten und Beschleunigungen der Körper, d.h. ihrer Körperkoordinatensysteme, erfolgt in die Datei `<MODELLNAME>.pva`. Der Datensatz umfasst jeweils 24 Werte für jeden Körper. Somit berechnet sich die Spalte mit dem ersten Wert des Körpers n durch $(n-1)\,24+2$:

Spalte	Anzahl	Dateninhalt	Dimension
1	1	Zeit	1x1
	Körper 1		
2–4	3	Position	3x1

Spalte	Anzahl	Dateninhalt	Dimension
5–13	9	Orientierung	3x3
14–16	3	Geschwindigkeit	3x1
17–19	3	Winkelgeschwindigkeit	3x1
20–22	3	Beschleunigung	3x1
23–25	3	Winkelbeschleunigung	3x1
		Körper 2	
26–28	3	Position	3x1
⋮	⋮	⋮	⋮
		Körper n	
$24n-22$	3	Position	3x1
⋮	⋮	⋮	⋮

5.2 Die Animation mit AniDySim

Nach der Simulation kann der Bewegungsablauf des Modells mit AniDySim animiert werden. Die Interaktion mit dem Programm erfolgt über die Tastatur und die Maus.

5.2.1 Die Animationsdatei

Die Daten, die AniDySim für die Animation benötigt, legt DySim während der Simulation in der Animationsdatei `<MODELLNAME>.ani` ab. Diese Datei enthält zunächst einen Dateikopf, der mit dem Kommentarzeichen `#` markiert ist, und der von AniDySim ausgewertet wird.

```
# DySimAniFileVersion: 0.5
# ModelName: <MODELLNAME>
# NumberOfBodies: <n>
# NamesOfBodies: <name_1 name_2 ... name_n>
```

Der Kopf umfasst jeweils mit Schlüsselwörtern gekennzeichnet die Versionnummer der Animationsdatei, den Modellnamen des Modells, mit dem die Daten erzeugt wurden, sowie die Anzahl und die Namen der Körper in der Reihenfolge, wie sie in der Datei vorliegen. Mit Hilfe dieser Körpernamen sucht AniDySim in der Konfigurationsdatei `<MODELLNAME>.dys` die entsprechenden Grafikprimitive zur Visualisierung der Körper.

Im Datenblock der Datei folgen dann die homogenen Lagematrizen der Körper $\boldsymbol{R}$ (vgl. Abschnitt 6), die die Positionen und die Orientierungen der Körperkoordinatensysteme beschreiben:

Spalte	Anzahl	Dateninhalt	Dimension
1	1	Zeit	1x1
		Körper 1	
2–17	16	Lage $\boldsymbol{R}$	4x4
		Körper 2	
18–33	16	Lage $\boldsymbol{R}$	4x4
⋮	⋮	⋮	⋮
		Körper n	
$16n-14$	16	Lage $\boldsymbol{R}$	4x4

5.2.2 Die Bedienung von AniDySim

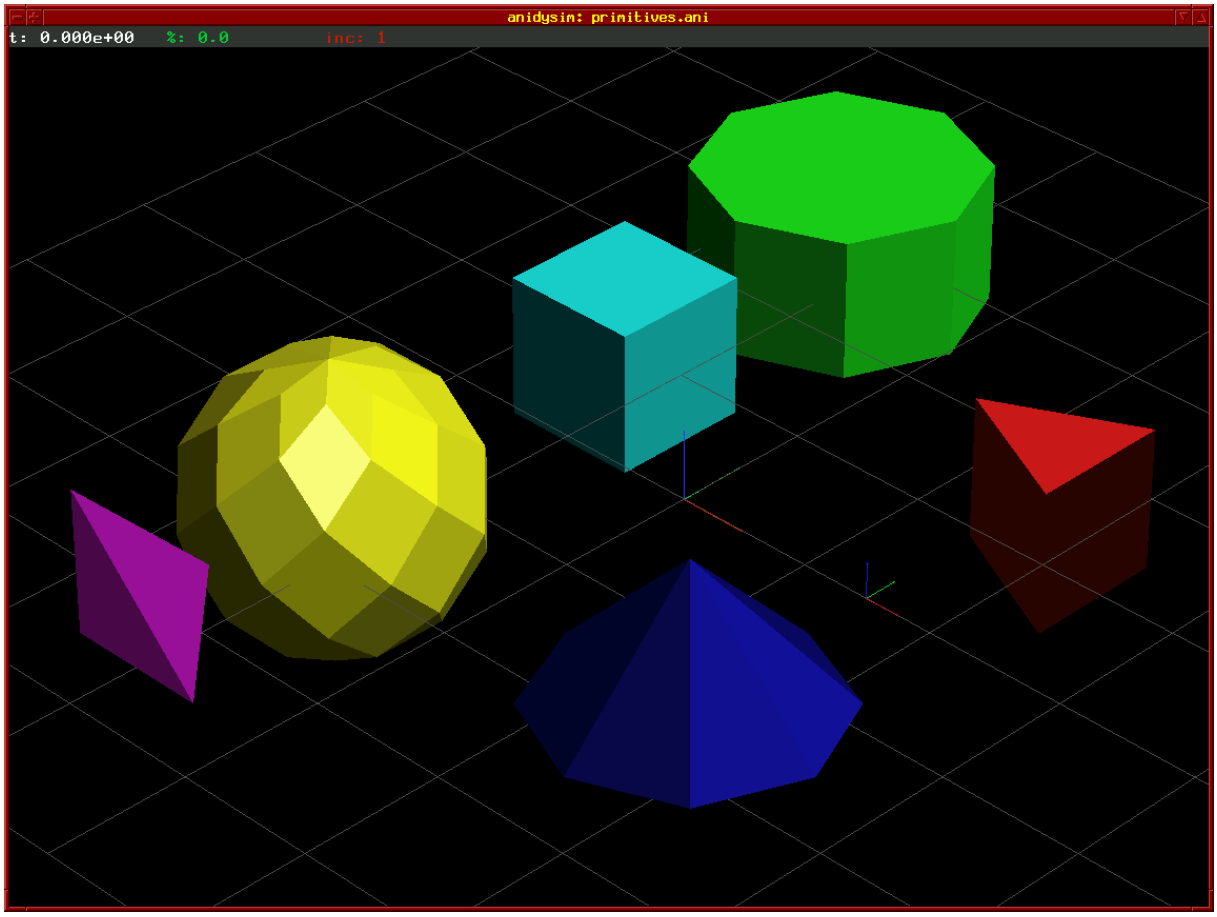


Abbildung 4: Das Animationsfenster von AniDySim mit allen verfügbaren Grafikprimitiven. Am oberen Rand werden neben der momentanen Animationszeit und der momentanen Animationsgeschwindigkeit die Einstellungen von AniDySim angezeigt.

Die Abbildung 4 zeigt das Animationsfenster von AniDySim. Es sind Beispiele für alle verfügbaren Grafikprimitive sowie das Inertialsystem (großes Dreibein) mit dem Gitter in der Ebene $z = 0$ dargestellt.

Der graue Streifen am oberen Rand des Fensters dient als Statusleiste, in der ausgewählte Daten sowie Einstellungen von AniDySim angezeigt werden: Ganz links ist als erstes die momentane Animationszeit in weiß dargestellt, d.h. die Simulationszeit der gerade sichtbaren Modellkonfiguration. Daneben ist die momentane Animationsgeschwindigkeit in Prozent der Echtzeit in grün wiedergegeben. Dieser Wert hängt von der Leistungsfähigkeit und der Auslastung des Rechners sowie vom Darstellungsmodus, der Größe des Fensters und der Anzahl der darzustellenden Objekte ab. Die Animationsgeschwindigkeit kann interaktiv vergrößert werden, indem nur jeder k-te Zeitschritt dargestellt wird. Der Wert von k ist rechts neben der Animationsgeschwindigkeit in rot angegeben. (Es besteht momentan keine direkte Möglichkeit, die Animation zu verlangsamen. Sie kann lediglich indirekt durch die Vergrößerung des Animationsfensters oder die Auswahl eines anderen Darstellungsmodus erreicht werden.) Als letzte Information zeigt die Statusleiste rechts neben der Animationsschrittweite k durch den Schriftzug `loop` in gelb an, wenn der Endlos-Animationsmodus aktiviert ist (in Abbildung 4 nicht sichbar).

Die Bedienung von AniDySim kann – bis auf Texteingaben – vollständig mit der Maus erfolgen, wobei für jedes Kommando auch eine Tastaturbelegung existiert.

Durch Betätigen der rechten Maustaste im Animationsfenster (nicht im Textfenster der Statusleiste) öffnet sich ein Menüfenster mit verschiedenen Auswahlpunkten, hinter denen sich ggf. Untermenüs verbergen. Dabei dient das Untermenü *Information* als Hilfe, die alle Tastaturbelegungen und Mausfunktionen kurz beschreibt. Die folgenden Interaktionen sind möglich:

Animation steuern	
ESC	Beenden
Maus R	Animationsfenster: Menü öffnen
g/h	Animation starten/anhalten
}/{	Bildweise vorwärts/rückwärts spulen
>/<	Animation beschleunigen/verlangsamen
Maus R/L	Statuszeile: Animation beschleunigen/verlangsamen
0	Animation zurückspulen
u	Aktuellen Datensatz nochmals einlesen
Ansicht wählen	
F5	YZ-Ansicht auswählen
F6	XZ-Ansicht auswählen
F7	XY-Ansicht auswählen
i	Standard-Ansicht auswählen
Rotieren, Verschieben, Skalieren	
SHIFT Maus L: $\updownarrow\leftrightarrow$	Animationsfenster: Rotation der Ansicht
Maus L: $\updownarrow\leftrightarrow$	Animationsfenster: Translation der Ansicht
Maus M: $\updownarrow$	Animationsfenster: Zoom
$\leftarrow/\rightarrow$	Rotation der Ansicht um z des Inertialsystems
$\uparrow/\downarrow$	Rotation der Ansicht um die Fensterhorizontale
x/X	Translation der Ansicht nach rechts/links
z/Z	Zoom nah/fern
y/Y	Translation der Ansicht nach oben/unten
a/A	Achslänge des Inertialsystems vergrößern/verkleinern
Tastatureingabe	
F1	Neue Animationsdaten lesen
F2	Dimension des xy-Ebenen-Gitters verändern
F3	Körper für den Folgemodus auswählen
F4	Basisname für TIFF-Bildausgabe wählen
Schalter	
f	Folgemodus an/aus
l	Endlosmodus an/aus
q	Visualisierung der xy-Ebene an/aus
r	TIFF-Bildausgabe an/aus
s	Flächenglättung an/aus
w	Gitterflächendarstellung an/aus

Die Tastatureingaben

Die Tastatureingaben müssen in derjenigen Konsole erfolgen, von der aus AniDySim gestartet wurde.
Mit F1 kann der Dateiname eines anderen, *jedoch mit derselben Modellkonfiguration erzeugten* Animationsdatensatzes angegeben werden, der dann in der Folge visualisiert wird. Ein mit der Konfiguration nicht konsistenter Datensatz kann zur Beendigung von AniDySim führen.
Mit F2 können die Lage und die Größe des Gitters in der xy-Ebene des Inertialsystems verändert werden. Dazu sind durch Kommata getrennt die Dimensionen des Gitters in negativer und positiver x-Richtung sowie in negativer und positiver y-Richtung anzugeben.
Mit F3 kann der Index des Folgekörpers angegeben werden, dem die „Kamera“ bei der Animation in xy-Richtung folgen soll. (Der Folgemodus kann dann mit „f“ deaktiviert und wieder aktiviert werden.)
Mit F4 wird der Basisname für die Dateinamen abgefragt, unter denen die TIFF-Bilder der Animationsschritte bei aktivierter TIFF-Ausgabe abgelegt werden sollen.

Die TIFF-Ausgabe

Es ist möglich, die einzelnen Animationsschritte als TIFF-Dateien herauszuschreiben, um die Bilder anderweitig zu verarbeiten. Um eine Animationssequenz aufzuzeichnen, muss zunächst ein Basisname für die Bilddateien angegeben werden (mit F4 wird automatisch Packbit-Kompression eingestellt, über das Menü kann zwischen keiner Kompression, LZW- oder Packbit-Kompression gewählt werden), und danach muss die TIFF-Ausgabe aktiviert werden. Bei nachfolgender Animation erfolgt neben der Darstellung im Animationsfenster auch die Ausgabe in Bilddateien mit den Namen `<Basisname>####.tif`. Dabei bezeichnet `####` die vierstellige, ggf. vorne mit Nullen aufgefüllte Bildnummer des jeweiligen Animationsschritts. Die Nummerierung beginnt bei Null, d.h. eine Animationssequenz kann maximal 10000 Bilder umfassen. Sobald die TIFF-Ausgabe nach der Aufzeichnung deaktiviert wird, verliert der Basisname seine Gültigkeit.

6 Die zugrundeliegende Theorie

Die Erzeugung der Bewegungsgleichungen zur dynamischen Analyse eines mechanischen Systems in DySim basiert auf einem ableitungsfreien Verfahren zur Formulierung der Lagrangeschen Gleichungen zweiter Art in generalisierten Koordinaten:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i \ , \quad i = 1, 2, \ldots, n \quad . \tag{1}$$

Bei diesem Verfahren handelt es sich um den Ansatz mit homogenen Matrizen, wie er von Legnani et al. [8, 9] vorgestellt und detailliert beschrieben wird.

Die Grundlage bildet die Beschreibung von Punkten im Raum durch jeweils vier *homogene Koordinaten*, $\boldsymbol{x} = [x_1, x_2, x_3, w]^T$, wobei die homogene Koordinate w in der Regel den Wert 1 annimmt, für Punkte im Unendlichen, Geschwindigkeiten, Beschleunigungen oder daraus abgeleitete Größen jedoch den Wert 0.

Die Position $\boldsymbol{t}$ und die Orientierung $\boldsymbol{D}$ eines Starrkörpers im Inertialsystem werden zu einer einzigen Größe, der 4×4 *Lage*-Matrix, zusammengefasst,

$$\boldsymbol{R} = \begin{pmatrix} \boldsymbol{D} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{pmatrix} \quad ,$$

so dass die affine Transformation der körperfesten Koordinaten $\boldsymbol{x}' = [\boldsymbol{s}'^T, 1]^T$ eines Punktes ins Inertialsystem durch ein Matrix-Vektor-Produkt beschrieben werden kann:

$$\boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{x}' \quad \Leftrightarrow \quad \boldsymbol{r} = \boldsymbol{t} + \boldsymbol{D}\,\boldsymbol{s}' \quad .$$

Die Lage des Inertialsystems relativ zum Körpersystem ist entsprechend durch $\boldsymbol{R}^{-1}$ definiert. Ebenso können 4×4 Geschwindigkeits- und Beschleunigungsmatrizen aus der antisymmetrischen Matrix $\tilde{\boldsymbol{\omega}}$ der Winkelgeschwindigkeit des Körpers und deren Zeitableitung sowie der Geschwindigkeit $\boldsymbol{v}_0$ und der Beschleunigung $\boldsymbol{a}_0$ seines körperfesten Koordinatensystems aufgestellt werden:

$$\boldsymbol{W} = \dot{\boldsymbol{R}}\,\boldsymbol{R}^{-1} = \begin{pmatrix} \tilde{\boldsymbol{\omega}} & \boldsymbol{v}_0 \\ \boldsymbol{0} & 0 \end{pmatrix} , \quad \boldsymbol{H} = \dot{\boldsymbol{W}} + \boldsymbol{W}^2 = \begin{pmatrix} \dot{\tilde{\boldsymbol{\omega}}} + \tilde{\boldsymbol{\omega}}^2 & \boldsymbol{a}_0 \\ \boldsymbol{0} & 0 \end{pmatrix} \quad .$$

Wiederum ergeben Matrix-Vektor-Multiplikationen die Geschwindigkeit sowie die Beschleunigung eines körperfesten Punktes $\boldsymbol{x} = [\boldsymbol{s}^T, 1]^T$ im Inertialsystem:

$$\begin{aligned} \dot{\boldsymbol{x}} = \boldsymbol{W}\,\boldsymbol{x} \quad &\Leftrightarrow \quad \boldsymbol{v} = \boldsymbol{v}_0 + \boldsymbol{\omega} \times \boldsymbol{s} \\ \ddot{\boldsymbol{x}} = \boldsymbol{H}\,\boldsymbol{x} \quad &\Leftrightarrow \quad \boldsymbol{a} = \boldsymbol{a}_0 + \dot{\boldsymbol{\omega}} \times \boldsymbol{s} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{s}) \quad . \end{aligned}$$

Die Matrizen $\boldsymbol{R}$, $\boldsymbol{W}$ und $\boldsymbol{H}$ sind hier die kartesischen Darstellungen von Tensoren in Koordinaten des Inertialsystems, und ihre Komponenten werden durch $\boldsymbol{W}' = \boldsymbol{R}^{-1}\boldsymbol{W}\,\boldsymbol{R}$ und $\boldsymbol{H}' = \boldsymbol{R}^{-1}\boldsymbol{H}\,\boldsymbol{R}$ in körperfeste Koordinaten überführt.

Die relative Kinematik dreier Körper i, j und k ist durch die Tensorgleichungen

$$\begin{aligned} \boldsymbol{R}_{i,k} &= \boldsymbol{R}_{i,j}\boldsymbol{R}_{j,k} \\ \boldsymbol{W}_{i,k} &= \boldsymbol{W}_{i,j} + \boldsymbol{W}_{j,k} \\ \boldsymbol{H}_{i,k} &= \boldsymbol{H}_{i,j} + \boldsymbol{H}_{j,k} + 2\boldsymbol{W}_{i,j}\boldsymbol{W}_{j,k} \end{aligned}$$

bestimmt, wobei der zweite Index jeweils angibt, welches System durch die entsprechende Größe relativ zum System des ersten Indexes beschrieben wird: $\boldsymbol{W}_{i,k}$ beschreibt demnach die Geschwindigkeit des Körpers k gegenüber dem Körper i.

Für die dynamischen Größen der Mechanik werden ebenfalls 4×4 Matrizen eingeführt. Die Trägheitseigenschaften eines Körpers der Masse m, der Schwerpunktlage $\boldsymbol{r}_{cg}$ und der Massendichte ρ werden dazu in einer Trägheitsmatrix zusammengefasst,

$$\boldsymbol{J} = \begin{pmatrix} \int \boldsymbol{x}\,\boldsymbol{x}^T \rho\, dV & m\,\boldsymbol{r}_{cg} \\ m\,\boldsymbol{r}_{cg}^T & m \end{pmatrix} \quad ,$$

die auf ihn wirkende Kraft $\boldsymbol{f}$ und das Drehmoment $\boldsymbol{n}$ in einer Kraft-Drehmoment-Matrix $\boldsymbol{\Phi}$ und sein Impuls $\boldsymbol{p}$ sowie der Drehimpuls $\boldsymbol{l}$ in der Impulsmatrix $\boldsymbol{\Gamma}$:

$$\boldsymbol{\Phi} = \begin{pmatrix} \tilde{\boldsymbol{n}} & \boldsymbol{f} \\ -\boldsymbol{f}^T & 0 \end{pmatrix} , \quad \boldsymbol{\Gamma} = \begin{pmatrix} \tilde{\boldsymbol{l}} & \boldsymbol{p} \\ -\boldsymbol{p}^T & 0 \end{pmatrix} \quad .$$

Wie die kinematischen Matrizen $\boldsymbol{R}$, $\boldsymbol{W}$ und $\boldsymbol{H}$ sind auch die dynamischen Größen Tensoren. Sie transformieren sich wie $\boldsymbol{J} = \boldsymbol{R}\,\boldsymbol{J}'\boldsymbol{R}^T$.

Unter diesen Voraussetzungen und mit der Erdbeschleunigungsmatrix $\boldsymbol{H}_g$, deren einzige von Null verschiedenen Einträge die Komponenten der Linearbeschleunigung $\boldsymbol{a}_0 = \boldsymbol{g}$ sind, ergeben sich somit die kinetische sowie die potentielle Energie eines Starrkörpers,

$$T = \frac{1}{2}\,\mathrm{spur}(\boldsymbol{W}\,\boldsymbol{J}\,\boldsymbol{W}^T) , \quad V = -\mathrm{spur}(\boldsymbol{H}_g\,\boldsymbol{J}) \quad , \tag{2}$$

und in kompakter Form die NEWTON-EULER-Gleichungen zur Beschreibung seiner Dynamik:

$$\boldsymbol{\Phi} = \boldsymbol{H}\,\boldsymbol{J} - \boldsymbol{J}\,\boldsymbol{H}^T \quad .$$

Um die LAGRANGE-Funktion – basierend auf den Ausdrücken für die Energien (2) – in generalisierten Koordinaten aufzustellen, wird jeder Teilkörper des mechanischen Systems sukzessive mit Freiheitsgraden relativ zu seinem Mutterkörper oder zum Inertialsystem versehen. Dieses Vorgehen lässt sich allerdings nur auf Systeme anwenden, die keine geschlossenen kinematischen Ketten enthalten. Zur Beschreibung geschlossener Ketten müssten nachträglich Zwangsbedingungen eingeführt werden, die die freigeschalteten Freiheitsgrade einer zunächst offenen Kette wieder entsprechend einschränken, um den Kettenschluss zu gewährleisten. Dadurch würde sich ein differential-algebraisches Gleichungssystem ergeben, bei dessen Lösung stets auf die Einhaltung der Schlussbedingungen geachtet werden muss. DySim unterstützt geschlossenen kinematische Ketten bisher nicht, so dass auf ihre Behandlung nicht weiter eingegangen wird.

Da durch die Geschwindigkeitsmatrix $\boldsymbol{W}$ eine momentane Schraubenachse definiert wird,

$$\boldsymbol{L} = \frac{\boldsymbol{W}}{|\boldsymbol{\omega}|} \quad \text{für} \quad |\boldsymbol{\omega}| \neq 0 \quad \text{sonst} \quad \boldsymbol{L} = \frac{\boldsymbol{W}}{|\boldsymbol{v}_0|} \quad ,$$

ist es möglich, zwei Basisfreiheitsgrade in Form dieser momentanen Schraubenachsen festzulegen. Der eine Freiheitsgrad, beschrieben durch $\boldsymbol{L}_s$, ist die Schraubenbewegung um die Achse $\boldsymbol{e}$, die durch den Punkt $\boldsymbol{r}_e$ verläuft, mit einem Vortrieb p. Der Spezialfall für $p = 0$ ist die reine Rotation um $\boldsymbol{e}$. Der zweite ist ein reiner Translationsfreiheitsgrad entlang der Achse und wird durch $\boldsymbol{L}_t$ beschrieben:

$$\boldsymbol{L}_s = \begin{pmatrix} \tilde{\boldsymbol{e}} & -\tilde{\boldsymbol{e}}\,\boldsymbol{r}_e + p\,\boldsymbol{e} \\ \boldsymbol{0} & 0 \end{pmatrix} , \quad \boldsymbol{L}_t = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{e} \\ \boldsymbol{0} & 0 \end{pmatrix} \quad .$$

Für jeden Freiheitsgrad wird nun jeweils eine generalisierte Koordinate q eingeführt, so dass die Geschwindigkeitsmatrix eines Körpers mit einem Freiheitsgrad als Funktion der generalisierten Geschwindigkeit geschrieben werden kann,

$$\boldsymbol{W}_s(\dot{q}) = \boldsymbol{L}_s\,\dot{q} \quad \text{bzw.} \quad \boldsymbol{W}_t(\dot{q}) = \boldsymbol{L}_t\,\dot{q} \quad ,$$

aus der sich durch Integration die Komponenten der Lage-Matrix als Funktion von q ergeben:

$$\begin{aligned} \boldsymbol{D}_s(q) &= \boldsymbol{1} + \tilde{\boldsymbol{e}}\,\sin q + \tilde{\boldsymbol{e}}^2(1 - \cos q) \\ \boldsymbol{t}_s(q) &= (\boldsymbol{1} - \boldsymbol{D}_s(q))\,\boldsymbol{r}_e + p\,\boldsymbol{e}\,q \end{aligned} \quad \text{bzw.} \quad \begin{aligned} \boldsymbol{D}_t(q) &= \boldsymbol{1} \\ \boldsymbol{t}_t(q) &= \boldsymbol{e}\,q \end{aligned} \quad .$$

Kinematische Ketten werden nun, beginnend beim Inertialsystem, baumartig aus einzelnen Körpern zusammengesetzt, die alle jeweils über einen dieser Basisfreiheitsgrade relativ zu ihrem Mutterkörper verfügen. Um daher Starrkörper mit mehr als einem Freiheitsgrad zu versehen, müssen masselose Hilfskörper mit $\boldsymbol{J}_{\text{hilf}} = \boldsymbol{0}$ entsprechend dazwischengeschaltet werden. Zur Beschreibung beispielsweise eines freien Starrkörpers, sind daher zusätzlich fünf Hilfskörper erforderlich, wobei drei dieser sechs über einen Rotations- und drei über einen Translationsfreiheitsgrad mit jeweils linear unabhängigen Rotations- bzw. Translationsachsen verfügen müssen. In diesem Sinne ist der Begriff „Körper“ gleichbedeutend mit „Koordinatensystem“ und „Freiheitsgrad“. Der Index 0 bezeichnet dann das Inertialsystem.

Eine Zwischenbemerkung zu den Rotationsfreiheitsgraden: Bei der Verwendung von generalisierten Koordinaten ist man zur Beschreibung der Körperorientierung – beispielsweise in Gelenken mit mehreren Rotationsfreiheitsgraden – prinzipiell auf Winkelkoordinaten angewiesen, d.h. auf die Drehwinkel aufeinander folgender Rotationen um unterschiedliche Achsen. Da diese Achsen im Laufe der Bewegung linear abhängig werden können, führt dies zu Koordinatensingularitäten, wenn nicht zusätzlich entsprechende Vorkehrungen getroffen werden, um in jedem Moment eine eindeutige Systembeschreibung zu gewährleisten. In DySim wird dieses Problem dadurch gelöst, dass eine nahende Singularität, deren Lage für jeden Satz Winkelkoordinaten a priori bekannt ist, frühzeitig detektiert und dann auf eine andere Rotationsreihenfolge umgeschaltet wird. Die generalisierten Koordinaten q_i erhalten durch diese Vorgehensweise zwar Unstetigkeiten – die entscheidenden kinematischen und dynamischen Matrizen, vor allem die Lage-, die Geschwindigkeits- und die Beschleunigungsmatrizen, bleiben davon jedoch unbeeinflusst. Dieses Verfahren hat sich bei sämtlichen mit DySim durchgeführten Simulationen als sehr zuverlässig erwiesen, wobei es ausreichend ist, lediglich zwischen EULER- und KARDAN-Winkeln hin- und herzuschalten, um die Singularitäten wirkungsvoll zu umgehen.

Mit den Ausdrücken für die Energien (2) kann nun die LAGRANGE-Funktion für eine kinematische Kette, bestehend aus n Körpern mit jeweils einem Freiheitsgrad, aufgestellt werden,

$$L = \sum_{k=1}^{n} \frac{1}{2} \,\mathrm{spur}(\boldsymbol{W}_{0,k}\, \boldsymbol{J}_k\, \boldsymbol{W}_{0,k}^T) + \mathrm{spur}(\boldsymbol{H}_g\, \boldsymbol{J}_k) \quad ,$$

wobei sich die Lage- bzw. die Winkelgeschwindigkeitsmatrix jedes Körpers aufgrund der beschriebenen Baumstruktur des Systems als Funktion der generalisierten Koordinaten und deren Geschwindigkeiten ausdrücken lassen:

$$\boldsymbol{R}_{0,k} = \prod_{i=1}^{n_k} \boldsymbol{R}_{k_{i-1},k_i}(q_{k_i}) \;, \quad \boldsymbol{W}_{0,k} = \sum_{i=1}^{n_k} \boldsymbol{L}_{k_{i-1},k_i}\, \dot{q}_{k_i} \quad .$$

Dabei ist n_k die Anzahl aller Freiheitsgrade von denen die Lage des Körpers k im Inertialsystem abhängt, und die Folge $\{k_1, k_2, \ldots, k_{n_k}\}$ ist die Reihe ihrer Indizes, beginnend beim Inertialsystem.

Auf diese Weise ist es möglich, alle Ableitungen, die zur Aufstellung der LAGRANGEschen Bewegungsgleichungen (1) benötigt werden, durch Summen und Produkte der Matrizen $\boldsymbol{R}$, $\boldsymbol{L}$, $\boldsymbol{W}$ und $\boldsymbol{J}$ sowie durch die generalisierten Koordinaten q_i, deren Geschwindigkeiten $\dot{q}_i$ und der Beschleunigungen $\ddot{q}_i$ darzustellen. Nach geeigneter Sortierung der so ermittelten Bewegungsgleichungen erhält man folgendes Gleichungssystem in generalisierten Koordinaten $\boldsymbol{q} = [q_1, q_2, \ldots, q_n]^T$:

$$\boldsymbol{M}\, \ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \quad . \tag{3}$$

Bezeichnet man mit k_p den Index des Mutterkörpers von Körper k, und definiert man für jeden Körper h einen Indikator $\boldsymbol{\delta}^{(h)} = [\delta_1^{(h)}, \delta_2^{(h)}, \ldots, \delta_n^{(h)}]$ mit

$$\delta_i^{(h)} = \begin{cases} 1 & \text{wenn der Körper } h \text{ vom Freiheitsgrad } i \text{ abhängt} \\ 0 & \text{sonst} \end{cases} \quad ,$$

so ergibt sich für die Komponenten der Massenmatrix

$$M_{ik} = \mathrm{spur} \left[\sum_{h=1}^{n} \delta_i^{(h)} \delta_k^{(h)} \boldsymbol{L}_{k_p,k} \boldsymbol{J}_h\, \boldsymbol{L}_{i_p,i}^T \right] \quad .$$

Mit

$$\bar{\boldsymbol{H}}_{0,k} = \sum_{i=1}^{n_k} \left(\boldsymbol{W}_{0,k_{i-1}} \boldsymbol{W}_{k_{i-1},k_i} - \boldsymbol{W}_{k_{i-1},k_i} \boldsymbol{W}_{0,k_{i-1}} \right) + \boldsymbol{W}_{0,k}^2$$

ergeben sich die Komponenten des Vektors $\boldsymbol{c}$ der CORIOLIS-, Zentrifugal- und Gravitationsterme:

$$c_i = \text{spur} \left[\sum_{k=1}^{n} \delta_i^{(k)} (\bar{\boldsymbol{H}}_{0,k} - \boldsymbol{H}_g) \boldsymbol{J}_k \boldsymbol{L}_{i_p,i}^T \right] \quad .$$

Die Komponenten der auf das System wirkenden generalisierten inneren und äußeren Kräfte $\boldsymbol{f}$ lauten:

$$f_i = \bar{f}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) + \sum_{k=1}^{n} \delta_i^{(k)} \boldsymbol{\Phi}_k \otimes \boldsymbol{L}_{i_p,i} \quad .$$

Dabei kann mit dem Term $\bar{f}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)$ eine generalisierte Kraft angegeben werden, die genau in Richtung der Koordinatenachse i zwischen denjenigen Körpern wirkt, die durch diesen Freiheitsgrad miteinander verbunden sind. Der Operator $\otimes$ bezeichnet ein Pseudo-Skalarprodukt, mit dem jeweils das Gesamt-Kraft-Drehmoment $\boldsymbol{\Phi}_k$, das von innen und außen auf den Körper k wirkt, auf die Achse der i-ten generalisierten Koordinate projiziert wird: $\boldsymbol{A} \otimes \boldsymbol{B} = A_{32}B_{32} + A_{13}B_{13} + A_{21}B_{21} + A_{14}B_{14} + A_{24}B_{24} + A_{34}B_{34}$.

Das Gleichungssystem (3) ist bei gleichem mechanischem Modell sehr viel kleiner als eines in nicht generalisierten Koordinaten. Allerdings ist die Massenmatrix $\boldsymbol{M}$ in (3) nicht dünn besetzt, weshalb die optimierten numerischen Verfahren zur Invertierung von solchen Matrizen hier nicht einsetzbar sind. Es kann jedoch ausgenutzt werden, dass $\boldsymbol{M}$ symmetrisch und, solange die Achsen der generalisierten Koordinaten – vor allem die der Rotationen – linear unabhängig voneinander sind, positiv definit ist. Das Gleichungssystem (3) kann daher mit Standardverfahren nach den Beschleunigungen aufgelöst und integriert werden.

7 Die Schalt-Funktionalität von DySim

Da die Integratoren für gewöhnliche Differentialgleichungssysteme $\dot{\boldsymbol{q}} = \boldsymbol{f}(t, \boldsymbol{q})$ in der Regel stetig differenzierbare Zustandsvariablen voraussetzen, ergeben sich Probleme bei der Lösung realistischer Modelle, bei denen Unstetigkeiten beispielsweise aufgrund von plötzlich auftretenden Kontaktkräften oder anderen diskreten Schaltvorgängen zwischen *binären* Zuständen keine Seltenheit sind. Fortschrittliche Integratoren mit Schrittweitensteuerung und variabler Ordnung finden und überwinden solche Unstetigkeiten zwar durch die geeignete Verkleinerung der Schrittweiten, allerdings auf Kosten der Integrationsgeschwindigkeit und der Genauigkeit.

Durch eine geeignete, programmiertechnische Formulierung ist es möglich, die Lösungen in stückweise stetig differenzierbare Abschnitte zu unterteilen, und bei jedem Übergang von einem zum anderen Abschnitt die Integration neu zu starten, um so die Genauigkeit und Zuverlässigkeit der Integration zu erhöhen. Als Anfangsbedingung für den Neustart wird jeweils der Zustand am Ende des letzten Abschnitts verwendet, wobei die weitere Integration unter den unstetig veränderten Bedingungen des neuen Abschnitts erfolgt.

Verfahrensbedingt verfügt der `de` von SHAMPINE/GORDON [13] intern über die Fähigkeit, die Zustandsvariablen zu interpolieren, weshalb aus ihm durch leichte Modifikation der Integrator `derf` mit einer Schalt-Funktionalität für DySim abgeleitet wurde.

Um die Schaltereignisse detektieren zu können, wird für jeden diskreten, binären Zustand eine stetige Funktion der Zustandsvariablen definiert, die im Moment des Schaltens eine Nullstelle besitzt. Ein positiver Wert einer Schaltfunktion kennzeichnet dann den Wert +1 und ein negativer Wert den Wert -1 des zugehörigen binären Zustands. Sobald eine Nullstelle während der Integration detektiert wird, kann der entsprechende Zustand geschaltet und mit der Integration ab dieser Stelle neu begonnen werden.

Um diese Funktionalität in `derf` zu realisieren, musste die ursprüngliche Parameterklammer des `de` erweitert werden, um dem Integrator zusätzlich die Anzahl `nrfkt` der Schaltfunktionen, die Funktion `rfkt` zur Berechnung der Schaltfunktionswerte, den Vektor `dstate` der Werte der diskreten Zustände sowie die Funktion `swdstate` zum Schalten der Zustände übergeben zu können:

```
      SUBROUTINE DERF(F,NEQN,Y,T,TOUT,RELERR,ABSERR,IFLAG,
     &     nrfkt,rfkt,dstate,swdstate)  .
```

In der Funktion

```
void rfkt(double* t, double* q, double* qd, int* dstate, double* rval);
```

stehen die Zeit `t`, die Zustandsvariablen und deren Geschwindigkeiten, `q` und `qd`, und die Werte `dstate` der diskreten Zustände zur Verfügung, um daraus die Werte `rval` der Schaltfunktionen zum gegenwärtigen Zeitpunkt zu berechnen.

Um die diskreten Zustände auf die neuen Werte `ndstate` zu setzen, ruft `derf` die Funktion

```
void swdstate(double* t, double* q, double* qd, int* dstate, int* ndstate);
```

auf. Dabei werden über `ndstate` hinaus die gleichen Parameter wie bei der Funktion `rfkt` übergeben, so dass neben dem puren Schalten der Zustände noch weitere Berechnungen möglich sind, die der Schaltvorgang ggf. nach sich zieht. Alle Zustände, die bei dem jeweiligen Aufruf von `swdstate` unverändert bleiben, sind in `ndstate` durch den Wert Null gekennzeichnet, für die übrigen enthält `ndstate` die neuen Werte.

Die Detektion der Nullstellen der Schaltfunktionen und das Schalten mit der Funktion `swdstate` wird von `derf` vollkommen selbstständig durchgeführt. Nach jedem erfolgreichen Integrationsschritt von t_{i-1} nach t_i wird folgender Algorithmus durchlaufen:

1. Berechnung der Werte der Schaltfunktionen zum Zeitpunkt t_i.
2. Interpolation der Zustandvariablen und -geschwindigkeiten zum Zeitpunkt t_{i-1} und Berechnung der Werte der Schaltfunktionen.
3. Vergleich der Vorzeichen der Schaltfunktionswerte zu den beiden Zeitpunkten.
4. Wenn kein Vorzeichenwechsel stattgefunden hat, dann weiter bei 8.
5. Interpolation der Zustandvariablen und -geschwindigkeiten und Berechnung der Werte der Schaltfunktionen in der Mitte $t_m = (t_{i-1} + t_i)/2$ des Integrationsschritts.
6. Interpolation der jeweils drei Werte für jede Schaltfunktion durch ein Polynom zweiten Grades, analytische Berechnung der Nullstellen und bei Nulldurchgängen mehrerer Funktionen Bestimmung der frühsten Nullstelle t_0.
7. Interpolation der Zustandvariablen und -geschwindigkeiten zum Zeitpunkt t_0, Schalten der entsprechenden Zustände und Neustart der Integration.
8. Nächster Integrationsschritt zum Zeitpunkt t_{i+1}.

Die erfolgreiche Detektion der Nullstellen und die Zuverlässigkeit des Verfahrens hängt ausschließlich von der geschickten Wahl der Schaltfunktionen ab: Es ist darauf zu achten, dass die Funktionen im Intervall $[t_{i-1}, t_i]$ in der Nähe der Nulldurchgänge von möglichst niedriger Ordnung sind, denn nur so ist gewährleistet, dass der Algorithmus die Nullstellen mit hinreichender Genauigkeit bestimmt und keinen Schaltvorgang „übersieht“.

8 Zur Effizienz von DySim

Da DySim generalisierte Koordinaten verwendet und auf dem derzeitigen Stand keine geschlossenen kinematischen Ketten unterstützt, hat das zu lösende Gleichungssystem gewöhnlicher Differentialgleichungen (3) eine minimale Größe, was jedoch nicht bedeutet, dass auch der numerische Aufwand bei dessen Integration minimal ist.

Im wesentlichen haben zwei Faktoren entscheidenden Einfluss auf den numerischen Aufwand bei der Lösung: Zum einen müssen die Massenmatrix $\boldsymbol{M}$ und die Vektoren $\boldsymbol{c}$ und $\boldsymbol{f}$ der generalisierten CORIOLIS- und Zentrifugalkräfte sowie der eingeprägten Kräfte bestimmt werden, und zum anderen muss die Massenmatrix invertiert werden, um dem Integrator den Vektor der Zeitableitungen bereitstellen zu können.

Aufgrund ihrer Kompaktheit ist die Massenmatrix nicht dünn besetzt, wodurch die effizienten Verfahren zur Invertierung solcher Matrizen nicht verwendet werden können. Der Aufwand der Invertierungsverfahren einer allgemeinen $n \times n$ Matrix, wie der Massenmatrix eines Systems mit n Freiheitsgraden, ist von der Ordnung $\mathcal{O}(n^3)$. Wenn man die Tatsache ausnutzt, dass $\boldsymbol{M}$ symmetrisch und positiv-definit ist, und die CHOLESKY-Zerlegung (vgl. NUMERICAL RECIPES [12] Kap. 2.9.) verwendet, so kann man die Matrizeninversion noch etwa um den Faktor 2 gegenüber den allgemeinen Standardverfahren beschleunigen.

Einen sehr großen Anteil am numerischen Aufwand hat die Berechnung der Massenmatrix, da für den i-ten Teilkörper, dessen absolute Lage im Inertialsystem von n_i sog. *relevanten* Freiheitsgraden abhängt, $(n_i)^2$ Operationen erforderlich sind. Im ungünstigsten Fall einer einzigen, unverzweigten Kette, die mit einem Freiheitsgrad am Inertialsystem befestigt ist und die aus Körpern besteht, die jeweils über einen Freiheitsgrad relativ zu ihrem Mutterkörper verfügen, sind bei insgesamt n Freiheitsgraden daher

$$\sum_{k=1}^{n} k^2 = \frac{1}{3}\,n^3 + \frac{1}{2}\,n^2 + \frac{1}{6}\,n$$

Operationen nötig. Allerdings hängt dieser Aufwand stark von der Topologie des Modells ab: Sobald die kinematischen Ketten verzweigt sind, so dass die Anzahl der relevanten Freiheitsgrade der Körper abnimmt, oder sobald die Körper über mehr relative Freiheitsgrade zu ihrem jeweiligen Mutterkörper verfügen, reduziert sich der Aufwand beachtlich.

Für ein System von beispielsweise $n = 20$ Freiheitsgraden wirkt sich diese Abhängigkeit von der Topologie folgendermaßen aus: Wenn jeder Körper über n_f relative Freiheitsgrade verfügt und n_c unverzweigte, gleichlange Ketten vorliegen, dann sind

$$n_c \sum_{k=1}^{\frac{n}{n_f\,n_c}} (n_f\,k)^2 = \frac{1}{3}\,\frac{n^3}{n_f\,(n_c)^2} + \frac{1}{2}\,\frac{n^2}{n_c} + \frac{1}{6}\,n_f\,n$$

Operationen erforderlich. Abbildung 5 verdeutlicht, dass sich der numerische Aufwand bei der Konstruktion der Massenmatrix immer mehr reduziert, je größer die Anzahl an

Einzelketten ist, d.h. je geringer die Anzahl der relevanten Freiheitsgrade ist, oder je größer die Anzahl der relativen Freiheitsgrade der Körper ist.

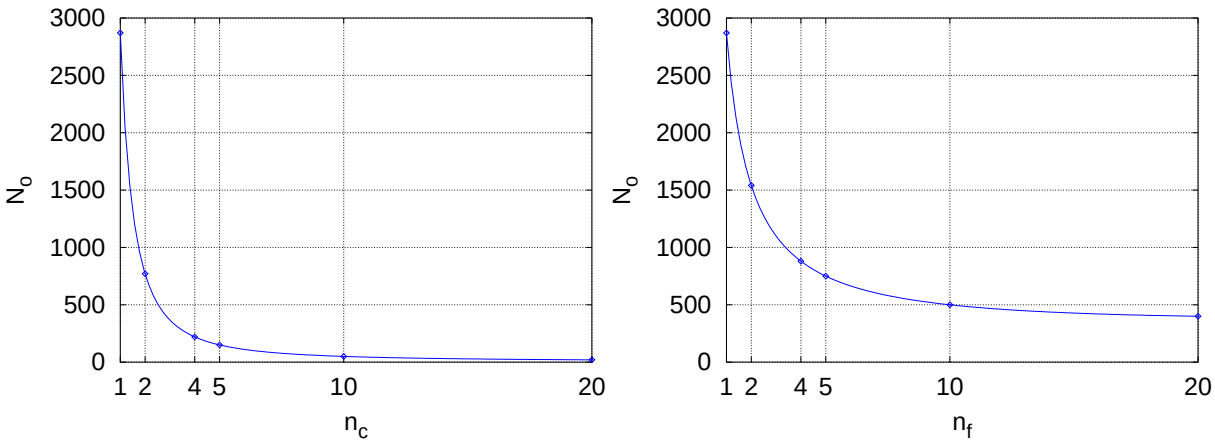


Abbildung 5: Die Anzahl der Operationen N_o bei der Konstruktion der Massenmatrix $\boldsymbol{M}$ mit DySim sinkt mit steigender Anzahl der Teilketten n_c, d.h. sinkender Anzahl relevanter Freiheitsgrade der Einzelkörper für n_f = konstant (links) und mit wachsender Anzahl n_f relativer Freiheitsgrade für n_c = konstant (rechts).

Diese Abhängigkeit der Anzahl der Operationen von der Topologie ändert letztlich jedoch nichts daran, dass der Aufwand bei der Berechnung der Massenmatrix $\boldsymbol{M}$ mit wachsender Zahl n von Freiheitsgraden mit der Ordnung $\mathcal{O}(n^3)$ ansteigt.

Da die Matrizeninversion ohnehin nicht effizienter als $\mathcal{O}(n^3)$ durchführbar ist, steigt der numerische und damit der zeitliche Aufwand bei der Integration mechanischer Systeme mit DySim mit $\mathcal{O}(n^3)$, so dass ab einer eine kritischen Modellgröße $\mathcal{O}(n^2)$- oder $\mathcal{O}(n)$-Verfahren überlegen sind. Aufgrund des variablen Aufwands bei der Bestimmung der Massenmatrix ist diese kritische Zahl der Freiheitgrade jedoch modellabhängig.

Um einen Anhaltspunkt zu erhalten, in welchem Bereich die kritische Zahl der Freiheitsgrade liegt, wurden mit einem Vergleichsmodell Simulationen mit DySim und mit MBSNAT von Kraus [5] durchgeführt. MBSNAT basiert auf natürlichen Koordinaten, bei denen jeder Körper durch 12 Koordinaten beschrieben wird, wodurch die Zahl der Bewegungsgleichungen enorm anwächst, die Massenmatrix jedoch dünn besetzt ist, so dass ein effizientes Verfahren der Ordnung $\mathcal{O}(n)$ zur Matrizeninversion verwendet werden kann.

Als Vergleichsmodell wurde ein Vielfachpendel betrachtet, das über ein Kugelgelenk an das Inertialsystem gekoppelt ist und dessen Einzelkörper ebenfalls über Kugelgelenke miteinander verbunden sind, so dass die Zahl der Freiheitsgrade des Systems mit jedem zusätzlichen Körper um drei anwächst.

Quantifiziert man den numerischen Aufwand durch die Integrationszeiten und bestimmt die Rechenzeit für 500 Zerlegungen (Berechnungen des Vektors der Geschwindigkeiten), so lassen sich – unabhängig von dynamischen Effekten und daraus resultierendem numerischem Mehraufwand aufgrund von variabler Schrittweitensteuerung des Integrators – Vergleiche anstellen.

Abbildung 6 zeigt auf der rechten Seite die auf einem Dual Pentium II (450 MHz) benötigte CPU-Zeit für jeweils 500 Zerlegungen in Abhängigkeit von der Zahl n der Freiheitsgrade. Mit wachsendem n zeigt sich deutlich das dominierende n^p-Verhalten mit einem Exponenten von $p = 2{,}98 \pm 0{,}047$. Der Aufwand für wenige Freiheitsgrade ist durch niedrigere Ordnungen bestimmt. Den Vergleich der Rechenzeiten bei Verwendung von DySim bzw.

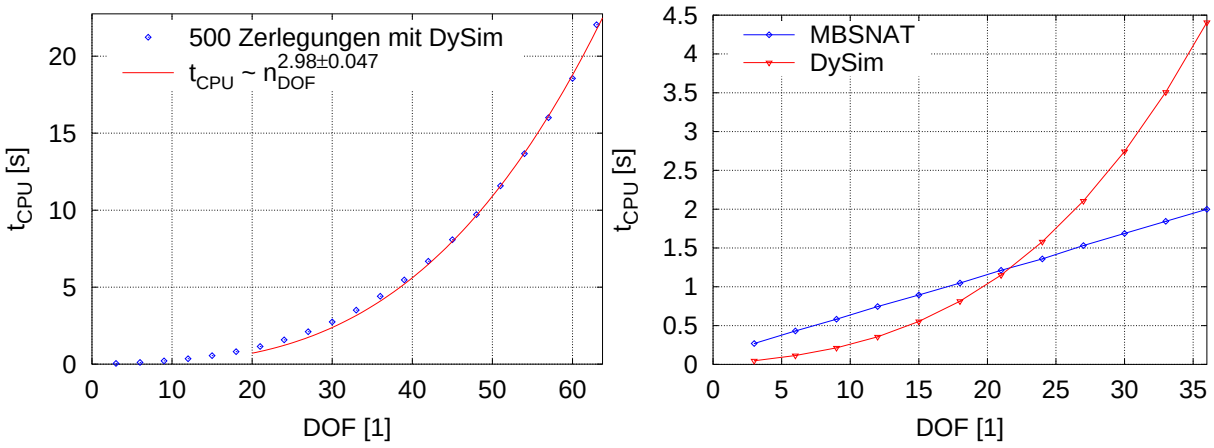


Abbildung 6: Die CPU-Zeit auf einem Dual Pentium II Rechner (450 MHz) für jeweils 500 Zerlegungen als Funktion der Zahl der Freiheitsgrade bei der Simulation eines Vielfachpendels: Die Teilkörper des Pendels verfügen jeweils über drei relative Freiheitsgrade. Links: $\mathcal{O}(n^3)$-Verhalten von DySim. Rechts: Benötigte CPU-Zeit von DySim und MBSNAT im Vergleich. Beim betrachteten Vielfachpendel ist das $\mathcal{O}(n)$-Verfahren von MBSNAT ab etwa 22 Freiheitsgraden überlegen.

bei Verwendung von MBSNAT zeigt die linke Seite von Abbildung 6. Die kritische Anzahl von Freiheitsgraden liegt bei diesem Modell bei etwa 22 Freiheitsgraden. Unterhalb von diesem Wert ist der Aufwand bei Verwendung von DySim beachtlich geringer, oberhalb beachtlich größer als bei dem $\mathcal{O}(n)$-Verfahren von MBSNAT.

Einige Bemerkungen zur Effizienz und zu weiterem Optimierungspotential

Zweifellos steigt der numerische Aufwand bei der Verwendung von DySim gegenüber anderen Simulationssystemen, die Verfahren von niedrigerer Ordnung erlauben, mit wachsender Zahl der Freiheitsgrade unangenehm stark an, und es wird immer eine Modellgröße geben, ab der diese Systeme effizienter arbeiten. Es stellt sich jedoch die Frage, ob die kritische Zahl hoch genug liegt, so dass mit DySim ein Großteil der Anwendungen – die Effizienz unterhalb der Grenze ausnutzend – abzudecken ist.

Die mit dem Testmodell ermittelte kritische Anzahl von Freiheitsgraden beschreibt den für DySim ungünstigen Fall einer einzigen langen Kette. Da die Anzahl von Operationen bei der Aufstellung der Massenmatrix stark von der Topologie des Modells abhängt (vgl. Abbildung 5), sinkt der Aufwand beträchtlich, sobald die kinematischen Ketten verzweigt sind oder die Teilkörper über eine größere Zahl relativer Freiheitsgrade verfügen. Damit wird die kritische Anzahl bei realistischen mechanischen Modellen in der Regel höher liegen.

Weiterhin gibt es bei DySim auf dem derzeitigen Stand der Programmierung noch Optimierungspotential. Bei den in Abbildung 6 dargestellten Rechnungen wurde noch nicht die effizientere Cholesky-Zerlegung zur Invertierung der Massenmatrix verwendet, die die benötigte CPU-Zeit verkürzen und damit die kritische Zahl zu höheren Werten verschieben wird. Zusätzlich kann die Berechnung der Massenmatrix programmiertechnisch noch optimiert werden, was die Rechnungen ebenfalls beschleunigen wird.

Da in DySim bei der Konstruktion der Massenmatrix darauf geachtet wird, dass kinematisch unabhängige Ketten in abgegrenzten Blöcken angeordnet werden, kann diese Blockgestalt bei der Zerlegung ausgenutzt werden: Im Fall kinematisch unabhängiger Ketten, die lediglich über Kraftkopplungen miteinander wechselwirken, ist es dadurch möglich, den numerischen Aufwand weiter zu reduzieren. Wird dann beispielsweise ein

Modell, bestehend aus einer kinematischen Kette, um eine identische zweite, von der ersten kinematisch entkoppelte Kette erweitert, so dass sich die Zahl der Freiheitsgrade verdoppelt, so wird der numerische Aufwand nicht auf das Achtfache, sondern lediglich auf das Doppelte anwachsen. Auf dem derzeitigen Stand der Programmierung ordnet DySim die Massenmatrix bereits in Blöcken an, allerdings nutzt die Matrizeninversion evtl. Blockstrukturen noch nicht aus.

Wenn man angesichts des noch vorhandenen Optimierungspotentials und der Abhängigkeit von der Modelltopologie davon ausgeht, dass bei durchschnittlichen mechanischen Modellen eine kritische Anzahl von 30 bis 35 Freiheitsgraden erreicht werden kann, so überschreitet eine Vielzahl der Anwendungen – vor allem auf dem Gebiet der Biomechanik – diese Zahl von Freiheitgraden nicht, und DySim ist aufgrund seiner Überlegenheit unterhalb dieser Grenze in weiten Bereichen einsetzbar.

Literatur

[1] GCC Team: *GCC 2.95.2*. http://www.gnu.org/software/gcc/gcc-2.95/gcc-2.95.2.html. 1999.

[2] ISO: *Programming languages – C++*. ISO/IEC 14882, International Organization for Standardization, Genf, Sept. 1998.

[3] ISO: *Programming languages – C*. ISO/IEC 9899, International Organization for Standardization, Genf, Dez. 1999.

[4] Kilgard, M. J.: *GLUT – OpenGL Utility Toolkit, version 3.7*. Silicon Graphics, Inc. http://www.opengl.org/developers/documentation/glut. 1998.

[5] Kraus, C./Winckler, M./Bock, H. G.: *Modeling mechanical DAE using natural coordinates*. Mathematical and Computer Modeling of Dynamical Systems (2001). Preprint.

[6] Leffler, S./Warmerdam, F./Welles, M.: *TIFF Software – libtiff*. http://www.libtiff.org/. 2001.

[7] Legnani, G./Adamini, R./Zappa, B.: *SPACELIB in C, version 2.1 — A software library for the kinematic and dynamic analysis of rigid bodies*. http://bsing.ing.unibs.it/~legnani/. 1998.

[8] Legnani, G./Righettini, P./Zappa, B./Casolo, F.: *A homogenous matrix approach to 3D kinematics and dynamics. Part I: Theory*. Mechanisms and Machine Theory (the scientific journal of IFToMM) **31** (1996), 573–587.

[9] Legnani, G./Righettini, P./Zappa, B./Casolo, F.: *A homogenous matrix approach to 3D kinematics and dynamics. Part II: Applications*. Mechanisms and Machine Theory (the scientific journal of IFToMM) **31** (1996), 589–605.

[10] Numerical Recipes Software: *Numerical Recipes Homepage*. http://www.nr.com/.

[11] The Portland Group: *PGI Workstation*. http://www.pgroup.com. 2000.

[12] Press, W. H./Teukolsky, S. A./Vetterling, W. T./Flannery, B. P.: *Numerical recipes in C — The art of scientific computing*, Aufl. 2. Cambridge University Press, Cambridge, 1995.

[13] Shampine, L. F./Gordon, M. K.: *Computer-Lösung gewöhnlicher Differentialgleichungen. Das Anfangswertproblem*. Übersetzung von J. Hoffmann. (Hrsg.: Engeln-Müllges, G.). Vieweg & Sohn, Braunschweig, 1984.