Starrkörpersimulation
Kugel in glatter Landschaft

Alexander Artiga Gonzalez
alexander@artigagonzalez.de

Universität Konstanz
14. April 2011

Zusammenfassung Diese Seminararbeit behandelt die Simulation von Starrkörperbewegungen. Als Beispiel für eine Starrkörpersimulation wird eine Kugel betrachtet, die über eine glatte Landschaft rollt.
Zunächst werden einige Definitionen und Sätze eingeführt, um eine Starrkörperbewegung zu definieren. Mit Hilfe der eingeführten Definitionen lassen sich gewöhnliche Differentialgleichungen herleiten, die die Bewegung der Kugel beschreiben.
Die Gleichungen werden mit numerischer Verfahren in MATLAB gelöst und die Bewegung der Kugel über einer glatten Landschaft visualisiert. Die dabei auftretenden Schwierigkeiten werden erläutert. Am Ende der Arbeit wird noch ein verbessertes Verfahren eingeführt.

Inhaltsverzeichnis

Einleitung

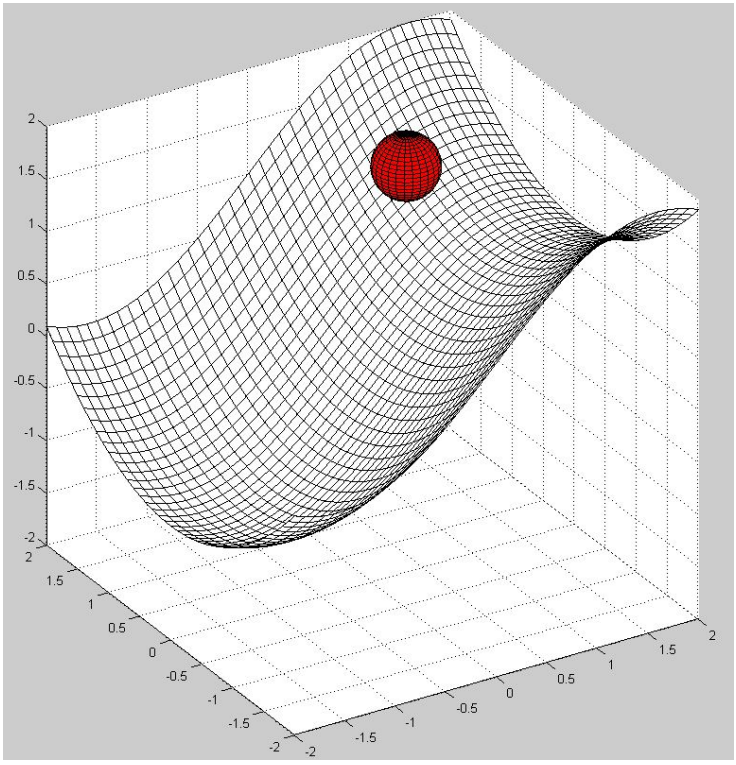

Abbildung 1. Kugel in glatter Landschaft

Das Ziel dieser Seminarabeit ist es, die Bewegung einer Kugel über einer glatten Landschaft zu beschreiben und danach zu simulieren.
Dabei wird die Kugel als Starrkörper betrachtet. Wie sich die Bewegung eines solchen Starrkörpers beschreiben lässt, wird im ersten Kapitel behandelt. Dazu werden die benötigten Grundlagen aus der Theorie der Starrkörperbewegungen eingeführt und erläutert. Das Resultat wird eine Gleichung für die Bewegung von Starrkörpern sein, die sich aus einer Rotation und einer Translation zusammensetzt.
Das zweite Kapitel beschäftigt sich mit der Modellierung des Problems. Hier wird festgelegt welche Vorraussetzungen Kugel und Landschaft erfüllen müssen. So soll es zum Beispiel nicht möglich sein, dass die Kugel sich von der Landschaft löst, also zum Bespiel über eine Erhebung springt. Man erhält mehrere Funktionen die Kugel und Landschaft beschreiben.
Mit Hilfe dieser Funktionen, der Gleichung aus dem ersten Kapitel und einiger

dynamischer Gleichungen aus der Physik werden sich dann gewöhnliche Differentialgleichungen herleiten lassen, die die Bewegung der Kugel beschreiben. Das Ergebnis ist ein System aus drei Differentialgleichung: eine für den Ort, eine für die Rotationsgeschwindigkeit und eine für die Rotation. Dies wird im dritten Kapitel beschrieben.
Im vierten Kapitel geht es dann um die Umsetzung dieser Ergebnisse in eine Simulation in MATLAB. Dazu wird die GUI (Grafical User Interface) und das Plotten der Landschaft inklusive Kugel erläutert. Es folgen die zum Lösen der Differentialgleichungen verwendeten Runge-Kutta-Verfahren (explizites Euler-Verfahren, klassisches Runge-Kutta 4, Verfahren von Lawson[2]). Führt man die so erstellte Simulation aus, lassen sich einige numerische Probleme erkennen. Unter anderem wird man sehen, dass bei der Verwendung von expliziten Verfahren zur Lösung der Differentialgleichung die Rotation nicht stabil ist. Dies führt zu Verformungen der Kugel, welche in einer Starrkörpersimulation nicht auftreten sollten.
Um dies zu verhindern wird im fünften Kapitel beschrieben, wie sich die Differentialgleichungen modifizieren lassen um eine stabile Rotation zu erhalten (Munthe-Kaas-Verfahren).

1 Starrkörperbewegungen

1.1 Definitionen

Ein Starrkörper ist ein nicht verformbarer Körper. Das bedeutet, dass zwei beliebige Punkte des Körpers immer den gleichen Abstand zueinander besitzen. Eine Abbildung, die einen Starrkörper bewegt, sollte also diese Eigenschaft erhalten. Dies ist genau die Aussage der ersten Definition (*abstandserhaltend*). Zusätzlich zu *abstandserhalten* werden noch die Definitionen für *winkelerhaltend* und *orientierungserhaltend* benötigt.

Definition 1. *Eine Abbildung* $L : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ *heißt* abstandserhaltend*, falls für alle* $x, y \in \mathbb{R}^3$ *gilt:*
$\|L(x) - L(y)\|_2 = \|x - y\|_2$

Definition 2. *Eine Abbildung* $L : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ *heißt* winkelerhaltend*, falls für alle* $x, y \in \mathbb{R}^3$ *gilt:*
$\|L(x)\|_2 \|L(y)\|_2 \langle L(x), L(y)\rangle = \|x\|_2 \|y\|_2 \langle x, y\rangle$

Definition 3. *Eine Abbildung* $L : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ *heißt* orientierungserhaltend*, falls für alle* $x_0, x_1, x_2, x_3 \in \mathbb{R}^3$ *gilt:*
$O(L(x_0), L(x_1), L(x_2), L(x_3)) = O(x_0, x_1, x_2, x_3)$ *mit*
$O(x_0, x_1, x_2, x_3) := sign(det(x_1 - x_0, ..., x_3 - x_0))$

1.2 Lemmas

Das folgende Lemma liefert nun die Eigenschaft *winkelerhaltend*, wenn zusätzlich zu *abstandserhalten* noch angenommen wird, dass für unsere Abbildung gilt $L(0) = 0$.

Lemma 1. *Sei* $L : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ abstandserhaltend *mit* $L(0) = 0$*. Dann ist* L winkelerhaltend.

Beweis. Für $x, y \in \mathbb{R}^3$ *gilt:*

$$\begin{aligned}\|L(x) - L(y)\|_2^2 &= \langle L(x) - L(y), L(x) - L(y)\rangle \\ &= \|L(x)\|_2^2 + \|L(y)\|_2^2 - 2\langle L(x), L(y)\rangle\end{aligned}$$

$$\|x - y\|_2^2 = \langle x - y, x - y\rangle = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y\rangle$$

Da L abstandserhaltend *ist folgt* $\|L(x) - L(y)\|_2^2 = \|x - y\|_2^2$.

$$\Longrightarrow \|L(x)\|_2^2 + \|L(y)\|_2^2 - 2\langle L(x), L(y)\rangle = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y\rangle$$

Sei $y = 0$*. Dann folgt* $L(y) = 0$ *und damit* $\|L(x)\|_2 = \|x\|_2$ *für alle* $x \in \mathbb{R}^3$*.*
Dann ist aber $\langle L(x), L(y)\rangle = \langle x, y\rangle$*.*
Damit ist L winkelerhaltend. □

Nun lässt sich einfach zeigen, dass eine *abstands-* und *winkelerhaltende* Abbildung *linear* sein muss.

Lemma 2. *Sei* $L : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ abstands- *und* winkelerhaltend. *Dann ist* L linear.

Beweis. Sei $k \in \mathbb{N}$ *beliebig. Dann gilt für beliebige* $\alpha_i \in \mathbb{R}$ *und* $x_i \in \mathbb{R}^3$, $i = (1, .., k)$*:*

$$\begin{aligned}\left\langle L(\sum_{i=1}^{k} \alpha_i x_i) - \sum_{i=1}^{k} \alpha_i L(x_i), L(u) \right\rangle &= \left\langle L(\sum_{i=1}^{k} \alpha_i x_i), L(u) \right\rangle - \sum_{i=1}^{k} \alpha_i \left\langle L(x_i), L(u) \right\rangle \\ &= \left\langle \sum_{i=1}^{k} \alpha_i x_i, u \right\rangle - \sum_{i=1}^{k} \alpha_i \left\langle x_i, u \right\rangle = 0\end{aligned}$$

für beliebige $u \in \mathbb{R}^3 \Longrightarrow L(\sum_{i=1}^{k} \alpha_i x_i) = \sum_{i=1}^{k} \alpha_i L(x_i)$
Damit ist L linear. □

Eine *abstandserhaltende* Abbildung mit $L(0) = 0$ ist also auch *winkelerhaltend* und *linear*. Fordert man nun noch zusätzlich von L, dass sie *orientierungserhaltend* ist, dann erhält man $L \in SO(3)$. L ist also eine Rotation.

Lemma 3. *Sei* $L : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ *eine* abstands- *und* orientierungserhaltende lineare *Abbildung. Dann gilt* $L \in SO(3)$.

Beweis.

$$\begin{aligned}O(0, Le_1, Le_2, Le_3) &= sign(det(Le_1, Le_2, Le_3)) = sign(det(L) \cdot det(e_1, e_2, e_3)) \\ &= sign(det(L)) \cdot sign(det(e_1, e_2, e_3)) \\ &= sign(det(L)) \cdot O(0, e_1, e_2, e_3)\end{aligned}$$

Da $O(0, e_1, e_2, e_3) = 1 = O(0, Le_1, Le_2, Le_3) \Rightarrow sign(detL) = 1$
$\Rightarrow L \in SO(3)$ □

1.3 Starrkörperbewegungen

Theorem 1. *Sei nun* $M : \mathbb{R}^3 \to \mathbb{R}^3$, $x \mapsto M(x)$ abstands- *und* orientierungserhaltend. *Sei* $L(x) := M(x) - M(0)$. *Dann ist* $M(x) = L(x) - M(0)$.
Eine Starrkörperbewegung $M(x)$ *lässt sich also als* Rotation $L(x)$ *und* Translation $M(0)$ *beschreiben.*

Beweis. $L(0) = M(0) - M(0) = 0$ *und* L abstands- *und* orientierungserhaltend. *Aus* Lemma 3 *folgt* $L \in SO(3)$.

Diese Aussage lässt sich einfach auf weitere Zeitpunkte $t \in [0, T]$ erweitern.

Theorem 2. *Eine Abbildung* $M : (t, x) \mapsto M(t, x) := M_t(x)$ *wird als* Starrkörperbewegung *bezeichnet, falls sie für alle* $t \in [0, T]$ abstands- *und* orientierungserhaltend *ist.*
$M_t(x)$ *gibt also die Position von* x *zum Zeitpunkt* t *an.*
Zu solch einer Abbildung M *existieren eine Abildung* $R : [0, T] \to SO(3)$ *und eine Abbildung* $C : [0, T] \to \mathbb{R}^3$*, so dass gilt:* $M_t(x) = R(t)x + c(t)$*.*

Beweis. Folgt aus Theorem 1*.*

$M_t(x)$ ist also die gesuchte Abbildung, die eine Starrkörperbewegung beschreibt. Diese wird im dritten Kapitel zum Herleiten der Differentialgleichungen benutzt. Zuerst werden aber noch einige Informationen über die „Kugel in glatter Landschaft“ benötigt.

2 Kugel in glatter Landschaft

2.1 Die Kugel

Zu Begin werden einige Infomationen über die Kugel benötigt. Um das Modell einfach zu halten wird angenommen, dass die Masse der Kugel homogen verteilt ist. Es genügt eine Referenzkugel um den Urspung. Diese kann in jedem Zeitpunkt der Simulation rotiert und verschoben werden. Der Vorteil einer homogenen Kugel um den Urspung ist, dass der Schwerpunkt im Urspung liegt und der Trägheitstensor an dieser Stelle sich einfach berechnen lässt. Die folgenden Aussagen gelten also für die Kugel:

- Referenzkugel $B = B_r(0), r > 0$ (Kugel mit Radius r um 0)
- Masse $m(B) > 0$ (homogene Massenverteilung)
- Schwerpunkt $\overline{x}(B) = 0$
- Trägheitstensor $T^*_{\overline{x}}(B) = \frac{2}{5} \cdot m(B) \cdot r^2 \cdot I$

2.2 Die glatte Landschaft

Auch die Landschaft, die durch eine Funktion dargestellt wird, muss einige Vorraussetzungen erfüllen. So sollte es keine Täler, also Minima, der Funktion geben, durch die die Kugel nicht „passt“. Beim Versuch durch solch ein Tal zu rollen hätte die Kugel zwei Berührpunkte mit der Funktion. In der Simulation würde die Kugel die Landschaft auf der anderen Seite des Tals durchstoßen. Im Tal selbst stimmt das Verhalten der Kugel dann nicht mehr, da sie nicht auf der anderen Seite hochrollen kann. Ebenso sollte beim Auswählen der Landschaftsfunktion bedacht werden, dass die Kugel nicht „springen“ kann. Lässt man die Kugel mit zu hoher Geschwindikeit über eine Anhöhe (Maximum der Funktion) rollen, bleibt diese auf der Landschaftsoberfläche „kleben“. Diese Einschränkungen sollte beim Verwenden der Simulation bedacht werden, da der Algorithmus zum Lösen der Gleichungen trotzdem Ergebnisse liefert.

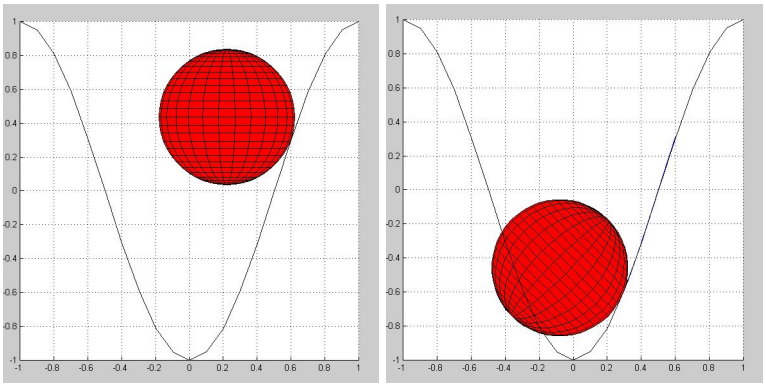

Abbildung 2. Zu große Kugel

Außerdem muss die Landschaft natürlich „glatt“ sein. Es darf keine Unstetigkeitsstellen geben, da das Verhalten der Kugel dort nicht definiert ist. Und man wird sehen, dass die Landschaftsfunktion zweimal differenzierbar sein muss, da für die Differentialgleichungen die Ableitung der Normalenvektorfunktion benötigt wird.

2.3 Kugel und Landschaft

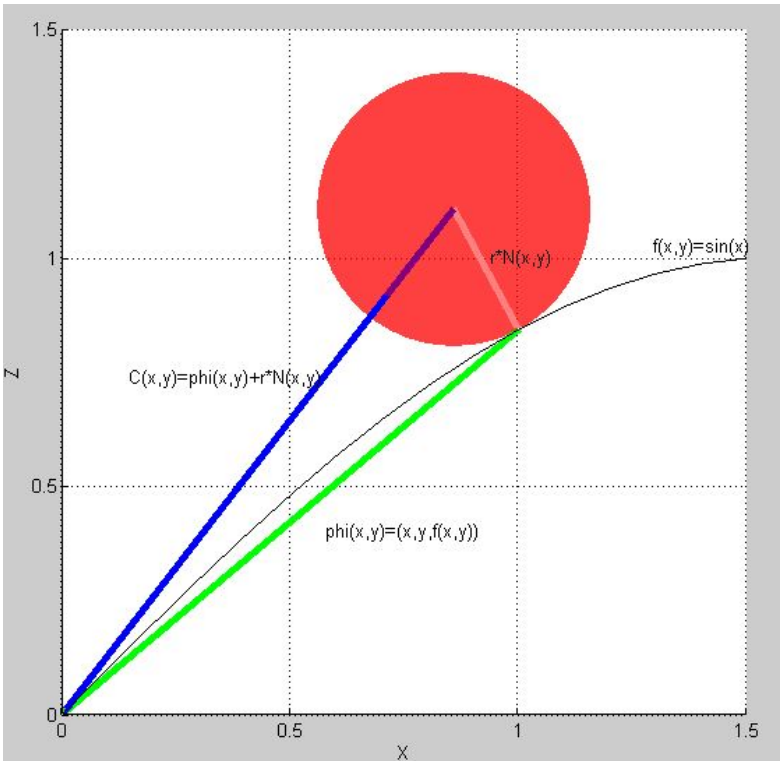

Abbildung 3. Kugel und Landschaft

Um das Verhalten zwischen Kugel und Landschaft zu beschreiben, werden folgende Funktionen benötigt (siehe auch Abbildung 3):

- Landschaftsfunktion:

$$f : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto f(x, y)$$

- Landschaftsoberfläche:

$$\varphi : \mathbb{R}^2 \to \mathbb{R}^3, (x, y) \mapsto (x, y, f(x, y))$$

- Normalenfeld:

$$N : \mathbb{R}^2 \to \mathbb{R}^3, (x, y) \mapsto \frac{\partial_1\varphi \times \partial_2\varphi}{\|\partial_1\varphi \times \partial_2\varphi\|}$$

- Mittelpunktsfunktion:

$$C : \mathbb{R}^2 \to \mathbb{R}^3, (x, y) \mapsto \varphi(x, y) + rN(x, y)$$

2.4 Reibung

Wichtig für das Modell sind auch die angenommenen Werte für die Reibung. So gibt es keine Rollreibung. Die Kugel rollt , wenn es eine Startgeschwindigkeit gibt oder die Kugel auf einer Erhöhung der Landschaft startet, unendlich lange. Im Gegensatz dazu soll die Haftreibung unendlich groß sein. Dies verhindert, dass die Kugel nur rutscht und zwingt sie zum Rollen. Diese sogenannte *schlupffreiheit* ist eine wichtige Eigenschaft, die im nächsten Kapitel benötigt wird.

3 Differentialgleichungen

Mit Hilfe der im letzten Kapitel definierten Funktionen lassen sich nun die Differentialgleichungen, die die Bewegung der Kugel auf der Landschaft beschreiben, herleiten. Zuerst wird die Rotationsgleichung betrachtet. Diese lässt sich ohne Einschränkung auf die Kugel oder die Landschaft herleiten. Das Ergebnis ist also eine Differentialgleichung für beliebige Rotationen. Für die Orts- und die Geschwindigkeitsgleichung werden dann die Vorraussetzungen aus dem Modell benötigt.

3.1 Rotationsgleichung

Für die Rotation R gilt $R \in SO(3)$, also $RR^T = I$.
$\Longrightarrow \frac{d}{dt} RR^T = 0$
$\Longrightarrow \dot{R}R^T + R\dot{R}^T = \dot{R}R^T + (\dot{R}R^T)^T = 0$
$\Longrightarrow \dot{R}R^T = -(\dot{R}R^T)^T$
Sei $A = \dot{R}R^T$ dann folgt $A = -A^T$. A ist also schiefsymmetrisch.
$A = \dot{R}R^T \Longrightarrow \dot{R} = AR$
Es gilt also

$$\dot{R} = A(\omega)R$$

mit

$$A(\omega) = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$$

schiefsymmetrisch.

3.2 Ortsgleichung

Für die Ortsgleichung, sowie auch für die Geschwindigkeitsgleichung, wird die im letzten Kapitel beschriebene *schlupffreiheit* benötigt. Durch die *schlupffreiheit* erhält man im Berührungspunkt immer eine Geschwindigkeit von null (Abbildung 4).

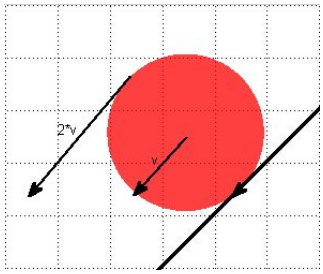

Abbildung 4. Rollgeschwindigkeit

Aus den x-y-Koordinaten für den Berührungspunkt lässt sich mit Hilfe der *Mittelpunktsfunktion* C (Kapitel 2.3) der Mittelpunkt der Kugel bestimmen. Es ist daher hilfreich eine Funktion zu definieren, die den Berührpunkt zwischen Kugel und Landschaft zu einem Zeitpunkt $t \in [0, T]$ liefert. Sei also $\alpha(t)$ eine Funktion über dem Intervall $[0, T]$, so dass

$$\alpha : [0, T] \to \mathbb{R}^2, t \mapsto (x, y)$$

mit

$$M_t(\alpha(t)) = \varphi(\alpha(t)) = b(t)$$

wobei $b(t)$ der Berührungspunkt zwischen Kugel und Landschaft zum Zeitpunkt t ist.
Die Gleichung für die Starrkörperbewegung ist (Kapitel 1.3)

$$M_t(x) = R(t)x + c(t)$$

Für die Geschwindigkeit erhält man dann

$$\frac{d}{dt}M_t(x) = \dot{R(t)}x + \dot{c(t)}$$

Für $\alpha(t)$ folgt dann aus der *schlupffreiheit*

$$\dot{M}_t(\alpha(t)) = \dot{R}(t)\alpha(t) + \dot{c}(t) = 0$$

Aus dem letzten Absatz ist bekannt $\dot{R} = A(\omega)R$. Es folgt:

$$\dot{c}(t) = -A(w(t))R(t)\alpha(t) = -A(w(t))(b(t) - c(t)) = A(w(t)) \cdot r \cdot N(t)$$

Da $c(t)$ der Verschiebung des Kugelmittelpunkts entspricht, gilt $c(t) = C(\alpha(t))$ und damit

$$\dot{c}(t) = C'(\alpha(t))\dot{\alpha}(t)$$

Man erhält:

$$C'(\alpha(t))\dot{\alpha}(t) = A(w(t)) \cdot r \cdot N(t)$$
$$\iff C'(\alpha(t))^T C'(\alpha(t))\dot{\alpha}(t) = C'(\alpha(t))^T A(w(t)) \cdot r \cdot N(t)$$

Sei $E(\alpha(t)) = C'(\alpha(t))^T C'(\alpha(t))$:

$$\begin{aligned}
&\Longrightarrow E(\alpha(t))\dot{\alpha}(t) = C'(\alpha(t))^T A(w(t)) \cdot r \cdot N(t) \\
&\Longleftrightarrow \dot{\alpha}(t) = E(\alpha(t))^{-1} C'(\alpha(t))^T A(w(t)) \cdot r \cdot N(t)
\end{aligned}$$

Die zweite Differentialgleichung liefert über den Berührpunkt den Ort der Kugel. Um die beiden ersten Differentialgleichungen zu lösen wird ω benötigt. Dies liefert die dritte Differentialgleichung, in die auch die physikalische Parameter wie Gewichtskraft und Drehimpuls einfließen.

3.3 Geschwindigkeitsgleichung

Aus den physikalischen Gleichungen für Impuls/Drehimpuls sowie für die auf die Kugel wirkenden Kräfte erhält man:

$$\dot{(T_\alpha\omega)} = -m(B)A(R\dot{\alpha})A(\omega)R(\bar{x}-\alpha) - m(B)A(R(\bar{x}-\alpha))\dot{(M_t(\alpha)\,)} + m(B)A(R(\bar{x}-\alpha))g$$

Mit $\dot{(M_t(\alpha)\,)} = 0$ (Schlupffreiheit) folgt:

$$\dot{(T_\alpha\omega)} = m(B)A(R(\bar{x}-\alpha))g - m(B)A(R\dot{\alpha})A(\omega)R(\bar{x}-\alpha)$$

Durch Rechnen erhält man:

$$R\dot{\alpha} = rA(\omega)N(\alpha) - rN'(\alpha) \text{ und } R(\bar{x}-\alpha) = rN(\alpha)$$

Damit folgt:

$$\dot{(T_\alpha\omega)} = m(B)rA(N(\alpha))g - m(B)r^2A(N'(\alpha)\dot{\alpha})A(N(\alpha))\omega$$

Mit $v(t) := T_\alpha(t)\omega(t)$ erhält man als dritte Differentialgleichung:

$$\dot{v}(t) = mrA(N(\alpha(t)))g - mr^2A(N'(\alpha(t))\dot{\alpha}(t))A(N(\alpha(t)))\omega(t)$$

Um die Bewegung der Kugel zu simulieren muss also das folgende Differentialgleichungssystem gelöst werden:

$$\dot{v}(t) = mrA(N(\alpha(t)))g - mr^2A(N'(\alpha(t))\dot{\alpha}(t))A(N(\alpha(t)))\omega(t) \tag{1}$$

$$\dot{\alpha}(t) = rE(\alpha(t))^{-1}C'(\alpha(t))^TA(\omega(t))N(\alpha(t)) \tag{2}$$

$$\dot{R}(t) = A(\omega)R(t) \tag{3}$$

Die genaue Realisierung wird im nächsten Kapitel skizziert.

4 Simulation

Die Simulation setzt sich aus zwei Bereichen zusammen. Die Lösung der Differentialgleichungen mit Hilfe geeigneter numerischer Lösungsverfahren und die GUI (Graphical User Interface) mit deren Hilfe sich die Parameter des Modells angeben lassen und die die Bewegung der Kugel simuliert (Plot).

4.1 GUI

Es gibt mehrere Möglichkeiten eine GUI mit MATLAB zu erstellen. Zum einen bietet sich das von MATLAB mitgelieferte Tool GUIDE an. Es liefert eine grafische Oberfläche mit deren Hilfe GUI-Elemente erstellt und angeordnet werden können. In diesem Fall, da die GUI mit dem Programm gewachsen ist, wurden die Bedienelemente direkt in den Code eingefügt.

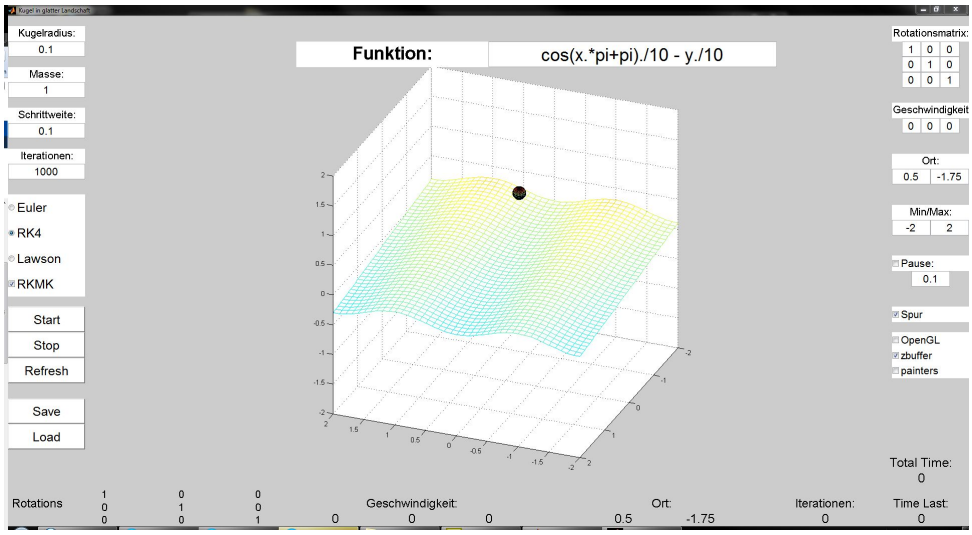

Abbildung 5. Programm

Dabei lassen sich in MATLAB Eingabe- und Anzeigefelder, sowie Buttons durch „uicontrol“-Elemente erstellen.

uicontrol-Elemente Es gibt Unterschiedliche Typen von „uicontrol“-Elementen: *Checkboxen, Eingabefelder, Frames, Listboxen, Popup-Menüs, Radio Buttons, Push Buttons, Slider, Textfelder und Toggle Buttons*

Abbildung 6. GUI-Elemente

Folgender Code erzeugt zum Beispiel einen Radio Button mit der Beschriftung „Euler“ (Abbildung 6, links unten):

```
ge.euler = uicontrol('style','radiobutton',
                     'background','white',
                     'unit','nomalized',
                     'FontUnits','normalized',
                     'FontSize',0.5,
                     'position',[0 .60 .08 .05],
                     'string','Euler',
                     'callback',@eulerChecked);
```

Unter `http://www.mathworks.com/help/techdoc/ref/uicontrol.html` gibt es eine Auflistung der möglichen Optionen. Wichtig ist vorallem die Option „callback“, bei der eine Funktion angegeben werden kann, die ausgeführt wird, wenn das Element sich verändert. Zum Beispiel durch Betätigen eines Buttons oder Editieren eines Eingabefelds.
Um auf den Inhalt oder Status zuzugreifen wird der Befehl *get(Objekt, Feld)* benutzt. Zum Beispiel *get(Objekt, 'string')* für den Inhalt eines Textfelds oder *get(Objekt, 'Value')* um den Status eines Radio Buttons zu erfahren. Dies wird in den folgenden Beispielen oft genutzt, um Einstellungen oder Werte abzufragen.

Die GUI-Elemente sind am Rand um das Koordinatensystem angeordnet (Abbildung 5). Es ist aber auch möglich, das Koordinatensystem (axis-Element) zu verschieben.

Die getätigten Einstellungen können gespeichert und geladen werden (siehe Anhang A).

Plot Auch zum Darstellen der Landschaft und der Kugel liefert MATLAB mehrere Möglichkeiten. Im Codebeispiel unten wird die Landschaft mit *mesh* erzeugt. Der Vektor *v* definiert dabei das Gitter (*meshgrid(v)*) und wird mit Hilfe der Werte aus den Textfeldern *ge.smin* und *ge.smax* erstellt. *landscape* ist die Landschaftsfunktion.
Eine Kugel um den Ursprung liefert der Befehl *sphere*. Da *sphere* Daten für *surf* erzeugt sind *Xs*, *Ys* und *Zs* Matrizen. Sinnvoll ist es aber, mit Koordinaten einer Referenzkugel um den Ursprung in Vektorform zu arbeiten, da sich auf diese die Rotation und Translation direkt anwenden lässt. Dazu werden die Kugeldaten in den Matrizen mit *surf2patch(Xs, Ys, Zs, Zs)* konvertiert und in *ge.ball* gespeichert. Der Befehl *get(ge.ball,'Vertices')'* liefert dann Daten im gesuchten Format. Die letzte Zeile im Codebeispiel zeigt wie die Kugel verschoben und gedreht wird. Dabei ist *r* der Radius, *R* ist die Rotationsmatrix und *mid* sind die Koordinaten des Kugelmittelpunktes.

```
hold off;
v = str2double(get(ge.smin,'string')):0.1:str2double(get(ge.smax,'string'));
[X,Y] = meshgrid(v);
Z = landscape(X, Y);
mesh(X, Y, Z);
hold on;
NoV=26; %gerade!
[Xs Ys Zs]=sphere(NoV);
ge.ball=patch(surf2patch(Xs, Ys, Zs, Zs));
ge.refvertices=get(ge.ball,'Vertices')';
set(ge.ball,'Vertices',(r*R*ge.refvertices)'+repmat(mid,size(ge.refvertices,2),1));
```

4.2 Lösungsverfahren

Um zu bestimmen wie die Kugel sich auf der Landschaft bewegt müssen die Differentialgleichungen gelöst werden. Dazu werden die in Kapitel 2.3 definierten Funktionen benötigt. Diese lassen sich aus der Landschaftsfunktion, die über die GUI eingegeben werden kann, berechnen.

Funktionen Eine Möglichkeit um die Ableitungen zu berechnen, ist die Verwendung der Symbolic Math Toolbox von MATLAB (`http://www.mathworks.com/help/toolbox/symbolic/`). Leider liefern die Funktionen der Symbolic Math Toolbox (*diff*, *jacobian*) einen symbolischen Ausdruck, der bei jedem Aufruf mit *eval* ausgewertet werden müsste. Da *eval* sehr langsam ist, wird die Hilfsfunktion *makefun* benötigt. Diese kopiert den symbolischen Ausdruck in einen String, dessen Auswertung wesentlich weniger Zeit in Anspruch nimmt.

```
function getFunctions(varargin)
   r = str2double(get(ge.radius,'string'));
   landscape = eval(['@(x,y)',get(ge.fun,'string')]);
   phi = eval(['@(x,y)','[x y landscape(x,y)]']);
   fx = diff(landscape(symx, symy), symx);
   FX = makefun(fx);
   fy = diff(landscape(symx, symy), symy);
   FY = makefun(fy);
   N = eval(['@(x,y)','[-FX(x,y) -FY(x,y) 1]./sqrt((-FX(x,y))^2+(-FY(x,y))^2+1)']);
   C = eval(['@(x,y)','phi(x,y)+r*N(x,y)']);
   dc = jacobian(C(symx, symy)+0*symx,[symx symy]);
   DC = makefun(dc);
   E = eval(['@(x,y)','DC(x,y)''*DC(x,y)']);
   dn = jacobian(N(symx, symy)+0*symx,[symx symy]);
   DN = makefun(dn);
   ref();
end

function f = makefun(s)
[m,n]=size(s);
str = '';
for i=1:m
    for j = 1:n
        str = [str,char(s(i,j)),','];
    end
    str(end)=';';
end
str = ['[',str,']'];
str = ['@(symx,symy)',str];
f = eval(str);
```

Differentialgleichungen Die Differentialgleichungen werden in einer Funktion zusammengefasst:

```
function [ dx ] = f(x)
global A r N E m g DC DN;
R = [x(1) x(2) x(3) ; x(4) x(5) x(6) ; x(7) x(8) x(9)];
v = [x(10) x(11) x(12)]';
alpha = [x(13) x(14)];
T = R*(2/5*m*r^2*eye(3))*R'-m*r^2*A(N(alpha(1),alpha(2)))^2;
w = T\v;
dR = (A(w)*R)';
dalpha = r*(E(alpha(1),alpha(2))\(DC(alpha(1),alpha(2))'*A(w)*N(alpha(1),alpha(2))'));
dv = m*r*A(N(alpha(1),alpha(2)))*g-m*r^2*A(DN(alpha(1),alpha(2))*dalpha)
         *A(N(alpha(1),alpha(2)))*w;
dx = [(dR(:))' dv' dalpha']';
end
```

Als einfaches Lösungsverfahren bietet sich ein explites Runge-Kutta-Verfahren (siehe Anhang B) an. Alternativ können auch die MATLAB Solver verwendet werden (`http://www.mathworks.com/help/techdoc/ref/ode23s.html`). Im Programm wird das Runge-Kutta-Verfahren aus Anhang B verwendet. Es gibt 3 Tableaus zur Auswahl: *Euler-Cauchy*, *RK4* und *Lawson*.

Probleme Alle drei im Programm verwendete Verfahren haben einen großen Nachteil. Wird die Schrittweite nicht klein genug gewählt, verformt sich die Kugel. Wird die Schrittweite klein genug gewählt, rollt die Kugel sehr langsam. Der Grund für die Verformung der Kugel findet sich in der Berechnung der Rotationsmatrix:

$$\begin{aligned} R(t_{n+1}) &= R(t_n) + \Delta t A(\omega(t_n))R(t_n) \\ &= (I + \Delta t A(\omega(t_n)))R(t) \end{aligned}$$

Damit $R(t_{n+1})$ wieder eine Rotationsmatrix ist, müssten schon $(I+\Delta t A(\omega(t_n)))$ und $R(t)$ aus $SO(3)$ sein. Da $A(\omega(t_n))$ schiefsymmetrisch ist, kann $(I+\Delta t A(\omega(t_n)))$ aber nicht in $SO(3)$ liegen. $R(t_{n+1})$ ist also keine Rotationsmatrix und es kann passieren, dass $R(t_{n+1})$ die Kugel verform anstatt sie zu drehen. Eine mögliche Lösung für dieses Problem wird im nächsten Kapitel vorgestellt.
Aber auch bei der Geschwindigkeit gibt es numerische Fehler. So gewinnt die Kugel zum Beispiel beim Euler-Cauchy Verfahren mit genügend großer Schrittweite Energie und rollt über Erhöhungen. Hier liefert aber schon der Wechsel auf Runge-Kutta 4 eine starke Verbeserung.

5 Munthe-Kaas-Verfahren

Das Munthe-Kaas-Verfahren soll eine Verformung der Kugel verhindern. Dazu muss $R(t_{n+1})$ eine Rotationsmatrix bleiben. Das Problem: $(I + \Delta t A(\omega(t_n)))$ ist keine Rotationsmatrix. I und $\Delta t A(\omega(t_n))$ sind aber die ersten zwei Summanden von $\exp(\Delta t A(\omega(t_n))$ und $\exp(\Delta t A(\omega(t_n))$ ist eine Rotationsmatrix.

$$\begin{aligned} &\exp(\Delta t A(\omega(t_n))) = I + \Delta t A(\omega(t_n)) + \sum_{k=2}^{\infty} \frac{1}{k!} (\Delta t A(\omega(t_n)))^k \\ &\implies (I + \Delta t A(\omega(t_n))) \approx \exp(\Delta t A(\omega(t_n))) \end{aligned}$$

Es gilt:

$$\begin{aligned} &\exp(A)\exp(A)^T = \exp(A)\exp(A^T) = \exp(A)\exp(-A) = \exp(0) = I \\ &\implies \exp(A) \in SO(3) \\ &\implies (I + \Delta t A(\omega(t_n))) \approx \exp(\Delta t A(\omega(t_n))) \in SO(3) \end{aligned}$$

Mit dieser Änderung entfällt die Berechnung von $\dot{R}(t)$.

Um $R(t_{n+1}), \omega(t_{n+1}), \alpha(t_n+1)$ aus $R(t_n), \omega(t_n), \alpha(t_n)$ zu berechnen werden die folgende zwei Schritte benötigt:
1. Schritt: Löse AWP auf $[t_n, t_{n+1}]$

$$\dot{u} = [I - \frac{1}{12}A(u)(6I + A(u))]\omega \tag{1}$$
$$\dot{v} = mrA(N(\alpha)g - mr^2A(N'(\alpha)\dot{\alpha})A(N(\alpha))\omega \tag{2}$$
$$\dot{\alpha} = rE(\alpha)^{-1}C'(\alpha)^TA(\omega)N(\alpha) \tag{3}$$

mit Anfangswerten $u(t_n) = 0$, $v(t_n) = T_a(t_n)\omega(t_n)$, $\alpha(t_n)$ gegeben.
2. Schritt:

$$w(t_{n+1}) = T_a(t_{n+1})^{-1}v(t_{n+1}) \tag{1}$$
$$R(t_{n+1}) = \exp(A(u(t_{n+1}))R(t_n) \tag{2}$$

Hierbei sollte bedacht werden, dass im Allgemeinen die Berechnung des Matrixexponentials $\exp(A(u(t_{n+1}))$ aufwändig ist. In diesem Fall lässt sich die Berechnung aber mit Hilfe der Rodriguez-Formel stark vereinfachen.

5.1 Rodriguez-Formel

Lemma 1 *Für eine schiefsymmetrische Matrix* $\exp(A(z))$ *gilt:*

$$\exp(A(z)) = \cos(\|z\|)I + \sin(\|z\|)A\left(\frac{z}{(\|z\|)}\right) + \frac{1-\cos(\|z\|)}{\|z\|^2}zz^T$$

Beweis. Für Potenzen von A gilt:

$$A(\omega)^2 = \omega\omega^T - \|\omega\|^2 I$$
$$A(\omega)^{2k+1} = (-1)^k|\omega\|^2 kA(\omega),\ (k \geq 0)$$
$$A(\omega)^{2k} = (-1)^{k-1}|\omega\|^{2k-2}A(\omega),\ (k \geq 1)$$

$$\begin{aligned}
\Longrightarrow exp(A(z)) &= \sum_{k=0}^{\infty}\frac{1}{k!}A(z)^k \\
&= I + \sum_{j=1}^{\infty}\frac{1}{(2j)!}A(z)^{2j} + \sum_{j=0}^{\infty}\frac{1}{(2j+1)!}A(z)^{2j+1} \\
&= 1 + \sum_{j=1}^{\infty}\frac{(-1)^{j-1}}{(2j)!}\|z\|^{2j-2}A(z)^2 + \sum_{j=0}^{\infty}\frac{(-1)^j}{(2j+1)!}\|z\|^{2j}A(z) \\
&= \frac{zz^T}{\|z\|^2} + \sum_{j=0}^{\infty}\frac{(-1)^{j-1}}{(2j)!}\|z\|^{2j-2}A(z)^2 + sin\|z\|A\left(\frac{z}{\|z\|}\right) \\
&= \cos(\|z\|)I + \sin(\|z\|)A\left(\frac{z}{(\|z\|)}\right) + \frac{1-\cos(\|z\|)}{\|z\|^2}zz^T
\end{aligned}$$

Das Munthe-Kaas-Verfahren verhindert eine Verformung der Kugel. Dies erlaubt es in der Praxis die Schrittweite zu vergrößern und die Kugel schneller rollen zu lassen. Die Implementierung des Munthe-Kaas-Verfahren findet sich in Anhang C.

Anhang

A Laden und Speichern in MATLAB

Ein kurzer Überblick, wie sich Daten in einer Variable speichern und wieder laden lassen:

```
%save configuration
function saveConf(varargin)
    %write data to var
    var.r = get(ge.radius,'string');
            %... fill v here
    var.max = get(ge.smax,'string');
    %save var to file
    uisave('var','examples/settings');
end

%load configuration
function loadConf(varargin)
    %load var
    [FileName,PathName] = uigetfile('examples/*.mat','Choose settings');
    file=[PathName FileName];
    load(file, 'var');
    %get data from var
    set(ge.radius,'string',var.r);
            %... get all values here
    set(ge.smax,'string', var.max);
    %update Functions and Plot
    opengl();zbuffer();painters();
    getFunctions();
end
```

B Runge-Kutta-Verfahren in MATLAB

Beispiel für die Implementierung des expliziten Runge-Kutta-Verfahrens mit variablen Tableau *TAB*.
data: Matrix mit Startwerten in *data(1,:)*
h: Schrittweite
f: Differentialgleichung

```
%Startwerte hier initialisieren
%Mögliche Tableaus:
%TAB = [1 0 ; 0 1];
TAB = [0 0 0 0 0; 0.5 0.5 0 0 0; 0.5 0 0.5 0 0; 1 0 0 1 0;0 1/6 1/3 1/3 1/6];
for i=2:n+1
    data(i,:)= data(i-1,:) + h*RKV(@f,data(i-1,:),h,TAB)';
end
```

```
function V = RKV(fhandel,yi,h,A)
%berechnet die Verfahrensfunktion V für ein RKV mit Tableau A an der Stelle
%yi von der Funktion f=fhandel mit Schrittweite h
%beachte: kein explizites Auftreten der Zeit in f berücksichtigt!
yi = yi';
s = length(A(:,1))-1; %Stufe bestimmen
L(:,1) = fhandel(yi); %K1 initialisieren
for j = 2:s
    for i = 1:length(yi)
    z(i) = A(j,2:j)*L(i,:)';
    end
    L(:,j) = fhandel(yi+h*z'); %Iterationsschritt
end
for i = 1:length(yi)
V(i) = A(s+1,2:s+1)*L(i,:)'; %Verfahrensfunktion
end
V=V';
```

C Munthe-Kaas-Verfahren in MATLAB

Differentialgleichungen:

```
function [ dx ] = f_rkmk( x )
%F_RKMK
global A m r N DC DN E g;
u = [x(1) x(2) x(3)];
v = [x(4) x(5) x(6)]';
alpha = [x(7) x(8)];
An = A(N(alpha(1),alpha(2)));
T_a = m*r^2*(2/5*eye(3)-An^2);
w = T_a\v;
du = (eye(3)-1/12*A(u)*(6*eye(3)+A(u)))*w;
dalpha = r*(E(alpha(1),alpha(2))\(DC(alpha(1),alpha(2))'*A(w)*N(alpha(1),alpha(2))'));
dv = m*r*An*g-m*r^2*A(DN(alpha(1),alpha(2))*dalpha)*A(N(alpha(1),alpha(2)))*w;
dx = [du' dv' dalpha'];
end
```

RKMK mit Rodriguez-Formel:

```
for i=2:n+1
    t = tic;
  %RKMK
  An = A(N(data(i-1,13),data(i-1,14)));
  T_a = m*r^2*(2/5*eye(3)-An^2);
  w = [data(i-1,10) data(i-1,11) data(i-1,12)]';
  v = T_a*w;
  x = [0 0 0  v' data(i-1,13) data(i-1,14)];
  rkv = x + h*RKV(@f_rkmk,x,h,TAB)';
  An = A(N(rkv(7),rkv(8)));
  T_a = m*r^2*(2/5*eye(3)-An^2);
  v = [rkv(4) rkv(5) rkv(6)]';
  w = T_a\v;
  u = [rkv(1) rkv(2) rkv(3)]';
  if (norm(u)==0)
     expo = cos(norm(u))*eye(3)+sin(norm(u))
                    *A(u)+(1-cos(norm(u)))*(u*u');
  else
     expo = cos(norm(u))*eye(3)+sin(norm(u))
                    *A(u./norm(u))+(1-cos(norm(u)))/(norm(u)^2)*(u*u');
  end
  R = [data(i-1,1) data(i-1,2) data(i-1,3) ;
         data(i-1,4) data(i-1,5) data(i-1,6) ;
         data(i-1,7) data(i-1,8) data(i-1,9)];
  R = (expo*R)';
  data(i,:) = [(R(:))' w' rkv(7) rkv(8)];
end
```

Literaturverzeichnis

Literatur

1. Johannes Budday. Numerische Behandlung der Rotation starrer Körper. Master's thesis, Universität Konstanz, 2009.
2. Johannes Schropp. Skript zur Vorlesung - Numerik gewöhnlicher Differentialgleichungen. `http://www.math.uni-konstanz.de/~schropp/numdgl/skript.pdf`, 2009.