Modellierung dynamischer Systeme
Abgabe der Praktikumsaufgabe 3

Maria Lüdemann und Birger Kamp

May 18, 2016

Contents

1 Teilaufgabe Schiefer Flipper

Im Vorlauf der Praktikumsaufgabe 3 haben wir uns dafür entschieden den schiefen Flipper zu simulieren.

Dafür soll die Bewegung einer Kugel auf einer geneigten Ebene simuliert werden. Die Ebene ist von drei Wänden begrenzt und hat in der Mitte ein zylindrisches Hindernis. Das Abprallen der Kugel von den Wänden und dem Hindernis verändert die Richtung ihrer Bewegung, dabei wird auch ihre Beschleunigung und somit Geschwindigkeit simuliert.

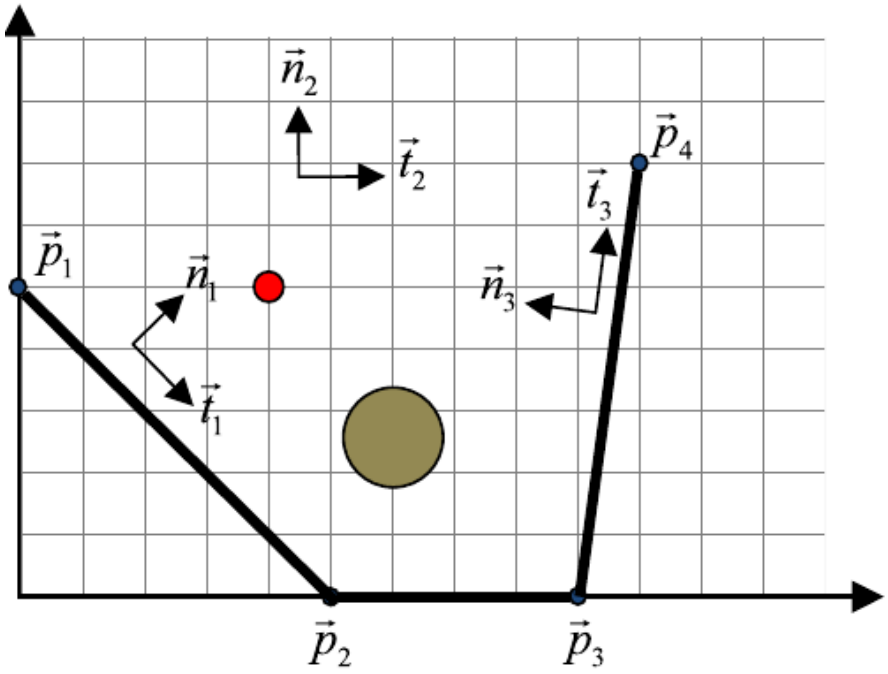

Figure 1: Skizze zum schiefen Flipper

1.1 Gegebene Formeln und Konstanten

Die begrenzenden Wände, durch Vektoren dargestellt definieren sich durch:

$$\vec{p_1} = (0,5)^T \ \vec{p_2} = (5,0)^T \ \vec{p_3} = (9,0)^T \ \vec{p_4} = (10,7)^T \tag{1}$$

Die Position und der Umfang des zylindrischen Hindernisses entsprechen:

$$\vec{p_Z} = (6,2.5)^T \ RHnd = 0.8 \tag{2}$$

Startposition der Kugel

$$\vec{xa}_0 = (4,5)^T \tag{3}$$

Startgeschwindigkeit der Kugel

$$\vec{v1}_0 = (0,0)^T \tag{4}$$

Radius der Kugel

$$R = 0.25 \tag{5}$$

1.2 Funktionen

Im Folgenden werden die verwendeten MatLab-Funktionen dokumentiert.

In der Betrachtung des Kontakts kann aus Sicherheitsgründen eine erweiterte Abfrage stattfinden. Ist die Kugel nach einer Reflexion noch immer im Reflektionsbereich kann es passieren, dass bei sehr kleinen Zeitschritten eine Abfrage erfolgt bevor die Kugel Zeit hatte sich aus dem Bereich heraus zu bewegen. In diesem Fall würde die Kugel erneut reflektiert werden was zur Folge hat dass,durch die Beschleunigung an der Stelle die Kugel in der Wand hängen bleiben kann. In unsere Modellierung und unseren Durchläufen ist dies nicht aufgetreten. Sollte ein solches Problem auftreten kann dies behoben werden in dem der Abfrage hinzugefügt wird, ob die Beschleunigung der Kugel in Vorwärtsrichtung der Wand erfolgt oder nicht.

1.2.1 Init

Die Funkion Init initialisiert die folgenden benötigten Größen:

- Wandeckpunkte $p1, p2, p3, p4$ und Hindernispositionen Hnd
- die Wand-Tangential $t1, t2, t3$ und Normalvektoren $n1, n2, n3$ die aus den Eckpositionen der Wände berechnet werden,
- Startposition $x1$ und Startgeschwindigkeit $v1$ der Kugel die gegeben sind

```
1 function Init
2
3 % Hindernis und Pfosten fuer Waende
4 Hnd = [6;2.5];
5 p1 = [0;5];
6 p2 = [5;0];
7 p3 = [9;0];
8 p4 = [10;7];
9
10 % Kugel
11 x1 = [4;5];
12 v1 = [0;0];
13
14 % Normale und Tangentiale der Waende
15 t1 = (p2-p1) / norm(p2-p1);
16 n1 = [-t1(2,1); t1(1,1)];
17 t2 = (p3-p2) / norm(p3-p2);
18 n2 = [-t2(2,1) ; t2(1,1)];
19 t3 = (p4-p3) / norm(p4-p3);
20 n3 = [-t3(2,1) ; t3(1,1)];
```

1.2.2 Acc

Die Funktion *Acc*() berechnet die Beschleunigung a der Kugel.

```
function [ax,ay] = Acc
ax = 0;
ay = -1;
```

1.2.3 Wand1Refl

Die Funktion Wand1Refl berechnet die Reflexion der Kugel wenn die Funktion Wand1Kontakt *true* zurück gibt. Dafür wird die aktuelle Richtung der Beschleunigung mit der Wandposition verrechnet und umgeleitet.

```
function Wand1Refl()

vt = v1' * t1;
vn = v1' * n1;

% Reflektierte Geschwindigkeit (mit Vektorisierung der Geschwindigkeit)
v1 = vt*t1 + (-vn*n1);
```

1.2.4 Wand2Refl

Die Funktion Wand2Refl berechnet die Reflexion der Kugel wenn die Funktion Wand2Kontakt *true* zurück gibt. Dafür wird die aktuelle Richtung der Beschleunigung mit der Wandposition verrechnet und umgeleitet.

```
function Wand2Refl()

vt = v1' * t2;
vn = v1' * n2;

% Reflektierte Geschwindigkeit (mit Vektorisierung der Geschwindigkeit)
v1 = vt*t2 + (-vn*n2);
```

1.2.5 Wand3Refl

Die Funktion Wand3Refl berechnet die Reflexion der Kugel wenn die Funktion Wand3Kontakt *true* zurück gibt. Dafür wird die aktuelle Richtung der Beschleunigung mit der Wandposition verrechnet und umgeleitet.

```
function Wand3Refl()

vt = v1' * t3;
vn = v1' * n3;

% Reflektierte Geschwindigkeit (mit Vektorisierung der Geschwindigkeit)
v1 = vt*t3 + (-vn*n3);
```

1.2.6 Wand1Kontakt

Die Wand1Kontakt Funktion berechnet ob die Kugel in ihrer Vorwärtsbewegung die Wand berührt. Dafür wird die Position der Kugel mit ihrem Radius und der Position der Wand abgeglichen und das Ergebnis als Boolean zurück gegeben.

```
1 function kontakt = Wand1Kontakt
2 kontakt = (((x1-p1)' * n1) <= R);
```

1.2.7 Wand2Kontakt

Die Wand2Kontakt Funktion berechnet ob die Kugel in ihrer Vorwärtsbewegung die Wand berührt. Dafür wird die Position der Kugel mit ihrem Radius und der Position der Wand abgeglichen und das Ergebnis als Boolean zurück gegeben.

```
1 function kontakt = Wand2Kontakt
2 kontakt = ((x1-p2)' * n2) <= R;
```

1.2.8 Wand3Kontakt

Die Wand3Kontakt Funktion berechnet ob die Kugel in ihrer Vorwärtsbewegung die Wand berührt. Dafür wird die Position der Kugel mit ihrem Radius und der Position der Wand abgeglichen und das Ergebnis als Boolean zurück gegeben.

```
1 function kontakt = Wand3Kontakt
2 kontakt = ((x1-p3)' * n3) <= R;
```

1.2.9 HndKontakt

Die Funktion HndKontakt berechnet den Kontakt der Kugel mit dem Hindernis. Dafür wird die Position und der Radius des Hindernisses mit der Position und dem Radius der Kugel abgeglichen.

```
1 function kontakt = HndKontakt
2 kontakt = norm(x1 - Hnd) <= RHnd + R;
```

1.3 Stateflow-Modell

Das folgende Modell veranschaulicht das Stateflow-Modell, das die Simulation durchführt.

Es gibt in diesem Modell nur einen Zustand, in dem erfährt die Kugel ihre Beschleunigung in Richtung *unten*. Das Modell arbeitet die Berechnungen der Simulation über die Transitionen ab, da es kein fundamentalen Veränderungen der Funktionalität des Modells gibt und somit nur ein Zustand notwendig ist. Es wird für jede Wand/Hindernis geprüft, ob die Kugel in Kontakt damit gekommen ist. Falls ja, dann wird berechnet in welche Richtung die Kugel umgelenkt wird.

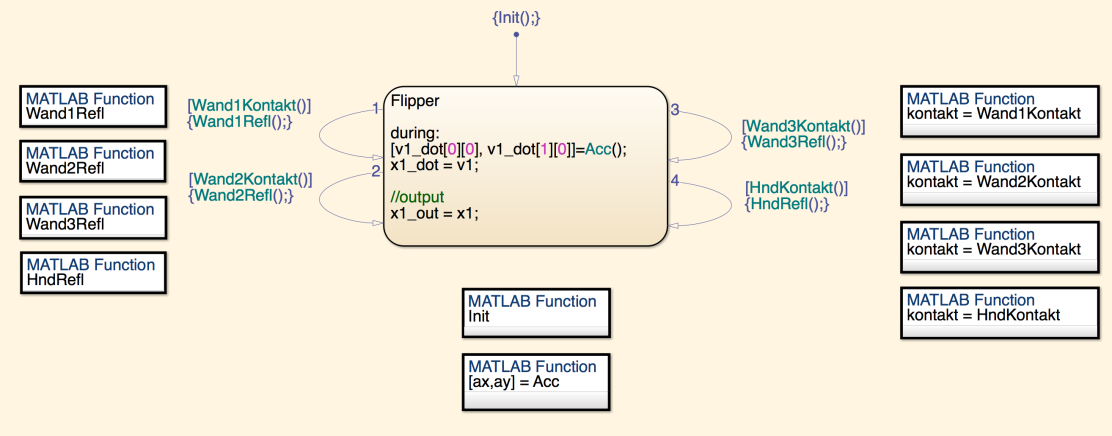

Figure 2: Stateflow Modell für den Flipper