Satellit - Raketen - Problem

Gennaro Piano
Philipp Schalcher

10. Juni 2012

Inhaltsverzeichnis

Kapitel 1

Einleitung

Auftrag ist es, eine Simulation eines Raketenstarts und flugs zu bauen. Die Rakete wird mit einer bestimmten Füllmenge eines Treibstoffes von einem Punkt auf der Erde starten. Das Programm soll aus den angegebenen Parametern eine Flugbahn berechnen, welche die folgenden Bedingungen erfüllt:

- Rakete kommt mit dem Treibstoff in den Orbit des Satelliten.
- Rakete ist am Ende des Fluges beim Satelliten angedockt und der Treibstoff ist verbraucht.
- Rakete und Satellit haben am Ende die gleiche Geschwindigkeit.

1.1 Erde

Die Erde befindet sich im Zentrum des Systems. Sie hat eine definierte Masse und eine definierte Gravitation. Beide sind konstant.

1.2 Satellit

Der Satellit befindet sich zum Startzeitpunkt in einer Umlaufbahn um die Erde. Er kreist in einer elliptischen Bewegung um die Erde. Wie die Erde ist dem Satellit eine definierte Masse zugewiesen. Ausserdem werden noch die Geschwindigkeit und die Position benötigt.

1.3 Rakete

Die Rakete steht zu Beginn an der Abschussrampe auf der Erde. Ihr ist ebenfalls eine Masse zugewiesen. Dazu kommen noch Treibstoffmenge (variabel), Verbrennungsgrad des Treibstoffes (variabel) und Position auf der Erde. Das Programm soll nun Geschwindigkeit, Richtung und Abschusszeitpunkt berechnen. Die Rakete wird dann diesen Weg abfliegen und so zum Satelliten gelangen, an welchem sie dann andocken wird.

1.4 Abgrenzungen

Die Körper sollen physikalisch korrekt zu einander reagieren und Effekte wie Massenanziehung beinhalten. Allerdings werden äusseren Einflüsse ausgeschaltet. Sprich die Sonne und sonstige ähnliche Einflüsse werden ignoriert.

Kapitel 2

Mathematische Beziehung

In diesem Abschnitt möchten wir mehr auf die mathematische Seite des Projektes eingehen. Dabei wird es in Satellit und Rakete unterteilt. Beide müssen bestimmte Voraussetzungen erfüllen, damit das Programm am Schluss die korrekte Flugbahn berechnen kann.

2.1 Satellit

Der Satellit bewegt sich in einer elliptischen Bahn um die Erde. Diese Bahn

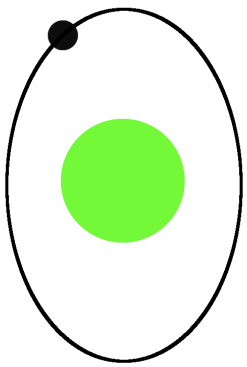

Abbildung 2.1: Erde mit Satellit in der Umlaufbahn

wird hauptsächlich von der Startgeschwindigkeit und der Massenanziehungskraft

der beiden Körper verursacht.

Zum Zeitpunkt der Ausgangssituation kennen wir die Formel für die Berechnung der Bahn nicht. Daher muss die Ellipse angenähert werden. Diese bewerkstelligen wir mit einem Polygon. Wir werden das Polygon durch die numerische Integration nach Euler erhalten.

2.1.1 Euler-Theorie

Die Integration nach Euler nähert einen Funktionsgraphen, wie schon erwähnt, durch ein Polygon an. Ein Polygon ist ein Körper, welcher aus vielen verschiedenen Datenpunkten besteht. Diese sind mit einer Geraden verbunden. Daraus folgt, je mehr Unterteilungen wir durchführen, desto genauer trifft das Annäherungspolygon den Funktionsgraphen. Allerdings brauchen wir für die Euler-Berechnung noch die Beschleunigung des Satelliten zur Zeit t. Der 0-Punkt (Punkt zur Zeit t = 0) kann durch einen Vektor in der Form (x_{sat}, y_{sat}, vx_{sat}, vy_{sat}). Dieser Vektor beschreibt die X- und die Y-Koordinate und die Geschwindigkeit v in X- und Y-Richtung. Für die Berechnung des neuen Punktes benötigen wir allerdings die Beschleunigung. Wenn wir nun den Vektor ableiten werden die Positionskoordinaten zu den Geschwindigkeiten in die jeweilige Richtung. Sprich der Vektor sieht danach so aus:

$$\text{Satellit} = (vx_{sat}, vy_{sat}, ax_{sat}, ay_{sat})$$

Wir sehen, dass die Geschwindigkeiten des Satelliten zur Beschleunigung werden. Die Beschleunigung können wir durch die Formel:

$$a_{x-Sat} = \frac{-g * m * (x_{Erde} - x_{Sat})}{|u|^3} \tag{2.1}$$

$$a_{y-Sat} = \frac{-g * m * (y_{Erde} - y_{Sat})}{|u|^3} \tag{2.2}$$

Nun haben wir die Informationen für Euler durch differenzieren erhalten.

Euler integriert nun immer den Graphen über eine bestimmte Strecke (t_0 bis t_{Ende}) in n Teilschritten. Die Anzahl der Teilschritte bekommen wir, wenn wir unser h (Unterschied zwischen t_0 und t1, t1 und t2, usw.) zuerst definieren, dann n berechnen durch (b-a)/h, dieses Ergebnis mathematisch korrekt runden (da n nur eine Ganze Zahl sein kann) und mit diesem n nochmals das korrekte h mit (b-a)/n ausrechnen. Nun können wir unseren

neuen Punkt des Satelliten ausrechen. Den erhalten wir durch die folgende Gleichung:

$$Satellit_{neu} = Satellit_{alt} + h * Satellit_{alt} \tag{2.3}$$

Sobald diese Rechnung durchgeführt wurde, wird t um h erhöht.

Da der Mensch diese kleinen Unterschiede bei der Darstellung gar nicht wahrnehmen könnte, wird der Satellit nur für ein klar unterscheidbares t (bis jetzt $t_0 + 0.1$) neu gezeichnet. Somit entsteht die korrekte Flugbahn.

2.1.2 Wichtige Codestellen

Calculation.java

```
/**
 * Ableitung des aktuellen Satelliten-Vektors
 * @param yAnfang
 * @param earth
 * @return res
 */
public double[] diff_sat(double[] yAnfang, Earth earth)
{
double[] z = new double[yAnfang.length];
double[] res = new double[yAnfang.length];
double[] u = new double[2];
double uBetrag,g,m;

g = 10; // Gravitationskonstante
m = earth.getMass(); // Masse der Erde

for(int i=0;i<yAnfang.length;i++)
z[i] = yAnfang[i];

u[0] = z[0] - earth.getPosx();
u[1] = z[1] - earth.getPosy();

uBetrag = Math.sqrt(u[0]*u[0] + u[1]*u[1]); //Distanz zwischen Satellit und Erde

//Ableitung von x- und y-Koordinate wird zur Geschwindigkeit
res[0] = z[2];
res[1] = z[3];

//Ableitung von x- und y-Geschwindigkeit wird zur Beschleunigung
res[2] = -g * m * (u[0]/Math.pow(uBetrag, 3));
res[3] = -g * m * (u[1]/Math.pow(uBetrag, 3));

return res;
}

/**
 * Polygon-Integration nach Euler zwischen tAnfang und tEnde mit einer
 * Aufteilung nach n. tEnde ist momentan immer tAnfang + 0.1
 * @param tAnfang
 * @param tEnde
 * @param yAnfang
 * @param n
 * @return
 */
public double[] euler(double tAnfang,double tEnde,double[] yAnfang, int n)
{
double h = (tEnde-tAnfang)/n; //korrektes h f"ur Unterschiede berechnen

double[] y = yAnfang;
double[] k;
double t = tAnfang;
```

```
for(int i=1;i<=n;i++)
{
k = diff_sat(y,new Earth()); //Ableitung des momentanen Vektors

y = addVector(y,multScalarVector(h,k)); //neuer Vektor berechnen
t = t+h;
}

return y;
}
```

2.2 Rakete

Die Rakete wird ebenfalls über Euler berechnet. Allerdings muss natürlich bei der Rakete eine andere Formel für die Beschleunigung genutzt werden. Die Beschleunigung der Rakete lässt sich folgendermassen ausrechnen:

$$a_x = (c * \frac{Verbrennung}{Masse_{Rakete}} * \frac{v_x}{|v|}) + (\frac{-g * m * (x_{erde} - x_{satellit})}{|u|^3}) \tag{2.4}$$

$$a_y = (c * \frac{Verbrennung}{Masse_{Rakete}} * \frac{v_y}{|v|}) + (\frac{-g * m * (y_{erde} - y_{satellit})}{|u|^3}) \tag{2.5}$$

Wir sehen nun, dass von der Beschleunigung der Rakete die Beschleunigung eines Satelliten abgezogen werden muss. Diese Kraft oder Beschleunigung wirkt in die Gegenrichtung der Rakete.

Die Rakete selber wird über die Formel

$$(c * \frac{Verbrennung}{Masse_{Rakete}} * \frac{v}{|v|}) \tag{2.6}$$

berechnet. C definiert einen Leistungskoeffizienten. Dieser sagt uns, wieviel Energie aus der Verbrennung des Benzins erhalten werden kann. Daher die Multiplikation mit der Konstante. Die Konstante Verbrennung beinhaltet also die Rate der Verbrennung pro Zeiteinheit. mmissile beschreibt die Masse der Rakete. Diese besteht aus der Masse des Benzines addiert mit der Leermasse der Rakete. Wir gehen zur Einfachheit davon aus, dass eine Einheit Benzin auch eine Einheit der Masse ist.

$|v|$ ist der Betrag der Geschwindigkeit. Berechnet wird der Betrag über v_x und v_y. Um nun den Abschusswinkel in die Berechnung einfliessen zu lassen, verwenden wir einen kleinen Trick. Richtigerweise müssten wir der Rakete als Startgeschwindigkeit von 0 in x-Richtung und 0 in y-Richtung. Allerdings können wir so die Richtung nicht wirklich bestimmen. Darum geben wir der Rakete eine minimale Geschwindigkeit in die jeweilige Richtung. Für den Start können wir mit dem Einheitskreis und Sinus und Cosinus arbeiten (siehe Abbildung 2.2).

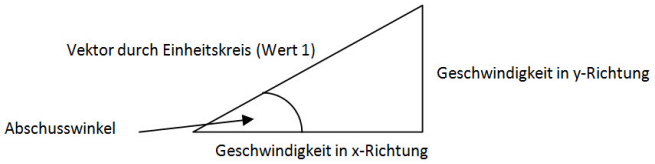

Abbildung 2.2: Berechnung erster Richtungsvektoren

So kriegen wir eine anständige Ausgangslage, mit der wir auch die Flugbahn besser berechnen können.

Der Rest der Berechnung läuft wieder gleich wie beim Satelliten. Die Rakete hat noch die Eigenschaft, dass sobald der Tank leer ist, sie zu einem Satelliten mutiert und dann die Satellitenbeschleunigung alleine gilt.

2.2.1 Wichtige Codestellen

Calculation.java

```
public double[] diff_mis(double[] yAnfang,double t, Earth earth, Missile missile){
double[] z = new double[yAnfang.length];
double[] res = new double[yAnfang.length];
double[] u = new double[2];
double uBetrag,vBetrag,m,k,mmissile;

m = earth.getMass(); // Masse der Erde
mmissile = missile.getMass() + missile.getTank();
for(int i=0;i<yAnfang.length;i++)
z[i] = yAnfang[i];

u[0] = z[0] - (earth.getPosx() + earth.getRad()/2);
u[1] = z[1] - (earth.getPosy() + earth.getRad()/2);

uBetrag = Math.sqrt(u[0]*u[0] + u[1]*u[1]); //Distanz zwischen Rakete und Erde
vBetrag = Math.sqrt(z[2]*z[2] + z[3]*z[3]);

k = 400; //Leistungskonstante

//Ableitung von x- und y-Koordinate wird zur Geschwindigkeit
res[0] = z[2];
res[1] = z[3];

//System.out.println(missile.getVx() + ", " + missile.getVy());
//Ableitung von x- und y-Geschwindigkeit wird zur Beschleunigung
if (missile.getTank() > 0.0) {
res[2] = -g*m*(u[0]/Math.pow(uBetrag, 3)) + (((k*missile.getVerbrennung())/mmissile)*(z[2]/vBetrag));
res[3] = -g*m*(u[1]/Math.pow(uBetrag, 3)) + (((k*missile.getVerbrennung())/mmissile)*(z[3]/vBetrag));
missile.setTank(missile.getTank()-(t*missile.getVerbrennung()));
}
else {
res[2] = -g*m*(u[0]/Math.pow(uBetrag, 3));
res[3] = -g*m*(u[1]/Math.pow(uBetrag, 3));
}
return res;
}

public double[] euler_mis(double tAnfang,double tEnde,double[] yAnfang, int n,Missile missile){
double h = (tEnde-tAnfang)/n; //korrektes h f"ur Unterschiede berechnen

double[] y = yAnfang;
double[] k;
double t = tAnfang;

for(int i=1;i<=n;i++){
k = diff_mis(y,h,new Earth(),missile); //Ableitung des momentanen Vektors

y = addVector(y,multScalarVector(h,k)); //neuer Vektor berechnen
t = t+h;
}
return y;
}
```

2.3 Lösungsansatz

Um das Ziel zu erreichen, dass die abgeschossene Rakete auch den Satellit trifft, stellen wir gewisse überlegungen auf. So müssen wir wissen, ob die Rakete überhaupt die Laufbahn des Satelliten erreichen kann. Wenn wir den Satellit einfach definieren, kann der Fall eintreten, dass die Rakete niemals die Punkte mit der Zielgeschwindigkeit erreicht. So wäre die Aufgabe natürlich auch unlösbar. Um diesem Problem entgegenzuwirken nehmen wir an, dass der Satellit zuvor noch eine Rakete war. Wir schiessen diese in eine Umlaufbahn. Rakete 1 hat also nun ihren Treibstoff verbraucht und ist zu diesem Zeitpunkt zu einem Satellit geworden. So wissen wir nun sicher, dass diese Laufbahn erreichbar ist.

Rakete 1 kreist nun um die Erde. Rakete 2 wird nun mit leicht anderen Werten gefüttert. Dabei soll nun von alleine ein Wert angepasst werden, damit schlussendlich die Umlaufbahnen übereinstimmt.

Das Ganze läuft also auf folgendes hinaus:

$$Rakete1 \begin{pmatrix} Pos_x \\ Pos_y \\ v_x \\ v_y \end{pmatrix} = Rakete2 \begin{pmatrix} Pos_x \\ Pos_y \\ v_x \\ v_y \end{pmatrix} \tag{2.7}$$

Und das zu einem Zeitpunkt t.

Der Ansatz in der Programmierung ist nun, die Vektoren der Raketen (mit Inhalt $\mathrm{Pos_x}$, $\mathrm{Pos_y}$, $\mathrm{v_x}$ und $\mathrm{v_y}$) von einander zu subtrahieren. Dabei muss als Resultat für jeden einzelnen Vektoreintrag der Wert 0 herauskommen. So können wir sagen, dass die Raketen mit der gleichen Geschwindigkeit und an der gleichen Position sind. Als Variablen im System sind daher die Treibstoffmenge, der Abschusswinkel, die Zeit zum Treffpunkt gelistet

Kapitel 3

Entwicklung

In diesem Teil betrachten wir primär die Entwicklung. Wir möchten diese geradewegs aufteilen auf die 4 Iterationen welche genutzt werden konnten. Zuerst aber werden wir aber unsere Zeitrechnung pro Iteration noch darlegen.

3.1 Aufwand pro Iteration

Vorgabe der Leitung waren 5 Stunden pro Woche. Die Teamgrösse ist auf 2 Personen begrenzt. Für die Fallstudie sind 12 Wochen eingeplant.

$$5 \text{ Stunden} * 12 \text{ Wochen} * 2 \text{ Personen} = 120 \text{ Stunden (ohne Abzüge)} \quad (3.1)$$

Die Velocity setzen wir auf 0.7, da dies ein guter Standard war.

$$120 * 0.7 = 84 \quad (3.2)$$

Da wir 4 Iterationen haben brauchen wir für eine Iteration 21 Stunden.

$$\frac{84 \text{ Stunden}}{4 \text{ Iterationen}} = 21 \text{ Stunden} \quad (3.3)$$

Zum Schluss ergeben sich daraus die Stunden pro Person.

$$\frac{21 \text{ Stunden}}{2 \text{ Personen}} = 10.5 \text{ Stunden.} \quad (3.4)$$

3.1.1 Iteration 1

Meilensteine

GUI erstellt, Klassendiagramm, Tasks, Projektplan,
Entwicklungsumgebung

Ablauf

Da uns schulischer Stoff noch fehlte um direkt mit der Mathematik zu beginnen, konzentrierten wir uns zuerst auf die Entwicklungsumgebung, die administrativen Arbeiten und das Erstellen der GUI. Wir entschieden uns für die Entwicklung in Java in Verbindung mit Maven und Github. Wir benutzen Maven für ein einheitliches Kompilieren mit externen Java-Libraries. Github wird unsere Versionskontrolle sein, damit wir uns gegenseitig nicht stören bei der Entwicklung.

Wir haben uns für diese Tools entschieden, da wir selber nicht als Entwickler arbeiten und uns andere Programme und Sprachen nicht wirklich geläufig sind. Im Kurs Methoden der Programmierung haben wir genau diese Tools näher angeschaut.

Unser Github ist zu finden unter
www.github.com

Wir senden unsere änderungen auf den Server durch folgenden Ablauf:

- git add .
- git commit -m 'Beschreibung der änderung'
- git push project master (project kann abweichen)

Für den Download gibt es den Befehl

- git pull project master

So haben wir nun die Speicherumgebung erstellt.

Maven benutzen wir dazu, eine einheitliche Kompilierungsumgebung zu bauen. So erstellen wir zuerst einmal ein Maven-Java-Projekt. Danach bearbeiteten wir die POM.xml und fügten unsere gewünschten Plugins hinzu (wie Javadoc, Testing, usw). Mit mvn package können wir nun

kompilieren lassen und Maven lädt geradewegs alle Plugins herunter. Unter dem Verzeichnis target wird dann ein ausführbares JAR-File erstellt.

Unsere nächsten Ziele waren dann die Tasks (per Scrumy) und der Projektplan. In Scrumy haben wir uns die Tasks folgendermassen aufgeteilt:

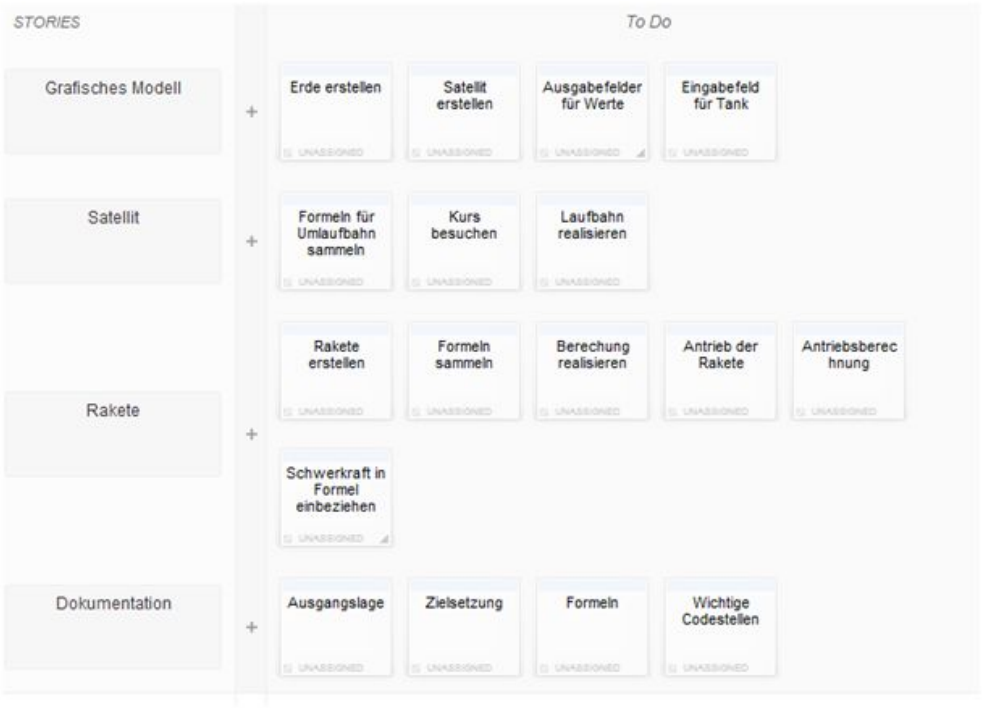

Abbildung 3.1: Scrumy-Tasks

Aus den Scrumy-Tasks wurde dann der Iterationsplan zusammengestellt.

1. Iteration	GUI (6.5 h)	Dokumentation (5.5 h)		Formeln sammeln (9 h)
2. Iteration	Beschleiunigung 5 h	Ableitung (5.5 h)	Dokumentation (5.5 h)	
3. Iteration	Integration 6 h	graphisch 3.5 h	Raketenbeschleunigung 5.5 h	Dokumentation 6 h
4. Iteration	Gleichung 7 h	graphisch 5 h	Doku 4 h	Präsentation 5.5 h

Abbildung 3.2: Iterationsplan

Die Planung des Projekts war bis jetzt also erfolgreich. Nun überlegten wir uns eine gute Möglichkeit, wie wir die Klassen zu designen hatten. Ganz getreu der Idee des objekt-orientierten Programmieren Entschieden wir uns für einen Aufbau, der Rakete, Erde, Satellit trennen sollte. So erstellten wir mal die aufgelisteten Klassen:

- Earth
- Satellite
- Missile
- App
- Animation

Nun hatten wir mal unser Grundgerüst. Aber wie sich später noch herausstellen sollte, mussten wir es doch noch stark erweitern. Aber dies erst in Iteration 2.
Für das GUI haben wir uns dieses Bild vorgestellt:

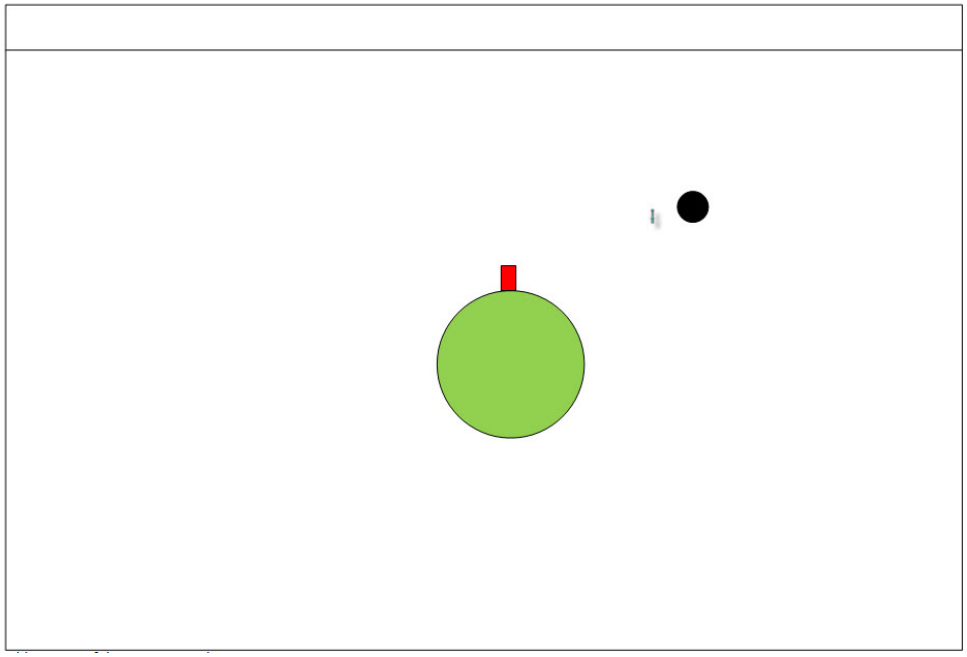

Abbildung 3.3: GUI

Der grüne Kreis stellt die Erde dar. Der Schwarze den Satelliten und das rote Rechteck die Rakete. Die Menüs werden im oberen Balken untergebracht. Wie diese allerdings aussehen sollten, haben wir uns noch nicht konkret überlegt.

Fazit

Iteration 1 ist beendet und wir haben unsere Ziele eigentlich alle erreicht. Wir haben allerdings kein Klassendiagramm erstellt. Die Hauptziele, ein GUI sowie die Planung zu haben, sind aber erreicht worden. Die Zusammenarbeit war gut und sollte es so weiter gehen, sollte das Projekt eigentlich keine Umstände bereiten.

3.1.2 Iteration 2

Meilensteine

Formeln für Berechnung verstehen, Berechnung Satellit und Rakete

Ablauf

Für die Iteration 2 wollten wir uns komplett auf die mathematischen Teile konzentrieren. Das heisst, in dieser Zeit wurde mehr auf Papier gearbeitet, als Programmcode geschrieben. Der Start war die Zusatzschulstunde, die Herr Heuberger gegeben hat, damit wir die Berechnung durch Euler schon kennen lernten. Er gab uns noch einige Beispiele wie er funktioniert. Allerdings mussten wir feststellen, dass uns jegliche Grundlagen in der Physik fehlten um die Flugbahn der zwei Objekte zu berechnen. So waren wir eine bis zwei Wochen damit beschäftigt, die Formel für die Beschleunigung zu suchen und das Ganze zu verstehen (leider waren die Java-Beispiele schlecht dokumentiert). Wir verabredeten uns mit Herr Heuberger um die Formel des Satelliten nochmals anzuschauen. Er half uns dann, nachdem wir die Situation geklärt hatten, weiter mit der Formel. Leider konnten wir diese aber nicht bis zum Iterationsmeeting einbauen, was dazu führte, dass wir nicht wirklich einen Fortschritt präsentieren konnten. Erfreulicherweise gelang es aber am darauffolgenden Tag den Satelliten korrekt kreisen zu lassen.

Fazit

Es braucht viel Zeit nur schon die Eulerintegration zu verstehen. Dieses Thema wird im Fach Numerik erst gegen Ende des Semesters angeschaut. Auch die physikalischen Gesetze des Satelliten (und natürlich auch der Rakete) waren uns nicht bekannt. Da wir in der ZHAW niemals diese Themen besprochen hatten, hatten wir starke Schwierigkeiten hier weiterzukommen. Im Meeting mit Herr Heuberger wurde uns aber nochmals alles erklärt und der Schleier hat sich gelüftet.

Im Iterationsmeeting hingegen stieen wir für diese Problematik auf taube Ohren. Vielleicht lag es auch daran, dass wir unter Zeitdruck standen, da auch noch Prüfung war.

Insgesamt war die Iteration 2 ein kleines Desaster und wir müssen nun in der Iteration 3 und 4 diese Fehler sehr schnell korrigieren.

3.1.3 Iteration 3

Meilensteine

Rakete eingebaut und flugfähig, Lösungskonzept begonnen und so gut wie möglich umgesetzt, Dokumentation auf aktuellen Stand gebracht

Ablauf

Nach dem Desaster aus Iteration 2 mussten wir uns neu organisieren. Der Satellit kreiste nun korrekt um die Erde. Wir hatten also mit einem Tag Verspätung das Konstrukt bereit. Wir setzten uns nun an die Formel für die Rakete. Gefunden haben wir die Raketengleichung. Allerdings stellte sich heraus, dass diese Formel viel zu kompliziert für unser Vorhaben war. Herr Heuberger war eine Woche nicht verfügbar. In dieser Zeit hatten wir hauptsächlich mit der Rakete gerungen. Die Idee, die Geschwindigkeit in X- und Y-Richtung durch Sinus und Cosinus berechnen zu lassen (da der Abschusswinkel ja eingegeben wird), stellte sich als gute Idee heraus. Nachdem Herr Heuberger wieder zurück war, setzten wir uns nochmals zusammen, damit wir die Formel von ihm so absegnen lassen konnten. über das folgende Wochenende wurde dann die Rakete fertig implementiert. Auch hatten wir schon die Lösungsidee begonnen einzufügen (2 Raketen, erste wird zum Satellit) um nur noch um das Treffen zu kümmern.

Probleme gab noch die versetzt startende Animation. Rakete 2 soll ja auf der Erde bleiben, bis sie weiss, wann sie abfliegen muss.

Fazit

Die Ziele der Iteration 3 wurden mehr oder weniger erreicht. Die Rakete war flugfähig und ein Lösungskonzept hatten wir auch. Die Dokumentation wurde ebenfalls erweitert. Allerdings sind wir auf die Tatsache gestossen, dass die Präsentationstermine (und damit das Ende der Projektzeit) um eine Woche nach vorne geschoben wurde. Grundsätzlich wurden unsere gesteckten Ziele aber erreicht.

3.1.4 Iteration 4

Meilensteine

Rakete 2 trifft Satellit, Dokumentation beendet, Präsentation

Ablauf

Wir haben uns für diese Iteration aufgeteilt. Einer konzentriert sich auf die Programmierung des Kollidierens der Raketen. Der andere kümmert sich um die Dokumentation und die Präsentation. Da die Dokumentation laufend erweitert wurde, haben wir hier nicht so einen grossen Klotz am Bein. Allerdings müssen wir die Präsentation noch erstellen. Das Treffen der beiden Flugkörper hingegen bereitet noch weiter Schwierigkeiten. Vor allem die überlegung, dass die Rakete auch in die gleiche Flugbahn geraten soll. Wenn der Abschusswinkel und die Tankfüllung verschieden zur ersten Rakete sind, dann wird die zweite Rakete nie in die gleiche Umlaufbahn kommen, ohne noch Steuerungselemente zu besitzen. Sie können also nur kollidieren.

Fazit

Das Projekt ging nun in die letzte Phase ein. Die Aufteilung war eine gute Idee, und so konnte die Dokumentation zum Beispiel von einem Word-Dokument noch in ein LaTeX-Dokument konvertiert werden. Die Präsentation findet am 15.06.2012 statt. Wir werden sehen, wie unsere Lösung ankommen wird.

Kapitel 4

Resultat, Fazit und Danksagung

4.1 Resultat

Am Beispiel des Satelliten mchten wir zuerst zeigen, dass wirklich eine elliptische Bahn entsteht mit unserer Formel. Da die Rakete nach dem Start sich in einen Satelliten verwandelt, gilt diese Beziehung auch fr diese. Allerdings wird die entsprechende Flugbahn anders verlaufen. Als Beispiel nehmen wir einen Satelliten, welcher schon in der Umlaufbahn ist. Er hat die Positionskoordinaten (0/60) fr X und Y und den Geschwindigkeitsvektor (50/20). Die Flugbahn soll von t_0 bis $t_{ende} = 80$ berechnet werden.

t	Pos_x	Pos_y	v_x	v_y
0	0	60	50	20
1	87.922	65.832	36.645	-6.683
2	150.101	46.087	26.451	-11.865
3	196.289	20.776	20.175	-13.148
4	232.068	-5.799	15.834	-13.323
5	260.334	-32.246	12.569	-13.081
6	282.784	-58.001	9.969	-12.654
7	300.504	-82.801	7.812	-12.136
8	314.238	-106.512	5.966	-11.569
9	324.521	-129.059	4.349	-10.974
10	331.748	-150.394	2.904	-10.359

Die Werte sind auf 3 Nachkommastellen gekrzt. Dieser kurze Ausschnitt

soll die Entwicklung zeigen. Wenn wir nun die Tabelle fortfhren bis t = 40, werden wir eine elliptische Bahn bekommen. Diese sehen wir, sobald die X- und Y-Koordinaten plotten.

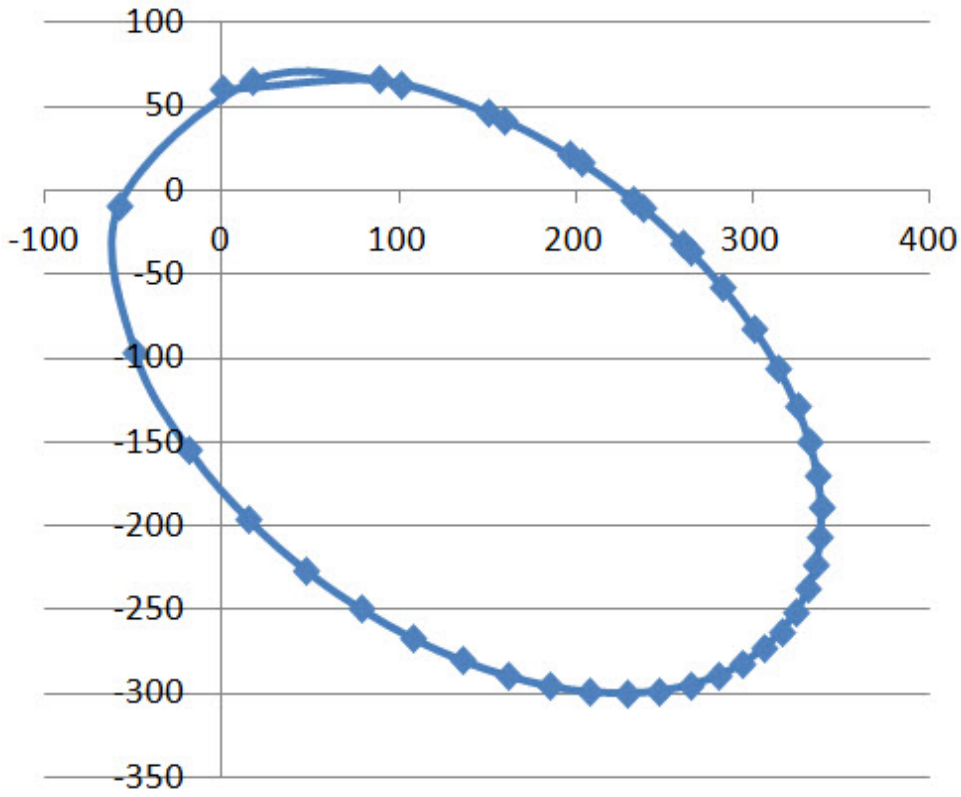

Abbildung 4.1: elliptische Bahn des Satelliten durch Euler-Integration

4.2 Fazit

Die Thematik des Projekts war sehr interessant. Die Motivation, das komplexe Thema zu schaffen war hoch. Leider unterschätzten wir die komplette Berechnung des Systems. So verloren wir viel Zeit während der zweiten Iteration. Die Theorie von Euler war für uns sehr abstrakt und am Anfang schwer zu verstehen. Die Programmierung des kreisenden Satelliten (und später der fliegenden Raketen) hingegen war schon eher

leicht. Daher konnten wir in der Iteration 3 Zeit wett machen und unser Projekt auf Vordermann bringen.
Die letzte Iteration lag (wie schon erwähnt) komplett in den Händen der Dokumentation und der Präsentation.
Schlussendlich kommen wir darauf, dass das Projekt sehr interessant war, uns einiges beigebracht hat was Mathematik an einem geschlossenem System, Planung und Realisierung eines Projekts und verschiedenen Tools angeht. Leider haben wir auch erfahren müssen, dass eine Leitung nicht immer wirklich gut leitet. Informationen wurden nicht weitergegeben und Meetings sehr kurzfristig und zu unmöglichen Zeiten angesetzt. Dies nur als Beispiel. Trotzdem war es toll, das gelernte mal gut, mal weniger gut anzuwenden.

4.3 Danksagung

An dieser Stelle möchten wir Herr Albert Heuberger danken. Er hat uns während der Projektarbeit tatkräftig unterstützt. Er war auch ausserhalb der Schulzeit bereit, mit uns mathematische Probleme anzuschauen und mögliche Lösungskonzepte aufzuzeigen. Besonders zu erwähnen wäre der zusätzliche Samstag, welchen er den Projektlern angeboten hat. Dort gab er uns die Chance, numerische Themen schon im Voraus zu betrachten, da diese erst gegen Ende Semester Teil des Stoffplanes waren. So konnten wir die Möglichkeiten gut ausschöpfen.

Literaturverzeichnis

[1] David Halliday, Robert Resnick, Jearl Walker; Halliday Physik Bachelor-Edition ; Verlag Wiley-VCH; ISBN 978-3-527-40746-0

[2] Grafiken zeichnen in SWING
www.javabeginners.de
Abrufdatum: 25. März 2012

[3] Maven Installation, Dokumentation
maven.apache.org
Abrufdatum: 19. März 2012

[4] Git Dokumentation
www.github.com
Abrufdatum: laufend (lezte 30.Mai.2012)

[5] Wikipedia
www.wikipedia.org
Abrufdatum: 26. April 2012