# Simulation

Jakob Klemm

20. September 2020

## Inhaltsverzeichnis



## 1 Einführung

Auch wenn sich die Flugbahn einer `Saturn V` gut in `Excel` oder einem ähnlichen Programm berechnen lässt, so sind diese Programme doch limitiert. Sowohl in ihrer Geschwindigkeit, als auch in den Funktionen und Einstellungen, welche für eine solche Simulation relevant sein könnten. Mit den wenigen Werten, welche uns für diese Simulation zur Verfügung standen, war es schnell möglich, neue Werte und Flugbahnen zu berechnen.

Und trotzdem ist es äusserst kompliziert, die Simulation interaktiv zu gestalten und dem Betrachter die Möglichkeit zu geben, selber die Kontrolle zu übernehmen. Dazu kommt ebenfalls noch die starke Systemgebundenheit solcher Dateien. Auch wenn `Excel` weit verbreitet ist und es genügend kostenlose Alternativen gibt, welche in der Lage sind, `Excel-Dateien` zu öffnen, ist

man doch an die Funktionen und Bedingungen dieser Programme gebunden. Obwohl einige *kostenlos* sind, findet man kaum *freie* oder gar *open-source* Varianten.

Die Formeln und Konzepte, mit welchen die Geschwindigkeiten und Flugbahnen einer `Saturn V` berechnet werden, lassen sich unter verschiedenen Bedienungen und für verschiedene Raketen oder Flugobjekte anwenden. Um den Prozess interaktiver und intuitiver zu machen, war es von Nöten, die Simulation als ein ausführbares, interaktives Programm zu implementieren.

# 2 Implementierung

Den Begriff *Raketensimulator* verbinden die meisten Menschen im ersten Moment mit Supercomputern und hoch komplexen Berechnungen. Trotzdem lässt sich das meiste dennoch auf einfache, lineare oder exponentielle Gleichungen bringen, die von Hand oder mit einem einfachen Taschenrechner gelöst werden können.

Daher war auch die Wahl der Programmiersprache eher offen, da die Berechnungen selbst nicht besonders aufwendig sind. Trotzdem gab es einige Kriterien, welche eine Programmiersprache für diese Aufgabe erfüllen musste:

- Die Fähigkeit Dateien zu lesen, zu interpretieren und zu schreiben.
- Einfache, lineare Algebra sowie häufige mathematische Funktionen.
- Zugriff auf externe Programme über eine standardisierte Schnittstelle.
- Kompilierung in eine alleinstehende, ausführbare Datei.

Auch wenn viele Sprachen und Programme zur Verfügung standen, fiel die Wahl auf `Golang`, eine Sprache die ursprünglich von `Google` entwickelt wurde. `Golang` erlaubt Entwickler schnelle, parallele und nebenläufige Programme systemnah zu schreiben.

## 2.1 Struktur

Obwohl der gesamte Code *open-source* und auf `Github` verfügbar ist, sollen einige Funktionen hier trotzdem besprochen werden, da die Codebase verwirrend sein kann.

### 2.1.1 Menü

Das Menü ist technisch zwar der am wenigsten interessante Teil, doch es ist ein äusserst relevantes Stück der Simulation. Basierend auf einfachen Konsolen-Printouts und Zahleninputs, macht dieses Menü die einfache Interaktion mit der Software möglich, ohne dabei besondere Anleitungen oder Erklärungen zu verfassen.

### 2.1.2 Einstellungen

Da Raketen unter verschiedenen Kriterien und in verschiedenen Situationen simuliert werden können, war es wichtig, die einzelnen Funktionen und Parameter sowohl einstellbar, als auch ausschaltbar zu machen. Dies geschieht über `settings.json`, einem einfachen `JSON-file`, worin alle Parameter und Einstellungsmöglichkeiten, sowie eine Liste der verfügbaren Umgebungen aufgelistet sind.
Über das Hauptmenü und die beiden Untermenüs ist es möglich, die Kriterien für den Flug zu bestimmen. Sobald diese feststehen, kann die Simulation gestartet werden.

### 2.1.3 Module

Da verschiedene Raketen verschiedene Treibstoffe, Bauarten, Triebwerke und Flugbahnen besitzen, war es ebenfalls wichtig, eine Software zu schreiben, die für verschiedene Situationen gewappnet ist. Im dedizierten `rockets/` Ordner können beliebige Konfigurationen und Modelle in Form von `JSON-Dateien` abgelegt werden, welche dann der Simulation zur Verfügung stehen.

## 2.2 Berechnungen

Auch wenn die `Raketengleichung` und andere Hilfsmittel zur Verfügung stehen, ist es doch weiterhin möglich, die Raketen schrittweise zu simulieren. Da jeder Schritt in einer solchen Simulation beinahe identisch abläuft, aber trotzdem von allen vorherigen Schritten abhängig ist, sowie alle nachfolgenden Schritte beeinflusst, war es nur logisch, die Simulation rekursiv zu implementieren.

Während das Berechnen der neuen Masse eine einfache Rechnung darstellte, mussten kompliziertere mathematische Konzepte verwendet werden, um die neue Geschwindigkeit zu berechnen. Zuerst wird über die Impulserhaltung die neue Geschwindigkeit ohne externe Kräfte, hier `ideal_vel`

genannt, berechnet. Danach werden damit die zusätzlich wirkenden Kräfte verrechnet.

### 2.2.1 Externe Kräfte

Mit der Impulserhaltung lässt sich lediglich die Geschwindigkeit oder Masse eines Objektes berechnen. Allerdings hat man dann noch keine Vorstellung von der Beschleunigung die auf ein Objekt wirkt. Allerdings stellt sich heraus, dass ein einfacher Trick verwendet werden kann, um diese Berechnung zu vereinfachen. Die Beschleunigung lässt sich wie folgt mathematisch berechnen:

$$\vec{a} = \frac{\triangle v}{\triangle t}$$

Da die Zeit im hier verwendeten Schrittverfahren pro Schritt immer `1` beträgt, kann man einfacher sagen:

$$\vec{a} = \triangle v$$

1. $\triangle t$ erlaubte es dann einfach, die extern wirkenden Kräfte, wie beispielsweise die Schwerkraft davon abzuziehen.

   $$a_n eu = t - 9.81$$

   . Die neue Beschleunigung lässt sich dann wieder gleich in eine Geschwindigkeit umrechnen und zur ursprünglichen Geschwindigkeit addieren, um die neue Geschwindigkeit zu erhalten.

2. Der Luftwiderstand war einiges komplizierter zu berechnen als die Schwerkraft. Zum einen mussten technische Probleme und Limitierungen berücksichtigt werden, wie beispielsweise die fehlende Präzision bei sehr kleinen Dezimalzahlen. Auch die fehlenden Daten und Werte stellten sich als grosses Problem heraus. Aus Zeitgründen wurden verschiedene Quellen gesammelt und verrechnet, anstatt die tatsächlichen Werte zu finden oder zu berechnen.

   (a) Tatsächlich gab es überraschend wenig Daten über die Form oder `c-Werte` von Raketen. Also wurde am Ende einfach der eines Kegels angenommen, also `0.75`. Obwohl dies bei weitem nicht Perfekt ist, ähneln die meisten Raketen in ihrer Form einem Kegel doch sehr, wodurch hier nur geringe Fehler entstehen sollten. In einer späteren Version des Simulators soll auch dieser Wert in den Raketeneinstellungen bestimmbar sein. Da sich der Wert aber bei

jeder einzelnen Stufe ändert und bei der `Saturn V` sogar während einer Stufe nicht konstant bleibt, war es nicht möglich, diese Funktion in absehbarer Zeit zu implementieren, wodurch `0.75` als Konstante gesetzt wurde.

(b) Auch bei der Berechnung des Luftdrucks kamen neue Probleme auf. Neben den ursprünglichen Problemen mit der korrekten Implementierung der Formel, gab es auch seltsame Fehler mit `Golang`. So musste am Ende für jede Höhe über `100km` (Karman-Linie) der Luftdruck auf `0` gesetzt werden, da sonst die Werte nicht mehr zu verarbeiten gewesen.

(c) Mit der Formel für den Luftwiderstand

$$F_l = \frac{1}{2} * A * c_w * p * v$$

lässt sich die aktuelle Kraft des Luftwiderstands berechnen. Diese muss dann allerdings noch durch die Masse der Rakete geteilt werden, um daraus eine Beschleunigung zu machen, welche dann wie oben beschrieben von der Geschwindigkeit abgezogen werden kann.

### 2.3 Raketen

Zwar waren bereits gute Werte für die `Saturn V` vorhanden, aber es stellte sich als überraschend kompliziert heraus gute Daten für die `Falcon 9` oder andere Raketen zu finden. Zum einen liegt dies an der Tatsache, dass `SpaceX` eine private Firma ist, welche natürlich nicht ihre gesamten Werte öffentlich macht, zum anderen liegt es aber ebenfalls an der `US-Regierung`, die den öffentlichen Zugang zu solchen Informationen erschwert, da diese oftmals als relevant für die nationale Sicherheit angesehen werden. Daher mussten für `Falcon 9` einige Annahmen und Schätzungen getroffen werden. Die Mehrheit der Daten stammten aber ursprünglich aus diesem inoffiziellen Reddit Post und ergeben tatsächlich Flugdaten, welche der echten Rakete ähneln.

### 2.4 Plotting

Die ursprüngliche Planung unserer Software enthielt die Absicht, die berechneten Flugdaten als Graphen zu exportieren. Aufgrund von zeitlichen und technischen Limitierungen mussten diese Funktionen allerdings weggelassen werden, sollen aber in einer späteren Version der Software implementiert werden. Aktuell werden die Flugdaten lediglich in der Konsole, sowie einem `CSV-ähnlichen` Format exportiert und gespeichert.

## 3 Resultate

TODO.