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 ihn den Funktionen und Einstellungen, welche für eine solche Simulation relevant sein könnten. Mit den wenigen Werten, welche uns für diese Simulationen 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 sich 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

Der Begriff "*Raketensimulator*"verbinden die meisten im ersten Moment mit Supercomputern und hoch komplexen Berechnungen. Trotzdem lässt sich das meiste auf einfache, lineare oder exponentielle Gleichungen bringen, die von Hand oder bereits 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 es Entwicklern 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

Technisch zwar der am wenigsten interessante Teil, ist es doch ein äusserst relevantes Stück der Simulation. Basierend auf einfachen Konsolen-Printouts

und Zahleninputs, macht es 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, sowie die beiden Untermenüs ist es möglich, die Kriterien für den Flug zu setzen. Sobald diese dann feststehen, kann die Simulation gestartet werden.

2.1.3 Module

Da verschiedene Raketen verschiedene Treibstoffe, Bauarten, Triebwerke und Flugbahnen besitzen, war es ebenfalls wichtig, die Software für verschiedene Situationen gewappnet zu strukturieren. 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 meist nur ein Handgriff entfernt sind, 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. Als erstes wird über die Impulserhaltung die neue Geschwindigkeit, ohne externe Kräfte, hier `ideal_vel` genannt, berechnet. Danach werden damit die zusätzlich wirkenden Kräfte verrechnet.

1. Beschleunigung Mit der Impulserhaltung lässt sich lediglich die Geschwindigkeit oder Masse eines Objektes berechnen. Allerdings hat

man dann noch keine Idee 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:

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

2.3 Raketen

2.4 Plotting

3 Resultate

Probleme genaue Daten für Falcon 9 zu bekommen (private firma, def. contract) -> grosse ungenauigkeit. cw: `http://www.staedtisches-gymnasium-wermelskirchen.de/sites/default/files/physik/Fall-Papierkegel-mit-Luftwiderstand.pdf` luftdruck: `https://wind-data.ch/tools/luftdichte.php?lng=de`