Rocket simulator: Ikarus

Giuseppe Luciano

Stefano Doria

INTRODUZIONE E MODELLO GENERALE

Il programma accompagnato da questa relazione è stato pensato per simulare il lancio di un razzo in una prima approssimazione che tenga però conto anche di non banali aspetti osservati sperimentalmente, soprattutto riguardo i motori.

Segue un elenco delle più rilevanti approssimazioni adottate: considerare l'aria un gas perfetto; assumere che i motori siano costantemente in uno stato di "steady-state", cioè operanti in condizioni di equilibrio ottimale, non considerando le fasi di avvio, "start-up", e spegnimento, "tail-off"; ignorare la corrosione nell'ugello dei motori, così come la variazione di temperatura e pressione all'interno della camera di combustione; non considerare effetti aereodinamici non elementari sul razzo.

Tutti i calcoli della simulazione derivano da elementari considerazioni di tipo discreto e dall'approssimazione del moto come uniformante accelerato su intervalli di un secondo, ovvero il tempo per il quale i motori sono stati pensati ed ottimizzati.

Molti aspetti, per esempio a livello di parametri da inserire, sono stati semplificati per rendere il software più accessibile, mentre si è cercato di migliorare altre funzionalità, come l'ottimizzazione della traiettoria. Su quest'ultima, viste le scarse trattazioni teoriche trovate, si è deciso di adattare i dati sperimentali di lanci alla nostra simulazione.

IMPLEMENTAZIONE

Per quanto riguarda l'aspetto di implementazione del codice, è stata organizzata la distribuzione, almeno per quanto concerne la simulazione, in due file di intestazione e file sorgente dei rispettivi eseguibili, tralasciando temporaneamente il discorso grafica, oltre al file sorgente dell'eseguibile principale. La decisione iniziale sulla struttura del codice è stata quella di dividere gli elementi legati al razzo, in un file nel namespace "rocket", da quelli relativi al resto della simulazione, in un altro file nel namespace "sim". È importante notare che tutti i nomi delle classi create iniziano con una lettera maiuscola, mentre quelli dei namespace e delle struct con un carattere minuscolo. Sempre per convenzione le variabili private o statiche terminano con "_".

Rocket

Partendo dal namespace "rocket", al suo interno è stata definita la classe "Rocket", che rappresenta la concezione dei parametri specifici del razzo. Qui si trovano metodi rappresentanti aspetti caratteristici del razzo, come la spinta o varie funzioni per restituire vari parametri, consentendo così a queste informazioni di essere utilizzate anche esternamente. Fuori dalla classe si trovano altre funzioni legate ai parametri dei razzi, ma che non riguardano direttamente il veicolo stesso, quali la forza gravitazionale e la resistenza aerodinamica.

Approfondendo la classe Rocket, questa può contare su un solo costruttore ma il livello di dettaglio nell'inserimento dei parametri viene scelto in fase di input, inserendo valori ragionevoli per i parametri omessi. Ciò permette anche a chi non conosce tutti i dettami relativi alle caratteristiche strutturali dei vettori di godersi comunque un'esperienza gradevole fin dal primo avvio lasciando però la possibilità agli utenti più informati di scegliere un grado di dettaglio superiore ma comunque non eccessivamente complicato.

Nella parte pubblica della classe "Rocket", abbiamo scelto di incorporare la classe "Engine". Questa scelta è stata dettata principalmente dall'evidente correlazione tra i due oggetti, che al momento del lancio agiscono come una singola entità. Tale decisione non è stata presa alla leggera, poiché l'aggiunta della classe "Engine" ha aumentato significativamente la complessità della classe "Rocket". Tuttavia, ciò contribuisce all'organizzazione del codice e semplifica il processo di manutenzione e gestione dello stesso, dato che la classe "Engine" è utilizzata esclusivamente all'interno di "Rocket".

La classe "Engine" è una classe astratta da cui derivano, attraverso l'utilizzo del polimorfismo, due ulteriori classi: "Ad_engine" e "Base_engine". Queste due classi offrono all'utente la possibilità di selezionare il modello da utilizzare nella simulazione. Infatti il motore è un aspetto molto importante della simulazione, dunque si vuole che gli venga permesso di essere scelto con grande dettaglio. Tuttavia, poiché non si conoscono in anticipo le scelte dell'utente, i vari oggetti sono gestiti tramite shared pointers al fine di minimizzare il rischio di creare "memory leak" (perdite di memoria). Anche in ciascuna di queste classi, gli utenti hanno la flessibilità di decidere quanti parametri inserire.

Chiudendo il discorso sul namespace "rocket", si fa notare l'utilizzo di "Vec" come alias per riferirci ad array di due double, in modo da rendere più chiaro il modello adottato. Questa scelta ci ha lasciati particolarmente soddisfatti e quindi si ritrova anche in molte altre parti del codice. Inoltre, sempre per essere il più immediati possibile, si è evitato l'utilizzo della parola chiave "auto".

Sim

Per quanto invece riguarda il namespace "sim", abbiamo creato al suo interno la classe "Air_var", dedicata a rappresentare le condizioni atmosferiche a una specifica altitudine. Non essendo strettamente correlata al razzo, tale classe non è stata inserita nel namespace "rocket". All'interno di questo namespace, è stata inclusa anche una struct denominata "cost", la quale raggruppa molte costanti fisiche utilizzate nel programma, le quali sono definite come "static constexpr", in modo che siano visibili in ogni parte del codice e che il codice sia più espressivo ed immediato. È importante notare che il nome "cost" presenta una connotazione italianizzata, in quanto il termine "const" avrebbe potuto causare confusione e problemi essendo un termine chiave del linguaggio c++.

Interface

Oltre a quelli già anticipati, c'è un altro file di intestazione insieme al sorgente associato, denominato "interface" (nome anche del namespace che raggruppa gli oggetti al suo interno), che funge principalmente da contenitore per tutte le componenti relative ai menu e a funzioni usate nella parte grafica. In riferimento ai menu, si è alla fine scelto di evitare un'interfaccia grafica, poiché, durante l'inizializzazione dei parametri, diventava caotica e rallentava il processo di inserimento dati, tanto che noi stessi preferivamo la semplice ma funzionale interfaccia a linea di comando.

All'interno di questo namespace, sono presenti anche strutture (struct) che agevolano la creazione di vari oggetti. Queste strutture sono state inizializzate in modo da non causare sovraccarichi sulla memoria in un secondo momento, in quanto sono state inizializzate all'interno di uno scope dedicato che si chiude prima dell'avvio.

Inoltre, per quanto riguarda i menu, si è scelto di evitare di creare enum class per gestire la navigazione tra le diverse schermate. Questa scelta, simile a quella che fatta per l'interfaccia a linea di comando, migliora l'esperienza complessiva, poiché non richiede di selezionare il menu ogni volta, consentendo di passare direttamente all'inserimento dei dati.

Le motivazioni che giustificano il separare le funzioni per la parte grafica in un altro eseguibile sono principalmente legate alla promozione della modularità e alla potenziale utilità come documentazione dell'interfaccia. Inoltre, vi sono vantaggi in termini di tempi di compilazione.

Un altro aspetto rilevante riguarda la distribuzione del codice all'interno del main loop. Abbiamo cercato di evitare che il loop diventasse troppo congestionato e quindi più difficile da gestire, pertanto si sono suddivise le operazioni in funzioni che a loro volta raggruppano attività specifiche. Un esempio di ciò è dato dalle funzioni come "set_state", le quali raccolgono le operazioni legate a un particolare compito. Questo approccio ci ha consentito di focalizzarsi meno sulla corretta esecuzione di queste funzioni durante il debug, visti i numerosi test a riguardo, semplificando la ricerca e la correzione degli errori e permettendo di concentrarsi sulle altre operazioni e calcoli

TEST E DATI SPERIMENTALI

In una prima fase si sono considerati diversi modelli teorici e tecniche di ottimizzazione da adottare, tuttavia diversi di questi hanno mostrato particolari criticità nella fase di implementazione. Di conseguenza si è preferito adottare modelli più ideali, reperendo le equazioni necessarie da fonti che trattavano di questo argomento. Quindi dopo aver provato a capire come manipolare tali strumenti matematici si sono fatti calcoli per vedere se i valori restituiti erano in linea i risultati documentati di razzi durante la fase di lancio. Quindi dopo essersi assicurati che le equazioni implementate si comportassero bene rispetto all’idea che intanto ci eravamo fatti, la parte di test è stata principalmente relativa al fatto che ogni calcolo venisse eseguito nel modo giusto. In particolare per la fase di test inizialmente è stato scritto un eseguibile in cui venivano lanciate tutte le funzioni in modo raggruppato per esaminare che tutto venisse eseguito come era consono rispettando if e cicli del caso. Successivamente abbiamo provato a combinare fra loro le diverse chiamate di funzioni per vedere come interagivano fra loro.

Naturalmente, riconoscendo che i test basati solo su valori prestabiliti non erano sufficienti a rilevare tutte le possibili anomalie, è stato eseguito il programma più volte, variando i parametri e gli input, e si è attentamente ispezionato il corretto funzionamento di ogni singola funzione. Inoltre, al fine di garantire una maggiore congruenza con le dinamiche di lancio dei razzi reali, si sono acquisiti dati approssimativi circa le prestazioni dei razzi reali. Questi dati hanno permesso di confrontare le performance dei nostri veicoli con quelle di video di lanci spaziali reperibili. Ecco un confronto tra i risultati ottenuti da questo simulatore e quelli registrati durante il lancio della sonda spaziale Euclid utilizzando il Falcon 9 come riferimento.

https://www.youtube.com/watch?v=NIwvwYVUuxg

Razzo simulato	Falcon 9
A 104 m velocità 124 km/h	A circa 100m velocità circa 120km/h
A 400 m velocità 230km/h	A circa 400m velocità circa 230km/h
A 619 m velocità 280km/h	A circa 600m velocità circa 280 km/h
A 700 m velocità 295 km/h	A circa 700 m velocità circa 304 km/h
A 874 m velocità 337 km/h	A circa 900 m velocità circa 341 km/h
A 1064 m velocità 353 km/h	A circa 1000 m velocità circa 360 km/h
A 1488 m velocità 433 km/h	A circa 1500 m velocità circa 445 km/h
A 1996 m velocità 490 km/h	A circa 2000 m velocità circa 530 km/h

In seguito, l'approssimazione tende a peggiorare poiché nei motori reali si passa dalla fase di accensione a quella di pieno regime, mentre i razzi nel simulatore non sono in grado di farlo, il che è determinato dalla complessità che ne sarebbe derivata altrimenti e dalla necessità di valutare parametri estremamente tecnici. Nel particolare test che stiamo descrivendo, i motori sono impostati in modo che fossero paragonabili all'accensione dei motori del Falcon 9 durante le prime fasi di volo, quindi copiandone la massa e considerando l’impulso dei motori Merlin all’avvio, decisamente minore di quello nella seconda parte del funzionamento. Ovviamente per noi che avevamo il pieno controllo di tutti i parametri è stato più semplice raggiungere tali risultati mentre per gli utenti che non possono inizializzare direttamente lo stesso numero di parametri potrebbe risultare una sfida più complessa ma tranquillamente raggiungibile.

Per quanto riguarda più propriamente la fase di testing, nell’eseguibile designato ogni funzione creata da noi viene chiamata almeno due volte verificandone il corretto comportamento in tutti i casi possibili. Per invece il testing della parte grafica, il feedback visivo costituiva in riscontro già ottimale per il funzionamento del codice, visto che non sono state implementate strutture eccessivamente complesse.

SINTESI ESECUZIONE PROGRAMMA

Trattando dell’esecuzione del programma, in seguito all’inizializzazione dei parametri, una volta entrati nel main loop, viene valutato se in quel momento specifico sono soddisfatte o meno alcune condizioni, come se ci si trova o meno in orbita.

Altro parametro che viene valutato, in fase ascensionale, è l’inclinazione del razzo. Per determinare ciò, si sono studiate le telemetrie dei razzi reali come riassunto nel file "analised.xlsx", i cui dati

sono stati presi da https://github.com/shahar603/Telemetry-Data. Utilizzando queste telemetrie, sono stati estratti valori ottimali basati su proporzioni specifiche al fine di determinare l'inclinazione più adatta da attribuire al razzo in base all'altitudine raggiunta.

Successivamente, il programma procede a determinare i parametri correlati alle condizioni atmosferiche a una specifica altitudine, basandosi su relazioni prese dal seguente link: http://www.centrometeo.com/articoli-reportage-approfondimenti/fisica-atmosferica/4142-.

Dunque il programma calcola la spinta dei motori. Per ragioni di concisione, poiché non è possibile entrare nel dettaglio delle complessità dei propulsori a razzo con brevità, per ulteriori informazioni sulle relazioni usate per questi calcoli rimandiamo il lettore alle fonti principali usate per questo simulatore e riportate alla fine della pagina, parte delle quali è contenuta nella cartella e altre reperibili su internet. Successivamente, il programma calcola i valori delle forze totali agenti, che includono la spinta del motore, la forza gravitazionale, la forza centripeta e la resistenza aereodinamica.

Una volta ottenuti questi valori, il programma procede con gli spostamenti, che seguono la legge dei moti uniformemente accelerati, una semplificazione necessaria data l'impossibilità di operare nel continuo. Infine, il programma si occupa di registrare valori dettagliati della telemetria su file appositi, consentendo agli utenti di ispezionare approfonditamente le performance dell'oggetto simulato. Di seguito il programma mette insieme le forze del motore e tutte le altre che abbiamo considerato, ovvero la forza gravitazionale, centripeta, la resistenza aerodinamica.

L’ordine di esecuzione di tali operazioni è legato al fatto che per poter spostare il razzo bisogna conoscere la velocità iniziale e di conseguenza l’ordine inverso renderebbe gli spostamenti eccessivi. Infine il programma si occupa di riportare i valori della telemetria su appositi file da ispezionare per vedere le performance dell’oggetto ipotizzato.

ISTRUZIONI PER COMPILAZIONE (LINUX), TESTING ED ESECUZIONE

Come prima cosa sarà necessario installare *SFML* (Simple and Fast Multimedia Library), se non è già stata installa, con questo comando:

```
sudo apt-get install libsfml-dev
```

Poi si utilizza il file CMakeLists.txt presente nella directory corrente per creare e configurare un’area di compilazione all’interno della cartella build con il seguente comando:

```
cmake -S . -B build
```

Quindi si esegue la compilazione dell’area presente all’interno della cartella *build* digitando sul terminale:

```
cmake --build build
```

Se tutto è stato svolto correttamente, nella cartella build dovrebbero essere stati generati gli eseguibili *rocket.t* e *rocket_test.t*, che costituiscono rispettivamente il programma del simulatore e i

test che lo riguardano. Prima di eseguire, per evitare errori nell'esecuzione del programma, si consiglia di copiare *theta_data.txt* nella cartella *build* con il comando:

```
cp theta_data.txt ./build/
```

Per eseguire i programmi infine sono da inserire i seguenti comandi:

```
./build/rocket.t

./build/rocket_test.t
```

INPUT OUTPUT

Come prima nota importante il programma non è progettato per essere user friendly, come suggeritoci nei laboratori al fine di dare più risalto alla descrizione fisica del progetto, dunque se si commette qualche errore si dovrà terminare e rilanciare il programma. Ciò è stato fatto anche per velocizzare la fase di input visto che non viene richiesto di selezionare i menù.

Dopo aver lanciato il programma, si chiede all'utente di inserire in input i dati per inizializzare il razzo e i motori. Durante questo processo sul terminale verranno anche stampati dei valori indicativi per guidare l'utente e fare in modo che la simulazione possa procedere nel miglior modo possibile. Nonostante uno possa seguire pedissequamente i suggerimenti è comunque improbabile raggiungere l'orbita visto che sono forniti solo valori indicativi che si possono trovare mediamente nei veicoli a propulsione a razzo.

Non abbiamo intenzionalmente fornito valori che garantiscano risultati eccezionali, poiché riteniamo importante dare all'utente un senso dell'ordine di grandezza coinvolto e mantenere alto il livello di sfida. Raggiungere i propri obiettivi non è un compito di certo banale, nemmeno tentando piccole scorciatoie. Infatti durante le nostre prove, abbiamo trovato davvero poche configurazioni che portano ai risultati aspettati. Ma secondo noi è giusto così, dopotutto se fosse semplice creare velivoli capaci di andare in orbita vorrebbe dire che non stiamo simulando abbastanza bene la realtà.

Nello specifico, dopo aver inserito il nome del razzo, viene chiesto di determinare il modello matematico che si vuole usare per i motori che si dividono in base o advanced e di passare i parametri per inizializzarlo. Poi verrà chiesto di fornire i valori per altri aspetti del razzo, quali area laterale e superiore, massa della struttura, numero di stadi e di motori (il numero consigliato potrebbe sembrare basso, ma sono molto potenti), quantità di carburante gli stadi liquidi e solido (per tutti gli stadi viene fissata un'unica quantità). In realtà i parametri da inserire potrebbero anche risultare meno, poiché sia per il razzo sia per i motori è possibile decidere se si vogliono fornire molti valori, per avere più controllo, o pochi, e lasciare che il programma determini automaticamente certi dati. È comunque fortemente sconsigliato optare per i modelli più semplici visto che aumenta notevolmente la probabilità di distribuire la massa dei propellenti in modo errato facendo si che il secondo stadio si stacchi in concomitanza con il primo. Cosa inaccettabile per il nostro programma. Infine, verrà chiesta l'altitudine a cui si vuole orbitare, anche se si è notato che difficilmente l'orbita verrà raggiunta nella simulazione.

Nelle successive tabelle sono mostrati i parametri vengono chiesti e cosa significano.

Advanced engine	
Few	
Pressure	Pressione all'intorno del motore (solo in Few e Some), in pascal
Temperature	Temperatura all'interno del motore, in kelvin
Some Inclusi parametri precedenti (se non specificato altrimenti)	
Burn area	Area dove avviene la combustione, in metri quadrati
Throat area	Area gola, cioè sezione minore espulsione carburante, in metri quadrati
Many Inclusi parametri precedenti (se non specificato altrimenti)	
Grain dimension	Dimensione grani carburante, in metri
Grain density	Densità grani carburante, in kilogrammi su metro cubo
Coeff. a	È un coefficiente di burn rate
Coeff. n	È un coefficiente di burn rate
Molar mass	Massa molare del carburante, in grammi su moli

Base Engine	
Few	
Specific impulse	Impulso su unità di massa, in secondi
Coeff. of losing mass	Variazione di massa per unità di tempo, in kilogrammi al secondo
Some Inclusi parametri precedenti (se non specificato altrimenti)	
Pressure	Pressione all'interno del motore
Many Inclusi parametri precedenti (se non specificato altrimenti)	
Burn area	Area dove avviene la combustione, in metri quadrati

Rocket	
Few	
Name	Nome del razzo
Stages number	Numero totale di stadi, il primo solido, gli altri liquidi
Solid stage engines	Motori presenti per lo stadio solido
Liquid stage engines	Motori presenti per ogni stadio liquido
Some Inclusi parametri precedenti (se non specificato altrimenti)	
Propellant mass	Massa del carburante in ogni stadio, in kilogrammi
Many Inclusi parametri precedenti (se non specificato altrimenti)	
Lateral area	Area laterale del razzo, in metri quadrati
Upper area	Area superiore del razzo, in metri quadrati
Structure mass	Massa della struttura costituente il razzo senza stadi, in kilogrammi
Container mass	Massa di ogni stadio quando privo di carburante, in kilogrammi

Un altro input richiesto all'utente è l'altitudine indicativa a cui il programma cercherà di mettere in orbita il razzo. Le prestazioni migliori dovrebbero ottenersi per valori vicini ai 180 km di altitudine,

in quanto vicini all’altitudine che raggiungeva il razzo preso come riferimento per i dati sperimentali dell’angolo d’inclinazione.

Come già citato, è anche presente un file che fornisce i dati necessari per calcolare l’angolo di inclinazione del razzo in base all’altitudine durante la fase ascensionale.

Fatto ciò, si aprirà una finestra, programmata con SFML, pronta per mostrare graficamente la simulazione e che sarà divisa in quattro sezioni. In quella più a sinistra si osserverà uno sprite del razzo che cambia la sua angolazione, mentre lo sfondo si sposta verso il basso e cambia a seconda che ci si trovi o meno nell’atmosfera, simulando così il punto di vista del razzo stesso. Nella sezione superiore centrale si trova invece un’immagine della Terra, e un cerchio di dimensioni molto ridotte rappresentante il razzo; a simulazione avviata questo comincerà a spostarsi in modo da raggiungere l’altezza di orbita richiesta e poi mantenerla. A destra di ciò troveremo stampati alcuni dati interessanti: altitudine, angolo di inclinazione del razzo, velocità, stadio a cui ci si trova (se rimangono zero stadi, verrà mostrato zero), carburante rimanente nello stadio attuale e tempo trascorso. Infine, troviamo nell’ultima sezione in basso un planisfero, dove il razzo è rappresentato da un punto posizionato inizialmente nel nord del continente sudamericano e che poi si sposterà lungo l’equatore, mostrando la posizione relativa del razzo rispetto alla superficie terrestre.

Il programma non è progettato per stampare a schermo eccezioni nel caso di comportamenti considerati senza significato fisico per la simulazione. In un caso del genere scatterebbe un assert a far terminare un programma.

Infine, i dati della telemetria del razzo e delle condizioni atmosferiche vengono stampate sui file, accessibili all’utente per ulteriori analisi dettagliate.

ESITI

Nella simulazione è possibile osservare vari esiti per il volo del razzo, per esempio si possono supporre i seguenti:

- non riuscire a raggiungere l’orbita, o raggiungerla con una velocità tale da non riuscire a mantenerla, e dunque precipitare al suolo;
- raggiungere l’orbita con velocità compresa tra quella necessaria per un’orbita circolare e la velocità di fuga, e quindi compiere un’orbita ellittica o circolare;
- raggiungere l’orbita con velocità maggiore della velocità di fuga, e allora allontanarsi con traiettoria iperbolica o parabolica

Ovviamente tutti questi problemi si riscontrano solo nel caso in cui il razzo sia stato correttamente settato per raggiungere grandi quote.
Il raggiungimento dell’orbita era però più verosimile durante alcuni momenti dello sviluppo visto che dopo lunghe ore di riflessione eravamo quasi riusciti ad implementare funzioni che ottimizzavano al meglio la spinta al fine di trovarsi sempre nella fase di velocità ottimale. Sfortunatamente però tali tecniche non erano prive di controindicazioni e quindi si è stati costretti ad eliminarle nella versione definitiva del programma.

ULTIMI APPUNTI

Alcuni aspetti da tenere in considerazione nella simulazione sono i seguenti:

- arrivati alla quota orbitale prescelta, il razzo non mostrerà alcuna rotazione rispetto agli 0° raggiunti e per ogni successivo spostamento sarà puramente considerato come punto materiale;
- nella realtà si osserva che la velocità di un razzo è parallela al suo asse, e dunque quando cambia la sua inclinazione abbiamo adottato una correzione per far avvenire ciò; tuttavia questa correzione abbiamo deciso di usarla a partire da 20 km di altitudine, poiché al di sotto la forza di attrito risulta molto elevata e ridistribuire la velocità in base all'angolo del razzo causerebbe effetti problematici;

- come nei razzi reali, è inoltre da tenere a mente che il distacco del primo stadio liquido non può avvenire prima del distacco dello stadio solido, con cui opera contempo.

"A man's reach should exceed his grasp...else, what's the heavens for?"

FONTI

Una breve lista delle fonti in ordine di importanza per le idee prese:

:

- https://ftp.idu.ac.id/wp-content/uploads/ebook/tdg/DESIGN%20SISTEM%20DAYA%20GERAK/Fundamentals%20of%20Rocket%20Propulsion.pdf

- http://dma.dima.uniroma1.it:8080/STAFF2/pa18.pdf

- sito di un amatore nella costruzione di modellini di razzi molto dettagliato

  https://www.nakka-rocketry.net/

- https://webthesis.biblio.polito.it/secure/12834/1/tesi.pdf

- http://wpage.unina.it/astarita/Corsi/PA/2019/Pa10%20Razzi.pdf

Diversi altri di importanza minore da cui abbiamo tratto ispirazioni per modelli poi abbandonati.