

Università degli Studi di Padova – Dipartimento di Ingegneria Industriale

Corso di Laurea in Ingegneria Aerospaziale

***Relazione per la prova finale
«Lunar Starship HLS: Simulazione e
Analisi dell'allunaggio con Unity»***

Tutor universitario:

Prof. Giacomo Colombatti

Laureando: *Riccardo Ortes*

Padova, 15/07/2026

Il numero di missioni verso la Luna negli ultimi anni continua a crescere, come aumenta la necessità di allunare nel modo più efficiente e sicuro possibile nei prossimi anni. La NASA non si limita a replicare le imprese delle missioni Apollo con Artemis. Il suo obiettivo è invece quello di raggiungere la Luna e rimanervi.

Tra i **principali obiettivi** delle missioni ci sono:

- Tecnologici;
- Collaborazioni;
- Presenza a lungo termine;
- Conoscenze;
- Risorse;
- Competizione.



Il presente lavoro di tesi ha come obiettivo la creazione di un **simulatore** e l'analisi delle fasi della manovra di allunaggio «Gravity Turn» di un lander lunare, con riferimento a veicoli di nuova generazione come lo Lunar Starship Lander HLS sviluppato da SpaceX nell'ambito del programma Artemis della NASA.



Unity è fra le piattaforme di sviluppo in tempo reale e motori grafici più famosi e utilizzati al mondo. Sviluppato da Unity Technologies e lanciato nel 2005, nasce come strumento per la creazione di videogiochi, ma nel tempo si è evoluto diventando un pilastro fondamentale anche in altri settori. Un esempio è l'ingegneria, grazie ad ambienti virtuali ad alta fedeltà che rispecchiano la realtà. Unity gestisce la simulazione fisica usando una architettura classica ad oggetti (MonoBehaviour), basata sul motore grafico NVIDIA PhysX.



In questo contesto, il sistema affronta la dinamica dei corpi rigidi formulando problemi di Complementarità Lineare Mista (MLCP) per gestire l'evoluzione temporale, i vincoli e i contatti tra di essi.

L'Integratore: Semi-Implicit Euler

Da un punto di vista dell'integrazione temporale delle equazioni differenziali del moto, PhysX utilizza un integratore di Eulero semi-implicito.

$$v_{t+\Delta t} = v_t + a_t \cdot \Delta t$$

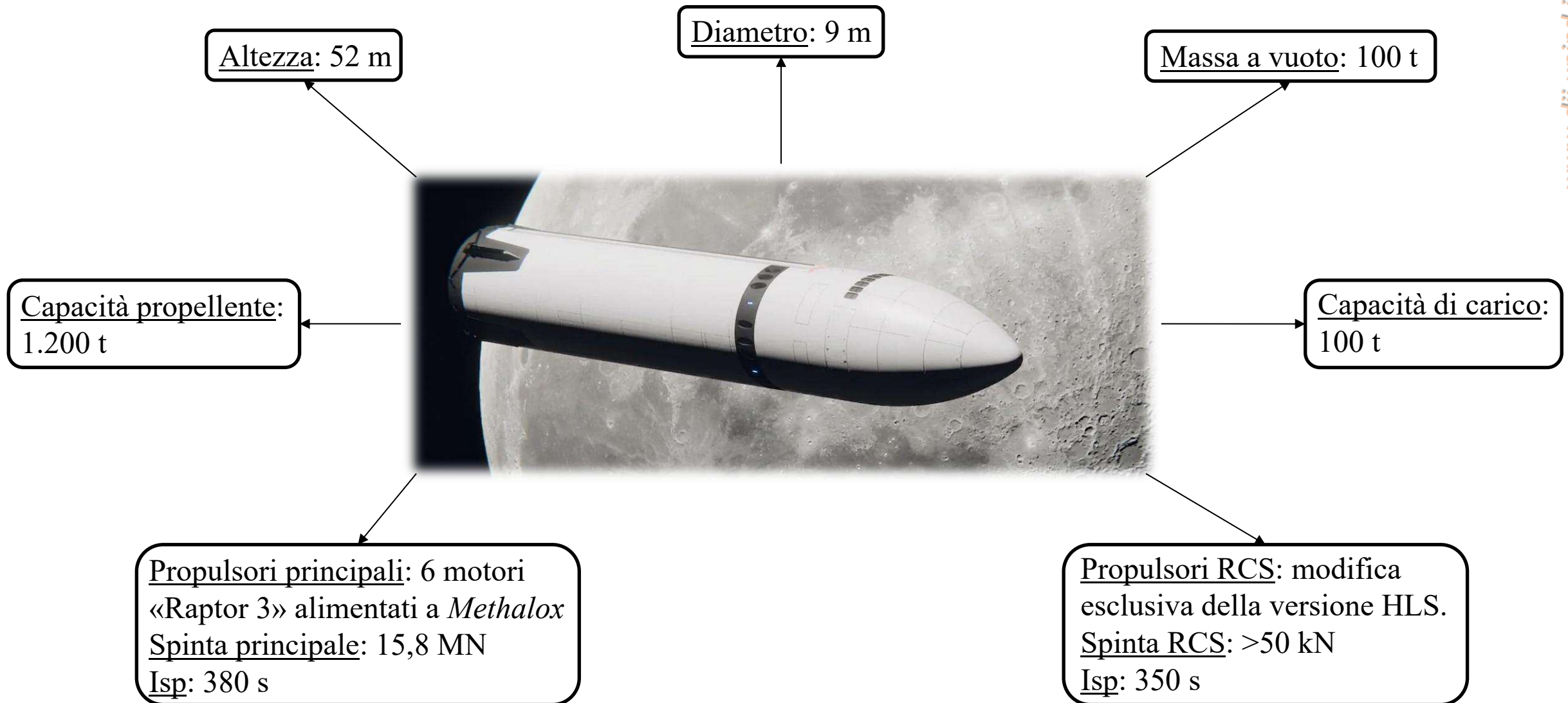
$$x_{t+\Delta t} = x_t + v_{t+\Delta t} \cdot \Delta t$$

Il Solutore di Vincoli: Projected Gauss-Seidel (PGS)

$$\lambda_i^{(k+1)} = \frac{1}{A_{ii}} (b_i - \sum_{j < i} A_{ij} \lambda_j^{(k+1)} - \sum_{j > i} A_{ij} \lambda_j^{(k)})$$

$$\lambda_i^{(k+1)} = \max(0, \lambda_i^{(k+1)})$$

Il costo computazionale di entrambi è $O(N)$.



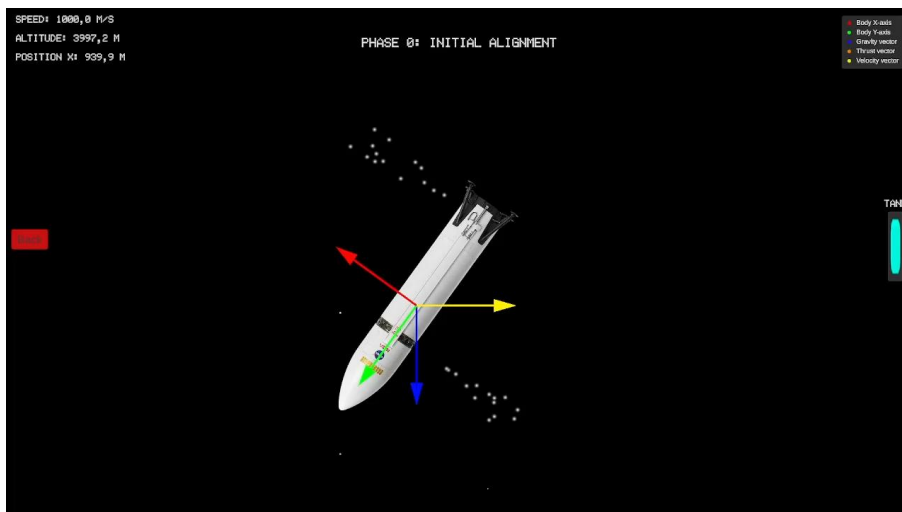
Le **assunzioni** prese durante lo sviluppo del simulatore sono:

- Centro di massa perfettamente centrato nell'HLS;
- Modello della gravità «Central body» calcolato istante per istante;
- Superficie della Luna piatta, senza crateri e senza regolite;
- Raggio medio della Luna considerato: 1737,4 km;
- Distanza tra il baricentro e il punto di applicazione della spinta RCS: 20 m;
- Calcolo dell'inerzia lungo l'asse z dell'HLS come se fosse un cilindro.

$$\vec{F}_{gravità} = G \frac{(m_{Lander} \cdot m_{Luna})}{(r_{Luna} + h)^2}$$

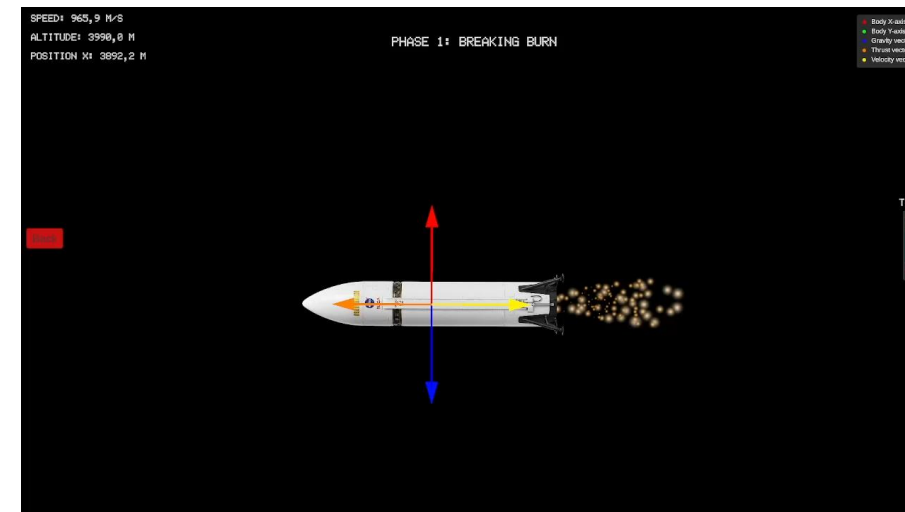
$$I_z = \frac{1}{12} \cdot m_{Lander} \cdot (3 \cdot r_{Lander}^2 + h_{Lander}^2)$$

Fase 0: Initial Alignment



```
// Script fase 0:
velAngMax = ((torque_RCS_max) / rb_Lander.inertia) * Time.fixedDeltaTime;
velAngTarget = Mathf.Sign(eroreAngolo) * velAngMax;
eroreVelAng = velAngTarget - rb_Lander.angularVelocity;
coppia = eroreVelAng * rb_Lander.inertia / Time.fixedDeltaTime;
coppia = Mathf.Clamp(coppia, -(torque_RCS_max), (torque_RCS_max));
if (!motoreSenzaCarburante) {
    rb_Lander.AddTorque(coppia);
    rcsImpegnato = true;
}
```

Fase 1: Breaking Burn



```
// Script fase 1:
if (!motoreSenzaCarburante) {
    rb_Lander.AddRelativeForce(Vector2.up * thrustMain);
    throttleMain = 1f;
    MostraVettoreThrust(vettoreThrustMain, puntaThrustMain, Vector2.up, true, 1f);
    MostraVettoreThrust(vettoreThrustRCS, puntaThrustRCS, Vector2.left, false, 0.5f);
}
```

Fase 2: Pitch Up



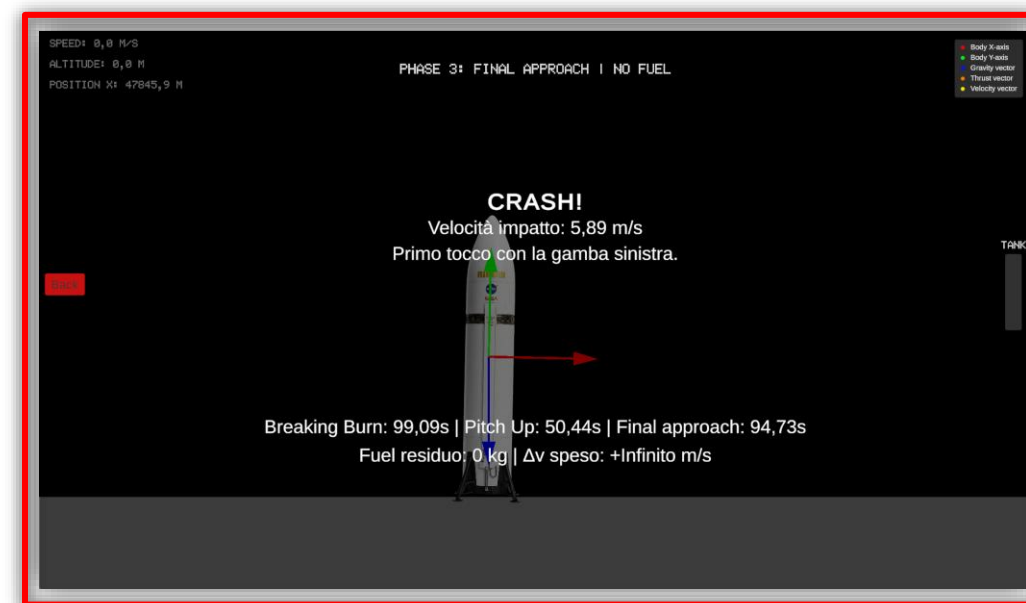
```
// Script fase 2: Velocità angolare massima raggiungibile dalla coppia RCS
velAngMax = (torque_RCS_max / rb_Lander.inertia) * Time.fixedDeltaTime;
velAngTarget = (angoloNorm > 0) ? -velAngMax : velAngMax;
erroreVelAng = velAngTarget - rb_Lander.angularVelocity;
coppia = erroreVelAng * rb_Lander.inertia / Time.fixedDeltaTime;
coppia = Mathf.Clamp(coppia, -torque_RCS_max, torque_RCS_max);
if (!motoreSenzaCarburante) {
    rb_Lander.AddTorque(coppia);
    rcsImpegnato = true;
}
MostraVettoreThrust(vettoreThrustRCS, puntaThrustRCS, Vector2.left, false, 0.5f);
MostraVettoreThrust(vettoreThrustMain, puntaThrustMain, Vector2.up, false, 1f);
```

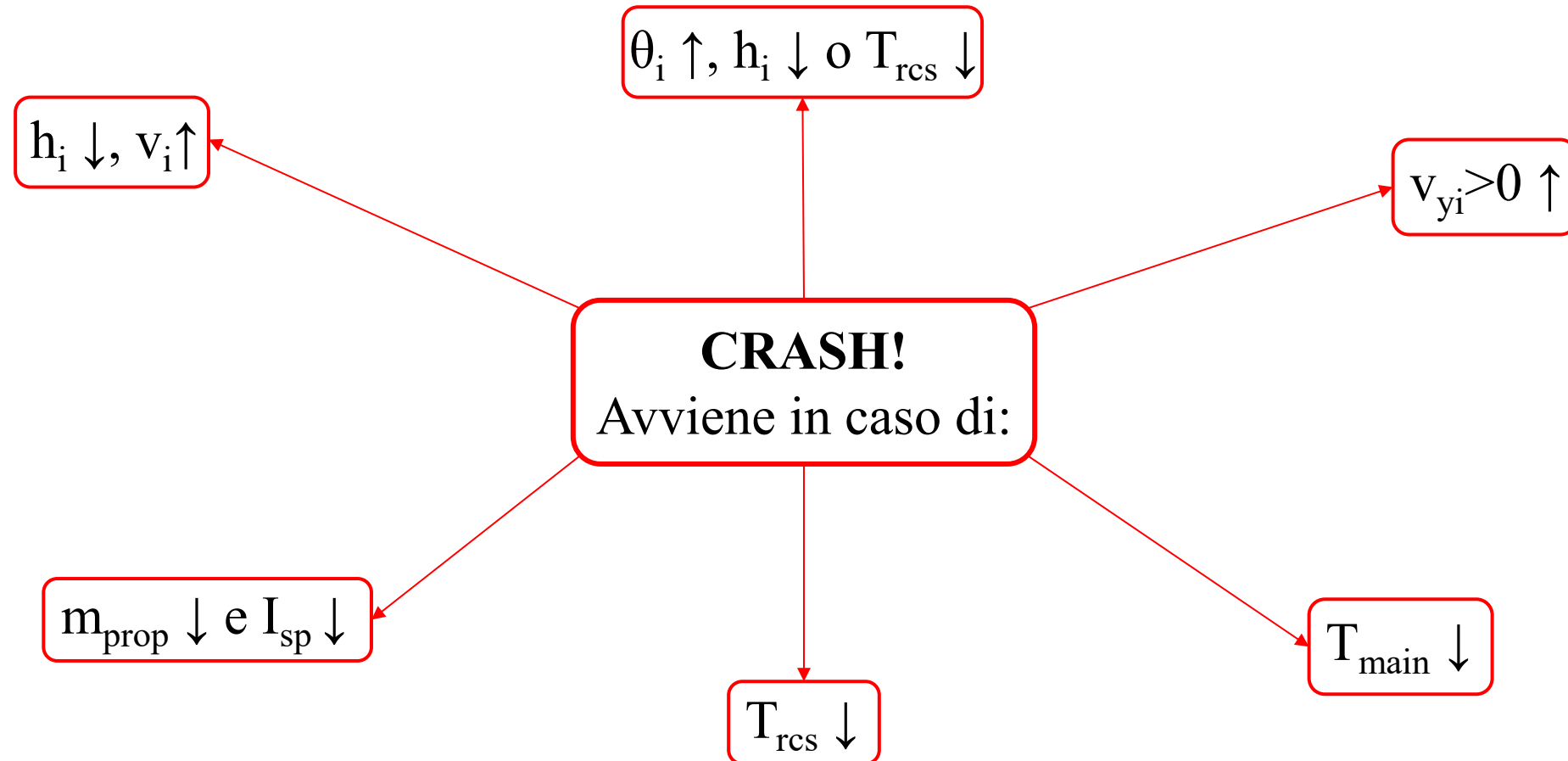
Fase 3: Final Approach



```
// Script fase 3:
vy = rb_Lander.linearVelocity.y;
velTarget3 = -Mathf.Lerp(1f, 60f, altitudine / 1000f);
erroreY = velTarget3 - vy;
if (erroreY > 0f) {
    regolatore = Mathf.Clamp01(erroreY / 3f);
    if (!motoreSenzaCarburante) {
        rb_Lander.AddRelativeForce(Vector2.up * (thrustMain * regolatore));
        throttleMain = regolatore;
        MostraVettoreThrust(vettoreThrustMain, puntaThrustMain, Vector2.up, true, regolatore);
    } else {
        MostraVettoreThrust(vettoreThrustMain, puntaThrustMain, Vector2.up, false, 1f);
    }
}
}
```

Allunaggio riuscito o no?





Formule usate per la gestione del carburante:

$$\frac{dm_{prop}}{dt} = \frac{T}{I_{sp} \cdot g_0}$$

$$\frac{\Delta m}{m_i} = 1 - e^{-\frac{\Delta v}{I_{sp} \cdot g_0}}$$

Equazione ideale del razzo
o equazione di Tsiolkovsky

```
// Consumo carburante istante per istante
void ConsumaCarburante() {
    if (motoreSenzaCarburante) return;
    float consumo = (portataMain * throttleMain * Time.fixedDeltaTime)
        + (rcsImpegnato ? portataRCS * Time.fixedDeltaTime : 0f);
    massa_prop -= consumo;
    if (massa_prop <= 0f) {
        massa_prop = 0f;
        motoreSenzaCarburante = true;
    }
    rb_Lander.mass = massa_landar + massa_prop;
}
```



Difficoltà incontrate:

- Imparare utilizzo di un nuovo ambiente di sviluppo per applicativi
- Reperire modelli CAD e informazioni specifiche sul lander usato
- Programmare il controller

Sviluppi futuri:

- 3D (con scelta del sito di allunaggio)
- Calcolo della gravità con armoniche
- Gestione dell'inerzia più realistica
- Animazione ed effetto fisico dell'estensione delle Landing Leg
- Utilizzo di un suolo lunare più realistico con annesse correzioni finali nella fase ultima di discesa
- Ottimizzazione della traiettoria
- Output a grafici/tabelle dei dati di simulazione

Bibliografia:

- "Orbital Mechanics for Engineering Students" di Howard D. Curtis
- S. Creech, K. Chojnacki, R. Ortega, T. K. Percy, M. Van Dyke, T. Pearson, K. Criswell, T. Polsgrove, T. Lockett, J. Stott, and B. Perry, "Update on NASA's Human Landing System (HLS) Program: Public-Private Partnership Advancing Artemis Sustainable Lunar Exploration," IAC-25,B3,1,7, 76th International Astronautical Congress, 2025.

Sitografia:

- <https://docs.unity.com/en-us>
- <https://www.spacex.com/vehicles/starship>
- <https://www.rmg.co.uk/stories/space-astronomy/nasa-moon-mission-artemis-program-launch-date>
- <https://learn.unity.com/tutorial/what-is-unity>
- <https://docs.unity3d.com/Packages/com.unity.physics@6.5/manual/index.html>
- <https://nvidia-omniverse.github.io/PhysX/physx/latest/docs/RigidBodyDynamics.html>

Grazie per l'attenzione