Pojednostavljena interaktivna simulacija protuoklopnog vođenog raketnog sustava primjenom pogonskog sustava Godot

Seminarski rad iz kolegija "Interaktivni simulacijski sustavi"

Leon Katić, Matija Bujanić, Ana Kasanić, Stjepan Džidžić, Josip Rončević
26. siječnja 2026.

Djelovođe: Leon Katić, Josip Rončević, Matija Bujanić, Stjepan Džidžić, Ana Kasanić

***Sažetak* – Rad predstavlja raščlambu literature o simulatorima protuoklopnih vođenih raketnih sustava, uključujući razne aspekte njena razvoja te određene elemente praktične realizacije. Pregledava se povijesni razvoj raketnih sustava koji su ,,tema'' simulacije, matematički modeli kojima se simulacija dovodi na visoku razinu realističnosti i praktičnosti, sustavi lociranja obuhvaćeni tim modelima te modularna arhitektura simulacije koja osigurava brojna unapređenja sustava. Prate se koraci razvoja sintetičkog okoliša unutar simulacije te navode mogući implementacijski alati za njeno ostvarenje. Pokazuju se određeni elementi implementacije u modernom *game engineu* Godotu i prilaže se razgovor s velikim jezičnim modelom o izrađenu sustavu.**

1 Uvod

Simulacije vojnih sustava predstavljaju jedan od najvažnijih alata u modernom vojnom inženjerstvu. One omogućuju operatorima, inženjerima i ostalim stručnjacima da (za razliku od testiranja pravom opremom u stvarnom okruženju) relativno jeftino i sigurno ispitaju i shvate kako sustavi koje proučavaju funkcioniraju. U ovakvim sustavima također je moguće promatrati razne scenarije kod upravljanja sustavom te tako pružiti optimalan trening upravljanja sustavom budućim operatorima. Protuoklopni vođeni raketni sustavi predstavljaju jednu od najvažnijih vojnih tehnologija razvijenih u modernom dobu. Riječ je o naprednim sustavima namijenjenima uništavanju (najčešće oklopljenih) ciljeva na nekoj udaljenosti (npr. tenka), s relativno visokim zahtjevima za preciznost. Visoka cijena i nepraktičnost testiranja u stvarnom okruženju čini ih popularnom ,,temom'' raznih simulacijskih sustava.

Razvojem raznih digitalnih alata simulatori postaju sve sofisticiraniji, pružajući operatorima u treningu zaokupljajuće i realistično iskustvo upravljanja pravim sustavima. Jedan od popularnih alata za konstrukciju simulacijskih sustava je i Godot, *open source engine* kojeg njegova fleksibilnost čini pogodnim za razvoj simulacija korištenih u mnoge svrhe, uključujući i vojne.

Ovaj seminar bavi se razvojem pojednostavljene interaktivne simulacije protuoklopnog vođenog raketnog sustava u Godotu. Fokus je na stvaranju realističnog, ali računalno pristupačnog modela koji prikazuje ključne karakteristike ovakvog sustava – lansiranje projektila, njegovo ponašanje pri letu i proces navođenja prema cilju. Simulacija uključuje fizikalne modele, vizualizaciju putanje rakete i sustav detekcije i uništenja cilja.

2 Uloge pojedinih članova tima

U sveukupnim poslovima na izradi seminarskog rada te pisanju ovog izvješća, članovi tima sudjelovali su na sljedeći način:

- Leon Katić - detekcija kolizije(sudar s tlom i sa tenkom), jednostavni let rakete, 3D modeli, kretanje tenka, trail rakete, game logic
- Ana Kasanić– vremenski uvjeti(magla, sunce i nebo, sjene, atmosfera), animacija eksplozija tenka
- Matija Bujanić - realistični let rakete, slow motion
- Stjepan Džidžić - audio
- Josip Rončević - dokumentacija

3 Pregled literature i arhitektura simulatora protuoklopnih vođenih raketnih sustava (s dodatkom praktične implementacije)

3.1 Vrste protuoklopnih vođenih raketnih sustava

Protuoklopni vođeni raketni sustavi (eng. *Anti-Tank Guided Missile* - **ATGM**) razvijali su se kroz nekoliko generacija, svaka sa specifičnim karakteristikama navođenja i upravljanja. Ovi sustavi klasificiraju se u tri osnovne generacije [2];

Prva generacija karakterizirana je potpunim ručnim praćenjem, gdje operater koristi optičke teleskope za praćenje i cilja i projektila. Druga generacija uvodi poluautomatsko navođenje (*Semi-Automatic Command to Line of Sight* - **SACLOS**). U ovim sustavima operater ručno prati cilj optičkim teleskopom, dok se raketa automatski prati pomoću infracrvenog (IR) senzora montiranog na lansirnu jedinicu u kombinaciji s teleskopom. IR senzor detektira zračenje iz izvora smještenog na stražnjem dijelu rakete, a sustav automatski generira upravljačke signale koji se raketi šalju putem žičane veze. Neki od poznatijih SACLOS sustava uključuju sovjetski *9M14 Malyutka* s maksimalnom brzinom od 115 m/s i efektivnim dometom od 500-3000 metara te *9K111 Fagot* s maksimalnom brzinom od 186 m/s i dometom od 70-2000 metara [4, 5].

Treća generacija karakterizirana je naprednim mogućnostima praćenja cilja: operater može koristiti optičke teleskope, televizijske kamere, laserske ili radio uređaje za automatsko ili ručno praćenje. Raketa se i dalje automatski prati kao u drugoj generaciji, ali se upravljački signali prenose bežičnom vezom umjesto žičanom.

3.2 Matematički modeli

Temelj većine simulatora ATGM sustava predstavlja dinamički matematički model rakete sa 6 stupnjeva slobode (6DOF). Ovaj model opisuje se kroz sustav nelinearnih diferencijalnih jednadžbi koje opisuju translacijsko i rotacijsko gibanje rakete kao krutog tijela [1].

Translacijske jednadžbe gibanja izražavaju se u referentnom sustavu brzine jednadžbom

$$m\frac{\mathrm{d}\vec{V}}{\mathrm{d}t} = \vec{F}_\mathrm{a} + \vec{F}_\mathrm{t} + G$$

gdje je *m* masa tijela, *V* je vektor brzine, *Fa* je vektor aerodinamičke sile, *Ft* vektor sile potiska (*thrust*), a *G* gravitacijska sila.

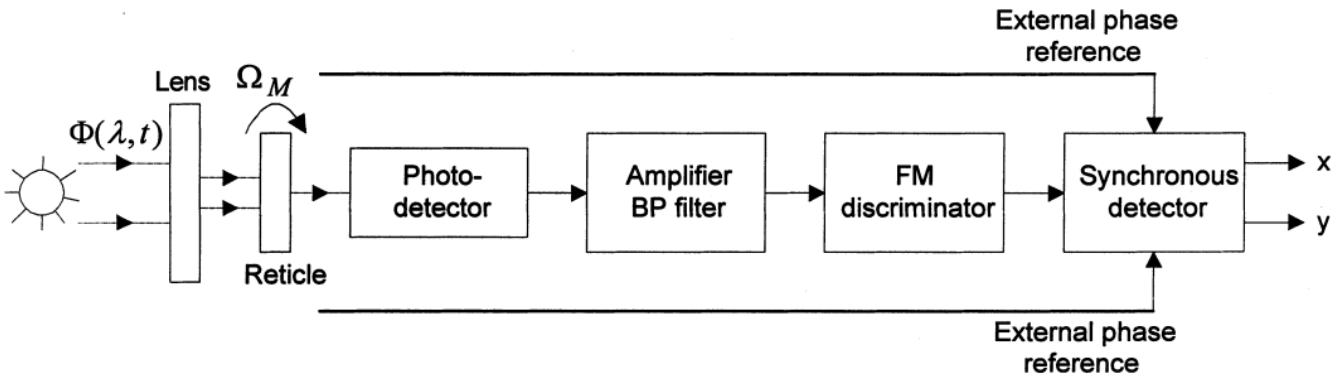
Rotacijske jednadžbe gibanja izražavaju se u koordinatnom sustavu vezanom uz tijelo rakete:

$$I\frac{\mathrm{d}\vec{H}}{\mathrm{d}t} = \vec{M}_\mathrm{a} + \vec{M}_\mathrm{t}$$

gdje je *I* tenzor inercije, *H* je kutni moment, *Ma* je aerodinamički moment, a *Mt* je moment potiska.

Prošle dvije jednadžbe definirane su u osnom sustavu rakete, no za kompletiranje matematičkog modela sustava za navođenje SACLOS-a potrebno je definirati/dodati jednadžbe o zakonima o navođenja i stvaranju naredbi, kontroli rotirajuće rakete, dinamike aktuatora te kinematičkih veza svih komponenti modela. Matrice transformacija između različitih koordinatnih sustava mogu se ostvariti korištenjem Eulerovih kutova (ψ, ϑ, φ) ili, alternativno, korištenjem kvaterniona (e_0, e_1, e_2, e_3). Pristup koji koristi kvaternione može biti efikasniji jer njegovih korištenjem izbjegavamo računanje transcendentalnih funkcija na koje često nailazimo kod korištenja Eulerovih kutova.

Infracrveni (IR) goniometar vrlo je važna komponenta za određivanje položaja rakete, pogotovo kod SACLOS sistema. Princip rada svodi se na hvatanje signala kojeg emitira infracrveni izvor signala na stražnjem dijelu rakete, dok se istovremeno ignoriraju/eliminiraju nepoželjni šumovi u pozadini. Sam IR goniometar (sistem koji detektira signal IR izvora) je pak baziran na napravi koja se naziva modulacijski disk. Dijelovi goniometra prikazani su na Sl. 1. Primljeni signal u polarnim koordinata modulira se prolaskom kroz modulacijski disk (*reticle*) sve dok FM diskriminator ne demodulira signal, pri čemu amplituda odgovara radijalnoj udaljenosti *r*, a faza signala kutnoj koordinati φ. Konačno, prolaskom kroz sinkroni detektor dobivamo kartezijanske koordinate rakete, tj. položaj u nama čitljivom obliku.

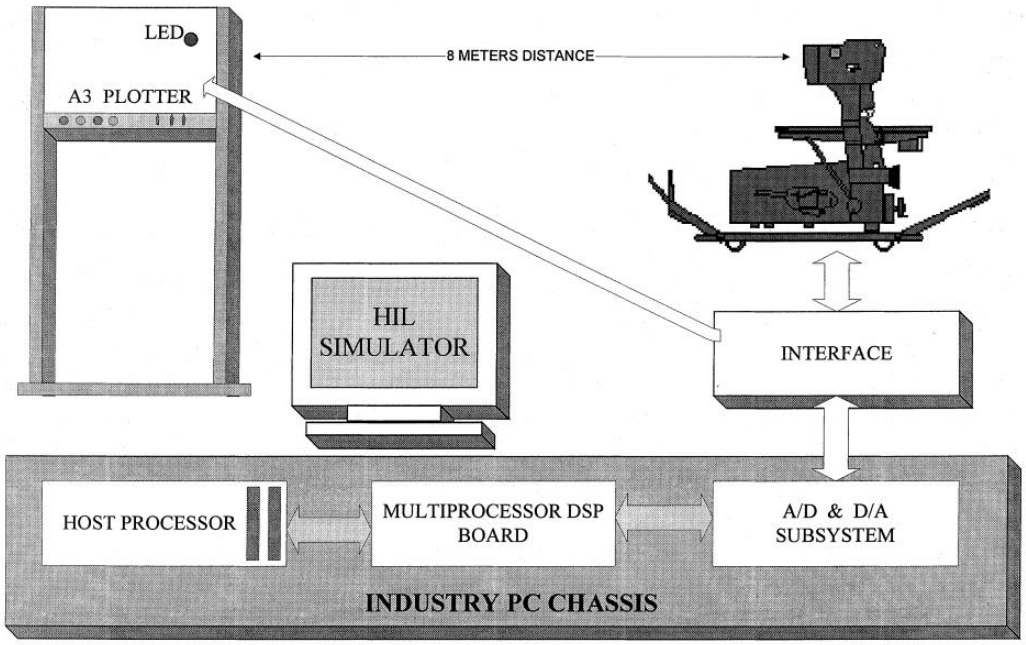


Sl. 1. IR goniometer za lociranje rakete, preuzeto iz [1]

3.3 Arhitektura simulatora

U ovoj podsekciji usredotočit ćemo se na koncept *hardware-in-the-loop* (HIL) pristupa arhitekturi. Hardverska struktura HIL arhitekture (Sl. 2) ostvaruje zatvorenu petlju kontrole i navođenja, što omogućuje realistično i modularno testiranje u svrhu parcijalnog razvoja i modifikacija SACLOS sustava.

Možemo općenito podijeliti ovu arhitekturu na tri dijela: multiprocesorsku računalnu tiskanu pločicu koja sadrži 4 TIM40 modula, na svakoj od kojih je processor s 6 COM portova, I/O podsustava koji podržava široki raspon frenkvencija (od guidance signala do FM signala) te visokobrzinskog plottera koji s LED-icom emulira pokretni IR izvor na vrlo malim udaljenostima u sklopu simulacije u laboratoriju. [1]



Sl. 2. Hardverska arhitektura HIL simulatora, preuzeto iz [1]

S druge strane, softverska arhitektura uključuje integrirano razvojno okruženje (IDE) razvijeno u Microsoft Visual C++ koje integrira TI C40 kompajler, asembler, linker i DB40 *debugger* [1]. Svaki model (6DOF dinamika rakete, zakon navođenja, digitalna koordinacija) izvršava se na zasebnom procesoru s vlastitim rokom izvršavanja.

3.4 Razvoj vizualne simulacije

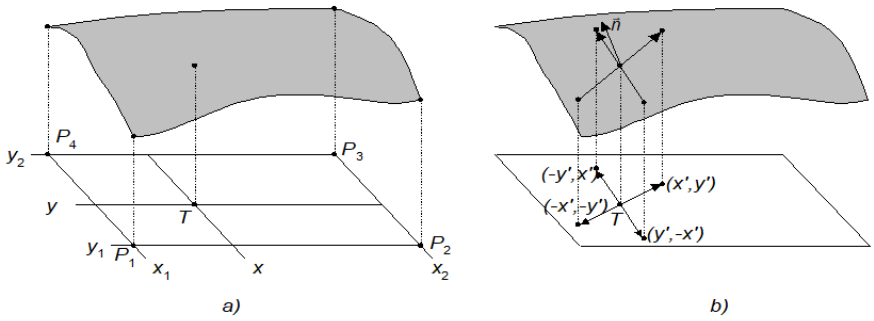
Arhitektura vizualnog simulatora ATGM sustava može kombinirati komercijalne alate (kao što su *MultiGen-Creator* ili *Vega*) s programerovim vlastitim kodom u *Visual C++* jeziku. [2]

U kontekstu sintetskih okolina, baze podataka vizualnih okoliša trebale bi sadržavati sljedeće elemente [3]:

- Geometriju ,,arene'' za igru, zadanu ili bazom podataka egzaktnih visina ili algoritmima koji definiraju oblik i elevaciju terena. Ta baza podataka sastoji se homogenih ili nehomogenih meshova poligona koji aproksimiraju površinu terena.
- Dvodimenzionalnih kulturnih značajki koje opisuju važne značajke površina poput vlage ili prometnosti, a uključuje i 2D infrastrukturu poput rijeka ili cesta.
- Trodimenzionalne značajke, tj. fiksirane objekte ili prepreke koji imaju vlastitu visinu, kao što su zgrade, vegetacija, stupovi itd. U nekim implementacijama može se dinamički utjecati na objekt, npr. on može biti uništen, no njegova pozicija i dalje ostaje nepromijenjena.
- Objekte koji se kreću, za koje je potrebno definirati njihovu geometriju i kinematičke i dinamičke karakteristike, a mogu se ostvariti kao predefinirani scenarij, algoritamski ili biti upravljani od strane kontrolora simulatora.
- Modifikatore scene (scenskih efekata) koji prilagođavaju konačni vizualni prikaz ovisno o željenom scenariju, pomoću značajki kao što su npr. svjetlost, teksture, vidljivost i tako dalje. Služe za unapređivanje fotorealističnog iskustva operatera.

Iako su za realistični doživljaj i visoku razinu utreniranosti najpogodniji simulacijski sustavi sa slobodnom točkom gledišta (tj. oni u kojima operater može slobodno gledati na okolinu iz različitih kutova, kao da okreće glavu u pravom svijetu), zbog visoke cijene i kompleksnosti izvedbe takvih sustava često se radi kompromis i odabire sustav u kojem je točka gledišta fiksna. Tako nestaje potreba za stalnim iscrtavanjem (*render*) terena za svaki *frame*, već se kao pozadina može koristiti digitalizirana slika preko koje se onda iscrtavaju umjetni objekti. Uzevši u obzir da je za glatku animaciju simulacije potrebno imati barem 15 slika po sekundi, jasno je da će ovo biti vrlo popularan pristup za simulacije s ograničenim budžetom.

Kod takvih sustava baza terena mora sadržavati samo matrice elevacije četverostranih okvira polja (Sl. 3) i algoritam za interpolaciju visine. Za uspješno poravnavanje digitalizirane slike s okolinom moramo znati neke detalje o kameri kao što su njena pozicija, orijentacija te žarišna udaljenost leće objektiva. Slika širokog formata izbjegavaju se zbog čestih distorzija koje otežavaju asimilaciju slike i okoline.



Sl. 3. Elevacija (a) i vektor normale (b) točke T

Kod fiksnih kulturnih značajki ključno je memorirati samo mali broj poligona njihove prednje strane, s tim da se treba pripaziti na njihov odnos s pozadinom i osigurati da prikrivaju dio terena kao što bi to radili u pravom svijetu. Dinamički objekti mogu biti pozicionirani bilo gdje, stoga se njihova implementacija ne razlikuje od one u kompliciranijim sustavima slobodne točke gledišta. Proces njihova iscrtavanja tako sadržava dva dijela; prvo se dio terena koji je objekt prikrivao prije 1-2 *framea* ponovno iscrtava, tj. obnavlja, a nakon toga se pokretni objekt premješta u novu poziciju. Za scenske modifikatore koriste se jednostavne tehnike saturacije boja da bi se dočaralo prilike kao što su sumrak, magla ili efekti infracrvenog svjetla. Obični RGB model boja transformira se u jednadžbu ciljnih boja dodajući u igru i konstante intenziteta(μ) i saturacije(λ). Ova tehnika se koristi i za boje pozadine, a može biti i unaprijeđena korištenjem boljih modela kao što su HSV i HLS ili korištenjem *palette* moda boja.

Što se tiče razvojne faze vizualne simulacije s konkretnim alatima [2], prvi korak je 3D modeliranje Za to je popularan izbor MultiGen-Creator, s kojim kreiramo modele rakete, cilja i terena, organizirajući *OpenFlight* bazu podataka s hijerarhijskom strukturom. Optimiziramo LOD (*Level-of-Detail*) za rad u stvarnom vremenu, pridajući više detalja (tj. građu od većeg broja poligona) objektu koji nam je bliži u virtualnom okruženju, a skidajući detalje u slučaju da je udaljen od nas. Osim što time štedimo memoriju i procesor, sasvim je opravdano jer i u pravom svijetu na udaljenim objektima ne vidimo detalje određene razine. Nakon optimizacije, konfiguriramo senzore i okruženje, koristeći alate poput TMM-a (*Texture Material Mapper*, pridavanje fizikalnih svojstava materijala geometrije) i MAT-a (*MOSART Atmospheric Tool*, preračunavanje atmosferskih uvjeta okoline). Na kraju, pokrećemo scenu alatom Vega, često u integraciji s Visual C++ kodom koji kontinuirano ažurira parametre rakete tijekom leta. Ovaj pristup u cjelini omogućava evaluaciju različitih algoritama praćenja, kao što su praćenje po boji i po razlici okvira, u različitim atmosferskim uvjetima i s različitim nesavršenostima IR senzora.

U sljedećoj podsekciji vidjet ćemo kako su određeni elementi simulacije implementirani u Godotu, modernom *game engineu* koji nudi moćne alate za vizualizaciju i fizikalnu simulaciju.

3.5 Odabrani elementi implementacije u Godotu

KRETANJE METE. Tenkovi se nasumično kreću po tlu lijevo-desno na način da se inkrementira iznos rotacije u stupnjevima za neki nasumični broj, ovisno o tome kolika je brzina rotacije. Prvo se odabire koliko će se dugo okretati prvo u lijevu, odnosno u desnu stranu te se opet iznova ponavlja taj proces.

```
39
40  func tilt_to_normal(terrain_normal: Vector3, delta: float):
41      previous_terrain_normal = previous_terrain_normal.lerp(terrain_normal, tilt_smoothness * delta).normalized()
42
43      var current_forward = transform.basis.z
44      var current_right = transform.basis.x
45      var new_up = previous_terrain_normal
46
47      var new_right = current_right - new_up * current_right.dot(new_up)
48      new_right = new_right.normalized()
49      var new_forward = new_up.cross(new_right).normalized()
50
51      transform.basis = Basis(new_right, new_up, new_forward)
52
53  func snap_to_terrain():
54      var terrain_info = get_terrain_info()
55      if terrain_info:
56          global_position.y = terrain_info.position.y + 100
57          previous_terrain_normal = terrain_info.normal
```

```
func get_terrain_info():
    var space = get_world_3d().direct_space_state
    var ray_start = global_position + Vector3(0,1,0) * 10
    var ray_end = global_position + Vector3(0,-1,0) * 10000

    var result = space.intersect_ray(PhysicsRayQueryParameters3D.create(ray_start, ray_end))

    return result
```

Budući da je tlo valovito treba voditi računa o nagibu tenkova i o tome jesu li na istoj visini kao i tlo. To osiguravaju metode *tilt_to_normal* i *snap_to_terrain* te pomoćna metoda *get_terrain_info*. Metoda *tilt_to_normal* koja prima vektor kao argument pomoću prethodnih vektora smjerova tenka izračunava nove smjerove kako bi tenk bio ispravno rotiran za sve tri osi rotacije. Kao argument šaljemo normalu tla za trenutnu poziciju tenka. Funkcija *snap_to_terrain* visinu na kojoj je tenk postavlja na visinu tla + neki *offset* da tenk ne završi u tlu. Pomoćna funkcija *get_terrain_info* nam daje informacije o tlu za trenutnu poziciju tenka na način da pošalje okomitu zraku pomoću koje izračunavamo informacije o sjecištu s tlom. Pomoću nje dobivamo točku sjecišta i normalu tla.

DETEKCIJA KOLIZIJE. Godot omogućava da se kolizije automatski detektiraju. Detekcija sudara rakete obavlja se na način da pozovemo funkciju *move_and_collide* koja nam pomakne raketu u smjeru vektora brzine i vraća nam (u slučaju da postoji kolizija) poticaj da se raketa resetira na početnu poziciju pomoću funkcije *restart_rocket*.

```
var collision = move_and_collide(velocity * sub_dt)
if collision:
    # Play explosion sound on collision
    match randi_range(0, 2):
        0:
            %SoundRocketExplode1.play()
        1:
            %SoundRocketExplode2.play()
        2:
            %SoundRocketExplode3.play()
    restart_rocket()
    emit_signal("collision")
    break
```

```
func _physics_process(delta):
    arrow.position=tank.position
    explosion.position=tank.position
    if(tank.visible==false):
        tank_camera.current=true
        tank.move_speed= 0
        tank.rotation_speed=0
        arrow.visible=false
        if(explosion.exploded==false):
            slowmo_enabled=false
            Engine.time_scale = 1.0
            explosion.explode()
            await get_tree().create_timer(4.0).timeout
            get_tree().reload_current_scene()
```

```
func _ready():
    body_entered.connect(_on_body_entered)

func _on_body_entered(body: Node):
    if body.is_in_group("rocket"):
        print("rocket entered tank area3d")
        call_deferred("destroy_tank", body)
```

Potrebno je razlikovati koliziju s tenkom i s tlom jer ovo prethodno vrijedi za svako tijelo koje ima omogućenu koliziju. Kada je detektirana kolizija s tenkom potrebno je napraviti da taj tenk nestane i da se pokrene animacija eksplozije.

3.6 Kretanje rakete (razgovor s LLM-om)

Kretanje rakete modelirano je pomoću pojednostavljenog dinamičkog modela temeljenog na Newtonovu drugom zakonu. Translacija rakete opisuje se sustavom diferencijalnih jednadžbi u kojem se položaj i brzina dobivaju numeričkom integracijom ukupnih sila koje djeluju na raketu. U model su uključene sila potiska, aerodinamički otpor proporcionalan kvadratu brzine te gravitacijska sila.

Orijentacija rakete (nagib i skretanje) ne računa se iz jednadžbi momenta, već se izravno upravlja pomoću korisničkog unosa. Na taj način smanjuje se složenost modela, dok se zadržava fizički smislen opis translacijskog gibanja. Smjer sile potiska uvijek je poravnat u odnosu na uzdužnu os rakete, definiranom trenutačnim kutovima orijentacije. Numerička integracija provodi se u diskretnim vremenskim koracima unutar fizikalne petlje simulacije, što omogućuje stabilno i predvidljivo ponašanje rakete u stvarnom vremenu. Model daje izlazne veličine položaja i orijentacije kompatibilne s jednostavnijim modelima vođenja, čime je omogućena laka zamjena različitih modela unutar simulacije.

U nastavku je razgovor s LLM-om (*ChatGPT*) o implementaciji gibanja rakete opisanoj diferencijalnim jednadžbama.

2. Minimal dynamics model (core equations)

State variables (continuous)

Your missile has:

- Position:

$$\mathbf{r} = (x, y, z)$$

- Velocity:

$$\mathbf{v} = (v_x, v_y, v_z)$$

- Orientation (Euler angles):

$$\theta = \text{pitch}, \quad \psi = \text{yaw}, \quad \phi = \text{roll}$$

Forces to include (keep it simple)

Use **only forces**, no moments:

1. **Thrust** (along missile forward axis)
2. **Drag** (opposes velocity)
3. *(Optional)* Gravity

From pitch and yaw:

$$\hat{f} = \begin{bmatrix} \cos\theta \cos\psi \\ \sin\theta \\ \cos\theta \sin\psi \end{bmatrix}$$

This vector defines where the missile is "pointing".

Forces

Thrust

$$\mathbf{F}_{thrust} = T \cdot \hat{f}$$

- `T` = constant thrust (or time-limited burn)

Drag (very simple)

$$\mathbf{F}_{drag} = -k_d \cdot \|\mathbf{v}\| \cdot \mathbf{v}$$

This is:

- Easy to implement
- Stable
- Good enough for a game

Gravity (optional)

$$\mathbf{F}_{gravity} = m \cdot \mathbf{g}$$

Equations of motion (this is the "differential equations" part)

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}$$

$$\frac{d\mathbf{v}}{dt} = \frac{1}{m}\left(\mathbf{F}_{thrust} + \mathbf{F}_{drag} + \mathbf{F}_{gravity}\right)$$

These are integrated numerically every frame.

Angle update (simple, controllable)

$$\theta_{new} = \theta + \dot{\theta}_{cmd} \cdot \Delta t$$

$$\psi_{new} = \psi + \dot{\psi}_{cmd} \cdot \Delta t$$

Where:

- `θ_cmd` and `ψ_cmd` come from input
- Clamp angles to avoid flipping if you want

This satisfies the requirement:

> "rotation angles controlled directly by keyboard or mouse"

4. Outputs (compatibility with option a)

Your model outputs:

- `x, y, z`
- `pitch, yaw, roll`

Exactly the same interface as option (a).

This means:

- You can **swap models** without changing the rest of the simulation
- Very good design choice (mention this explicitly in your report)

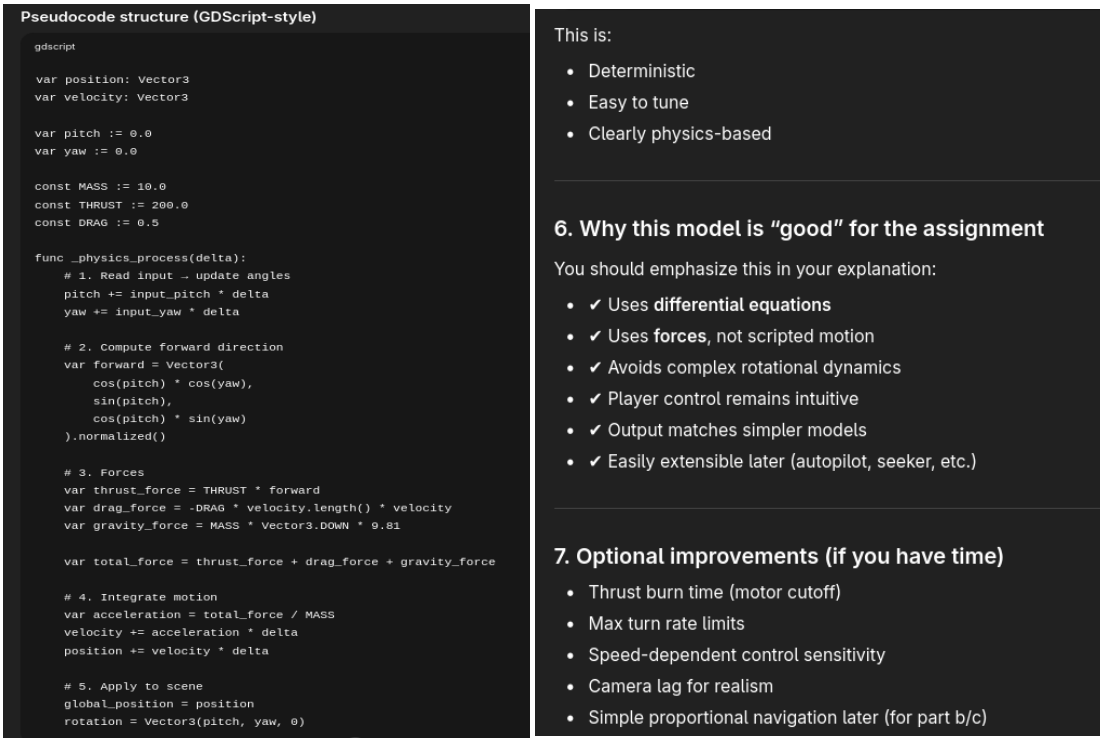
---

5. Godot implementation outline (very practical)

Node setup

Use:

- `Node3D` (missile root)
- Physics updated in `_physics_process(delta)`



Pseudocode structure (GDScript-style)

gdscript

```
var position: Vector3
var velocity: Vector3

var pitch := 0.0
var yaw := 0.0

const MASS := 10.0
const THRUST := 200.0
const DRAG := 0.5

func _physics_process(delta):
    # 1. Read input → update angles
    pitch += input_pitch * delta
    yaw += input_yaw * delta

    # 2. Compute forward direction
    var forward = Vector3(
        cos(pitch) * cos(yaw),
        sin(pitch),
        cos(pitch) * sin(yaw)
    ).normalized()

    # 3. Forces
    var thrust_force = THRUST * forward
    var drag_force = -DRAG * velocity.length() * velocity
    var gravity_force = MASS * Vector3.DOWN * 9.81

    var total_force = thrust_force + drag_force + gravity_force

    # 4. Integrate motion
    var acceleration = total_force / MASS
    velocity += acceleration * delta
    position += velocity * delta

    # 5. Apply to scene
    global_position = position
    rotation = Vector3(pitch, yaw, 0)
```

This is:

- Deterministic
- Easy to tune
- Clearly physics-based

6. Why this model is "good" for the assignment

You should emphasize this in your explanation:

- ✔ Uses **differential equations**
- ✔ Uses **forces**, not scripted motion
- ✔ Avoids complex rotational dynamics
- ✔ Player control remains intuitive
- ✔ Output matches simpler models
- ✔ Easily extensible later (autopilot, seeker, etc.)

7. Optional improvements (if you have time)

- Thrust burn time (motor cutoff)
- Max turn rate limits
- Speed-dependent control sensitivity
- Camera lag for realism
- Simple proportional navigation later (for part b/c)

4 Zaključak

Ovaj seminar pruža pregled arhitekture i pristupa razvoju simulatora protuoklopnih vođenih raketnih sustava. Kroz analizu literature moglo se doći do raznih zaključaka. Jedan od njih je činjenica da ne postoji optimalan, univerzalno najbolji pristup izradi ATGM simulatora, već odabrani pristup ovisi o našim potrebama i financijskim sposobnostima, iako se primjećuje težnja prema multimodalnim sustavima. To se vidjelo u kompromisima poput fiksne točke gledišta i korištenju komercijalnih alata poput Vege. Matematički modeli su od velike važnosti, iako je i tu primjetan mogući kompromis zbog kompleksnosti (Eulerovi kutovi naspram kvaterniona). IR goniometar je istaknut kao ključan i kritičan dio simulatora, čija potreba za širokim rasponom frekvencija predstavlja izazov pri izradi sustava. Primjećuje se i trend evolucije skupih, specijaliziranih sustava u pristupačnija rješenja bazirana na PC platformama. U moderno doba svakako su dostupne sve bolje razvijene tehnologije vizualizacije koje mogu realističnije prikazati simulacijsku okolinu, umjesto jednostavnih tehnika kao što je obična digitalizirana slika. Moguća su i razna poboljšanja upotrebom hibridnih arhitektura.

Sve u svemu, razvoj ovih simulatora predstavlja spoj matematičkog modeliranja, računalne grafike i računalnog inženjerstva visoke razine. U modernom dobu se očitim napretkom smatra integracija ovakvih sustava s *game engine* alatima kao što su Unity ili Godot, a potonjeg smo predstavili i praktično. Njihove moćne računalne karakteristike mogle bi dovesti do razvoja i testiranja novih algoritama navođenja, kontrole i obrade signala. Također, važno se podsjetiti da je na kraju krajeva cilj simulatora ljudima omogućiti siguran, učinkovit i ekonomičan trening koji će ih adekvatno pripremiti da u stvarnom životu sposobno postupe u situacijama izučenima na simulatoru.

5 Literatura

[1] Ćosić, K., Kopriva, I., Kostić, T., Slamić, M., & Volarević, M. (1999). Design and implementation of a hardware-in-the-loop simulator for a semi-automatic guided missile system. *Simulation Practice and Theory*, 7(2), 107-123.

[2] El-gabri, A. N. O., Ahmed, A. S., Elhalwagy, Y. Z., & Khamis, A. (2018). Command Guidance Missile Tracking Algorithms Evaluation Based On Visual Simulation. *International Journal of Engineering Research and Technology*, 7, 110-116.

[3] Slamić, M., Penzar, I., Penzar, D., & Stjepanović, J. (1997). Fixed-viewpoint representation of synthetic environment for the low-cost industrial and military simulators. *Mathematics and computers in simulation*, 44(3), 263-269.

[4] https://en.wikipedia.org/wiki/9M14_Malyutka

[5] https://en.wikipedia.org/wiki/9K111_Fagot