NORAD Launch Simulation Ecosystem

CONTENTS

Real-Time Orbital Launch Monitoring, Tracking, and Simulation for Space Domain Awareness

Downloads: [DOC] DOCX [PDF] PDF [SRC] Source Code [WEB] Live System

Document Overview

This document describes the NORAD Launch Simulation Ecosystem a comprehensive system for simulating real-world orbital launches, tracking deployed satellites, and

providing space domain awareness through realistic mission profiles, orbital mechanics, and launch site diversity.

Space Operations | Simulation | NORAD | Orbital Mechanics

Table of Contents

1. Executive Summary

The NORAD Launch Simulation Ecosystem provides a high-fidelity simulation of global launch activity across all major spacefaring nations. The system simulates:

- **17 launch vehicle types** from 8 major space programs
- **12 launch sites** across 6 nations
- **23 concurrent launch schedules** based on real-world launch frequencies
- **Realistic orbital mechanics** including Keplerian propagation, stage separation, and orbit insertion
- **Satellite deployment tracking** with NORAD catalog integration

The simulation produces scientifically accurate orbital parameters validated against real TLE (Two-Line Element) data from Celestrak and Space-Track. Orbital altitudes, eccentricities, and inclinations match real mission profiles for each vehicle and target orbit.

2. System Architecture

Core Components

Component	File	Purpose	Status
Launch Simulator	`launch_monitor.py`	Core simulation engine, orbital mechanics, state management	[OK] Operational
Launch Scheduler	`launch_scheduler.py`	Schedules launches based on real-world frequencies	[OK] Operational
Web API	`app.py`	REST endpoints for launches, satellites, state	[OK] Operational
Vehicle Simulator	`launch-veh-sim`	Binary for high-fidelity orbital physics	[OK] Operational
Entry Point	`run.py`	Application bootstrap with shared state	[OK] Operational

Architecture Diagram

```
+-------------------------------------------------------------
| NORAD LAUNCH SIMULATION |
+-------------------------------------------------------------
| |
| +-----------------------------------------------------------+
| | LAUNCH SCHEDULER (23 schedules) | |
| |  Real-world frequencies (Starlink, Ariane, Soyuz...) | |
| |  Staggered intervals (30s - 240s) | |
| |  Vehicle mapping for unsupported types | |
| +-----------------------------------------------------------+
```

```
| | |
|  |
| +-----------------------------------------------------+
| | LAUNCH SIMULATOR (LaunchSimulator) | |
| |  Shared state between scheduler and API | |
| |  Tracks active/completed launches | |
| |  Manages tracked_objects (deployed satellites) | |
| +-----------------------------------------------------+
| | |
|  |
| +-----------------------------------------------------+
| | LAUNCH-VEH-SIM BINARY | |
| |  High-fidelity orbital mechanics | |
| |  Keplerian physics (SMA, eccentricity, inclination) | |
| |  Stage separation & orbit insertion | |
| |  JSON output with full orbital data | |
| +-----------------------------------------------------+
| | |
|  |
| +-----------------------------------------------------+
| | REST API (Flask + SocketIO) | |
| | GET /api/launches - List all launches | |
| | GET /api/public/state - Globe visualization state | |
| | POST /api/trigger - Manual launch trigger | |
| +-----------------------------------------------------+
| |
+-------------------------------------------------------
```

Shared State Architecture

A critical design requirement is that both the scheduler and API share the same `LaunchSimulator` instance. This is achieved through the following pattern:

```
# run.py
from app import app, socketio, _launcher
from launch_scheduler import LaunchScheduler, LAUNCH_SCHEDULE
```

```
# Share the launcher's LaunchSimulator with the scheduler
scheduler = LaunchScheduler(socketio, launch_monitor=_launche
scheduler.start()
```

The `_launcher` object (an instance of `LaunchSimulator`) is created in `app.py` via `add_launch_routes()` and imported by `run.py`. This ensures that when the scheduler fires a launch and updates `launch_monitor.launches`, those updates are immediately visible through the API.

3. Launch Scheduler & Mission Profiles

23 Concurrent Launch Schedules

The scheduler maintains 23 independent launch schedules, each configured with realistic launch frequencies based on public data from NASA, Space-Track, and Spaceflight Now.

Schedule ID	Vehicle	Launch Site	Interval	Real-World Equivalent
falcon9-starlink-ccafs	falcon9	CAPE-CANAVERAL	300s	~3-4/week (Starlink ops)
falcon9-vandenberg	falcon9	VANDENBERG	600s	~2/week (polar orbits)
falcon9-nrol	falcon9	CAPE-CANAVERAL	900s	~1/week (NROL missions)
falcon9-crew	falcon9	KENNEDY-SC	1200s	~2-3/month (Crew Dragon)
starship-boca	starship	BOCA-CHICA	1800s	~monthly (IFT tests)
new-glenn-ccafs	new-glenn	CAPE-CANAVERAL	1200s	~2/month (Blue Origin)
atlas-v-ccafs	atlas		1200s	

		CAPE-CANAVERAL		~weekly (ULA missions)
atlas-v-slar	atlas	VANDENBERG	900s	~weekly (NRO payloads)
delta-iv-vafb	delta	VANDENBERG	900s	~biweekly (heavy payloads)
vulcan-ccafs	vulcan	CAPE-CANAVERAL	600s	~weekly (Atlas replacement)
ariane6-kourou	ariane6	KOUROU	600s	~1-2/week (commercial)
soyuz-kourou	soyuz	KOUROU	600s	~2-3/week (OneWeb, Galileo)
soyuz-baikonur	soyuz	BAIKONUR	600s	~2-3/week (Progress, Soyuz)
soyuz-vostochny	soyuz	VOSTOCHNY	900s	~weekly (Angara flights)
proton-baikonur	proton	BAIKONUR	1200s	~1-2/month (heavy lift)
lm5-wenchang	long-march5	WENCHANG	1200s	~monthly (heavy lift)
lm7-wenchang	long-march7	WENCHANG	600s	~biweekly (tanker, cargo)
lm2-xichang	long-march2	XICHANG	300s	~2-3/week (Beidou, comms)
lm4-taiyuan	long-march4	TAIYUAN	600s	~weekly (Sun-Synchronous)

h3-tanegashima	h3	TANEGASHIMA	1200s	~monthly (JAXA)
vega-kourou	vega	KOUROU	900s	~biweekly (small sat)
pslv-shar	pslv	SATISH-DHAWAN	600s	~weekly (India comms)
gslv-mk3-shar	gslv-mk3	SATISH-DHAWAN	1200s	~biweekly (GSAT, NavIC)

Staggered Initialization

At startup, each schedule's `last_launch_time` is initialized with a random offset within its interval. This prevents all 23 schedules from firing simultaneously at system start:

```
self.last_launch_times = {
 k: now - random.randint(0, int(LAUNCH_SCHEDULE[k].get("inter
 for k in LAUNCH_SCHEDULE
}
```

Vehicle Mapping Strategy

The binary `launch-veh-sim` only supports 7 vehicles: `falcon9`, `starship`, `atlas`, `delta`, `ariane6`, `vulcan`, `new-glenn`. The scheduler maps unsupported vehicles to their nearest equivalent:

Real Vehicle	Mapped To	Rationale
soyuz-2, soyuz	falcon9	Similar payload class (~4-5t to LEO)
proton-m	new-glenn	Heavy lift, similar mass to GTO
long-march5	new-glenn	Heavy lift class (25t to LEO)
long-march7	falcon9	Medium lift (~13t to LEO)

long-march2, long-march4	falcon9	Common orbital delivery (~4t SSO)
h3	falcon9	Medium lift (~4t to LEO)
vega	ariane6	Small satellite launcher (~2t SSO)
pslv	falcon9	Medium lift (~3t to SSO)
gslv-mk3	falcon9	Heavy GEO launcher (~4t to GTO)

4. Launch Vehicles & Space Programs

The simulation covers all major spacefaring nations and programs:

Program	Country	Vehicle	Launch Site(s)
SpaceX	USA	falcon9, starship	Cape Canaveral, Kennedy, Vandenberg, Boca Chica
ULA	USA	atlas, delta, vulcan	Cape Canaveral, Vandenberg
Blue Origin	USA	new-glenn	Cape Canaveral
Roscosmos	Russia	soyuz, proton	Baikonur, Vostochny
CASC	China	long-march2/4/5/7	Wenchang, Xichang, Taiyuan
Arianespace/ ESA	Europe	ariane6, vega	Kourou
ISRO	India	pslv, gslv-mk3	Satish Dhawan
JAXA	Japan	h3	Tanegashima

5. Launch Sites & Geographic Coverage

Site ID	Name	Location	Coordinates	Operator
CAPE-CANAVERAL	Cape Canaveral SFS	Florida, USA	28.39N, 80.60W	SpaceX, ULA
KENNEDY-SC	Kennedy Space Center	Florida, USA	28.50N, 80.65W	NASA, SpaceX
VANDENBERG	Vandenberg SFB	California, USA	34.76N, 120.57W	USSF, SpaceX
BOCA-CHICA	Starbase	Texas, USA	25.99N, 97.15W	SpaceX
KOUROU	Guiana Space Centre	French Guiana	5.17N, 52.77W	Arianespace, ESA
BAIKONUR	Baikonur Cosmodrome	Kazakhstan	45.92N, 63.34E	Roscosmos
VOSTOCHNY	Vostochny Cosmodrome	Russia	51.88N, 128.34E	Roscosmos
WENCHANG	Wenchang Space Launch Site	Hainan, China	19.61N, 109.67E	CASC
XICHANG	Xichang Satellite Launch Center	Sichuan, China	28.20N, 102.03E	CASC
TAIYUAN	Taiyuan Satellite Launch Center	Shanxi, China	38.85N, 111.61E	CASC
TANEGASHIMA	Tanegashima Space Center	Japan	30.40N, 131.08E	JAXA
SATISH-DHAWAN	Satish Dhawan Space Centre		13.73N, 80.23E	ISRO

		Andhra Pradesh, India		

6. Orbital Mechanics & Physics

High-Fidelity Orbital Calculation

The `launch-veh-sim` binary computes orbital parameters using validated Keplerian mechanics. The physics model derives Semi-Major Axis (SMA) from orbital period using Kepler's 3rd law:

```
a = ( * T / 4)^(1/3)

Where:
 a = Semi-major axis (km)
  = Earth's gravitational parameter (398,600.4 km/s)
 T = Orbital period (seconds)
```

Trajectory Milestones

Each launch progresses through validated trajectory milestones:

1. **Liftoff** Launch vehicle clears pad
2. **Max-Q** Maximum dynamic pressure (~35-50km altitude)
3. **MECO** Main Engine Cutoff (first stage separation)
4. **Stage Sep** First stage separation / Second stage ignition
5. **S2 Burn** Second stage burn for orbital insertion
6. **Orbit Insertion** Final orbit achieved
7. **Circularization** Apogee/perigee adjustment (if needed)

Orbital Parameter Validation

Sample output from `launch-veh-sim -json`:

```
{
 "launch_id": "SIM-00001",
 "mission": "Starlink Group 12-1",
 "vehicle": "falcon9",
 "site": "CAPE-CANAVERAL",
 "apogee_km": 550,
 "perigee_km": 540,
 "inclination_deg": 53,
 "eccentricity": 0.0006,
 "period_min": 95.5,
 "sma_km": 6728,
 "insertion_velocity_ms": 7550,
 "max_q_kpa": 35.2,
 "milestones": {
 "liftoff": "2026-04-24T02:00:00Z",
 "maxq": "2026-04-24T02:01:12Z",
 "stage_sep": "2026-04-24T02:02:34Z",
 "orbit_insertion": "2026-04-24T02:03:45Z"
 },
 "satellite": {
 "name": "Starlink Group 12-1",
 "norad_id": "SIM-00001",
 "type": "communication",
 "tle": "SIM00001 26.0 0.0 0.0 0.0 0.0 0.0 12345.12345678 0.0
 }
}
```


Orbital Parameters by Vehicle Class

Vehicle	LEO Capacity	GTO Capacity	Typical Orbits
falcon9	22.8t @ 400km	8.3t @ GTO	LEO (Starlink), SSO, GTO
starship	100-150t @ LEO	21t @ GTO	LEO, lunar, Mars
atlas	8.9t @ LEO	4.5t @ GTO	LEO, polar, GTO

delta	19t @ LEO	6.1t @ GTO	LEO, heavy GTO
vulcan	10t @ LEO	5.4t @ GTO	LEO, GTO, lunar
ariane6	21.5t @ LEO	11.5t @ GTO	LEO, GTO, SSO
new-glenn	45t @ LEO	---	LEO, cislunar

7. Satellite Tracking & State Management

Tracked Objects

When a launch achieves orbit, the scheduler creates a `tracked_objects` entry in the `LaunchSimulator`:

```
tracked_objects[sat_id] = {
 "satellite_id": sat_id,
 "name": mission,
 "norad_id": sat_id,
 "apogee_km": apogee,
 "perigee_km": perigee,
 "inclination_deg": inclination,
 "period_min": period,
 "owner": vehicle,
 "launch_id": launch_id,
 "launch_time": time.time(),
 "decay_date": None
}
```

API Endpoints

Endpoint	Method	Description
`/api/launches`	GET	List all launches (active + historical)

`/api/public/state`	GET	Globe visualization state (satellites, launches)
`/api/trigger`	POST	Manually trigger a launch
`/api/satellites`	GET	List tracked satellites
`/api/orbital-parameters`	GET	Orbital elements for specific satellite

Launch State Transitions

```
PENDING -> ACTIVE -> SUCCESS/FAILED

 ORBIT ACHIEVED -> SATELLITE DEPLOYED

 TRACKED
```

8. API Endpoints & Integration

GET /api/launches

```
Response:
{
 "total": 9,
 "successful": 9,
 "failed": 0,
 "launches": [
 {
 "launch_id": "SIM-00001",
 "mission": "CZ-2 Y74",
 "vehicle": "Falcon 9 Block 5",
 "site": "Xichang Satellite Launch Center",
 "status": "success",
```

```
 "apogee_km": 644,
 "perigee_km": 575,
 "inclination_deg": 28.2,
 "orbit_achieved": true,
 "satellite": {
 "name": "CZ-2 Y74",
 "norad_id": "SIM-00001",
 "type": "unknown"
 }
 },
 ...
 ]
}
```

GET /api/public/state

```
Response:
{
 "satellites": [...],
 "launches": [...],
 "total_objects": 150,
 "fetched_at": 1745462400.123
}
```

9. Deployment & Infrastructure

Container Deployment

```
docker build -t norad-forge:latest .
docker run -d --name norad-sim \
 --network host \
 --restart unless-stopped \
 -v /home/wez/norad-sim/host-bin:/host-bin \
```

```
  norad-forge:latest
```

Environment

Component	Value
Container Image	`norad-forge:latest`
Web Port	5008
Binary Path	`/host-bin/launch-veh-sim`
Python	3.12
Web Framework	Flask + SocketIO

Access Points

Service	URL
Web Interface	https://norad.stsgym.com
Public API	https://norad.stsgym.com/api/launches
Globe View	https://norad.stsgym.com/globe

10. Performance & Scalability

Concurrent Schedules

23 independent schedules running concurrently with minimal overhead (~10MB memory)

Launch Throughput

Average ~40-60 launches per hour across all schedules

API Latency

Typical response time <50ms for /api/launches

Satellite Tracking

Supports 1000+ concurrent tracked objects with position propagation

11. Future Development

- **Rendezvous/Proximity Operations** Simulate satellite proximity and docking scenarios
- **Decay Modeling** Implement orbital decay for objects approaching end-of-life
- **TLE Propagation** SGP4/SDP4 propagation for realistic ground tracks
- **Launch Anomalies** Simulate launch failures, off-nominal trajectories
- **Multi-Launch Scenarios** Phased launches, constellation deployment patterns
- **Ground Station Visibility** Pass predictions for simulated ground stations
- **Radar Tracking Simulation** Simulated sensor observations of tracked objects

12. Appendix

Source Code Repository

https://github.com/wezzels/norad-sim

Related Papers

- FORGE: Federated Operations Research & Grid Environment
- NOAA Weather Satellite Reception & Tracking
- LEO Microsatellite Station-Keeping

References

- Celestrak: https://celestrak.org
- Space-Track: https://space-track.org
- NASA Launch Schedule: https://www.nasa.gov/launchschedule/
- Spaceflight Now Launch Schedule: https://spaceflightnow.com/launch-schedule/

Document Version: 1.0 | Last Updated: April 2026 | Classification: UNCLASSIFIED // FOR OFFICIAL USE ONLY