ORBITAL PLANE ALIGNMENT FOR MULTI-TARGET ACTIVE SPACE DEBRIS REMOVAL VIA J_2 PERTURBATION WITH AGENTIC-ASSISTED DEVELOPMENT

By Sidak Plaha

Individual Third Year Project Report April 2026
Supervisor: Dr. Angadh Nanjangud

SCHOOL OF ENGINEERING AND MATERIALS SCIENCE
ENGINEERING
THIRD YEAR PROJECT
EMS690U

APRIL 2026

DECLARATION

This report entitled

Orbital Plane Alignment for Multi-Target Active Space Debris Removal Via J_2 Perturbation with Agentic-Assisted Development

Was composed by me and is based on my own work. Where the work of others has been used, it is fully acknowledged in the text and in captions to table illustrations. This report has not been submitted for any other qualification.

Name: *Sidak Plaha*

Signed: SIDAK PLAHA

Date: 29/04/2026

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Angadh Nanjangud for his guidance and support throughout this project. Without his expertise, this dissertation would not have been possible.

I am also extremely grateful to my family for their unwavering support to get me to where I am today. Without their love and encouragement, I would not be in the position I am today to complete this dissertation.

Sidak Plaha
Date: 22/04/2026
Aerospace Engineering

Abstract

Active debris removal (ADR) in Sun-synchronous orbit is fuel-intensive because debris is distributed across orbital planes. This dissertation designs a Python-based mission planner that exploits secular J_2 precession to reduce the velocity change required for ADR. Each transfer leg is modelled by selecting a drift orbit whose altitude and inclination passively close the right ascension of the ascending node (RAAN) gap between the mothership and a target, avoiding direct impulsive manoeuvres. A greedy sequencer applies this transfer model to a 637-object Space-Track.org debris catalogue under a 500 m/s mission budget, with only RAAN propagated between legs in this report.

For an illustrative single-leg case, the J_2-assisted transfer costs 4.20 m/s compared with 781.0 m/s for a direct impulsive round trip, a 99.5% reduction. The baseline sequencer removes 13 debris objects, using 363.0 m/s over 8.34 years. A composite cost sweep then imposes a time penalty on drift duration. Small positive penalties restructure the target sequence before substantial duration compression occurs, raising the removal count from 13 to 19 with little change in duration. The best compliant time-constrained case removed 19 objects in 1.65 years while remaining within the 500 m/s budget after a 5% propellant margin.

Contents

Nomenclature Table

Acronym	Meaning
ADR	Active debris removal
AI	Artificial intelligence
ARMA	Autonomous Removal and Maintenance Architecture
DOP853	Dormand-Prince 8(5,3) Runge-Kutta integrator
ECI	Earth-Centered Inertial (reference frame)
GVE	Gauss's Variational Equations
LEO	Low Earth orbit
LVLH	Local-Vertical-Local-Horizontal (reference frame)
NOAA	National Oceanic and Atmospheric Administration
NORAD	North American Aerospace Defense Command
NSGA-II	Non-dominated Sorting Genetic Algorithm II
RAAN	Right Ascension of the Ascending Node
SGP4	Simplified General Perturbations 4
SSO	Sun-synchronous orbit
TLE	Two-Line Element
UAV	Unmanned aerial vehicle

Symbol	Description	Units
Physical Constants		
μ	Standard gravitational parameter of Earth	$\mathrm{m}^3\,\mathrm{s}^{-2}$

Symbol	Description	Units
R_E	Mean equatorial radius of Earth	m
J_2	Second zonal harmonic coefficient of Earth's gravitational potential	—
Classical Orbital Elements		
a	Semi-major axis	m
e	Eccentricity	—
i	Orbital inclination	rad
i_1, i_2	Initial and final inclinations for an inclination change	rad
Ω	Right ascension of the ascending node	rad
Ω_0	Initial RAAN of the mothership parking orbit	rad
Ω_{park}	RAAN of the parking orbit	rad
Ω_{target}	RAAN of the target orbit	rad
$\Omega_{\text{mothership}}$	RAAN of the mothership orbit	rad
ω	Argument of perigee	rad
u	Argument of latitude	rad
θ	True anomaly	rad
Derived Orbital Quantities		
n	Mean motion	$\text{rad}\,\text{s}^{-1}$
n_{drift}	Mean motion of the drift orbit	$\text{rad}\,\text{s}^{-1}$
T	Orbital period	s
p	Semi-latus rectum	m
p_{drift}	Semi-latus rectum of the drift orbit	m
h	Specific angular momentum	$\text{m}^2\,\text{s}^{-1}$
r	Instantaneous orbital radius	m
v_c	Circular orbital velocity	$\text{m}\,\text{s}^{-1}$
v_{park}	Circular velocity at the mothership parking orbit	$\text{m}\,\text{s}^{-1}$
$\dot{\Omega}$	Secular J_2 RAAN precession rate	$\text{rad}\,\text{s}^{-1}$

Symbol	Description	Units
$\dot{\Omega}_0$	RAAN precession rate of the mothership parking orbit	$\mathrm{rad\,s^{-1}}$
$\dot{\Omega}_{\mathrm{required}}$	Drift-orbit RAAN rate required to close the RAAN gap	$\mathrm{rad\,s^{-1}}$
$\dot{\Omega}_{\mathrm{target}}$	Target debris RAAN precession rate	$\mathrm{rad\,s^{-1}}$
Gauss's Variational Equations		
f_r	Perturbing acceleration in the radial direction	$\mathrm{m\,s^{-2}}$
f_c	Perturbing acceleration in the along-track direction	$\mathrm{m\,s^{-2}}$
f_n	Perturbing acceleration in the cross-track (orbit-normal) direction	$\mathrm{m\,s^{-2}}$
$\dot{a}$	Time derivative of semi-major axis	$\mathrm{m\,s^{-1}}$
$\dot{e}$	Time derivative of eccentricity	$\mathrm{s^{-1}}$
$\dot{i}$	Time derivative of inclination	$\mathrm{rad\,s^{-1}}$
$\dot{\omega}$, $\dot{u}$	Time derivatives of argument of perigee and argument of latitude	$\mathrm{rad\,s^{-1}}$
Manoeuvre Parameters		
ΔV	Impulsive velocity change (delta-V)	$\mathrm{m\,s^{-1}}$
ΔV_1, ΔV_2	First and second Hohmann-transfer burn magnitudes	$\mathrm{m\,s^{-1}}$
ΔV_{leg}	Total ΔV of a single transfer leg	$\mathrm{m\,s^{-1}}$
$\Delta V_{\Delta i}$	Pure inclination-change cost	$\mathrm{m\,s^{-1}}$
$\Delta V_{\Delta\Omega}$	Pure RAAN-change cost	$\mathrm{m\,s^{-1}}$
$\Delta V_{\Delta i,\Delta\Omega}$	Combined inclination and RAAN plane-change cost	$\mathrm{m\,s^{-1}}$
ΔV_{Hoh}	Hohmann-transfer contribution to a transfer leg	$\mathrm{m\,s^{-1}}$
$\Delta V_{\mathrm{plane}}$	Single combined plane-change cost in the direct baseline	$\mathrm{m\,s^{-1}}$

Symbol	Description	Units
$\Delta V_{\mathrm{direct}}$	Direct impulsive round-trip cost	$\mathrm{m\,s^{-1}}$
ΔV_{J_2}	J_2-assisted transfer-leg cost in the demonstration case	$\mathrm{m\,s^{-1}}$
$\sum \Delta V$	Cumulative mission ΔV expenditure	$\mathrm{m\,s^{-1}}$
Δi	Inclination change across a manoeuvre or between two orbits	rad
$\Delta\Omega$	RAAN change across a manoeuvre or between two orbits	rad
ϑ	Combined plane-change angular separation	rad
a_t	Semi-major axis of the Hohmann transfer ellipse	m
a_{park}, a_{drift}, a_{target}	Parking, drift, and target-orbit semi-major axes	m
i_{park}, i_{drift}, i_{target}	Parking, drift, and target-orbit inclinations	rad
r_1, r_2	Initial and final orbital radii of a Hohmann transfer	m
t_{Hohmann}	Hohmann transfer time of flight	s
t_{drift}	Duration of the J_2 drift phase	s (or days)
Mission Sequencing		
B	Mission ΔV budget	$\mathrm{m\,s^{-1}}$
λ	Time-penalty weighting coefficient	$\mathrm{m\,s^{-1}\,day^{-1}}$
C	Composite transfer cost ($C = \Delta V_{\mathrm{leg}} + \lambda\, t_{\mathrm{drift}}$)	$\mathrm{m\,s^{-1}}$
Δh	Altitude offset relative to the mothership parking orbit	km
$\Delta\Omega_{\mathrm{init}}$	Initial RAAN gap at the mission epoch	deg
$\Delta\Omega_{\mathrm{prop}}$	Propagated RAAN gap when the leg is priced	deg

Chapter 1

Introduction and Literature Review

1.1 Problem Statement

Orbital debris in low Earth orbit (LEO) threatens the long-term sustainability of space operations. Sun-synchronous orbits (SSO) are particularly vulnerable, concentrating a disproportionate density of both active spacecraft and defunct debris within a narrow range of inclinations and altitudes[1]. Left alone, these will collide and fragment, generating more debris in a chain reaction where the number of fragments increases exponentially. This is known as the Kessler syndrome[2]. A historical example is the 2009 Iridium-Cosmos collision, which created over 1800 debris pieces larger than 10 cm and more smaller fragments[3,4].

This necessitates active debris removal (ADR), but multi-target ADR is fuel-intensive, since direct impulsive manoeuvres between debris in different orbital planes are prohibitively expensive. This dissertation exploits the secular J_2 precession of the right ascension of the ascending node (RAAN), caused by Earth's oblateness, to reduce the cumulative velocity change (ΔV) required to service multiple targets from a single spacecraft.

Figures 1.1 and 1.2 show the 637 Space-Track.org debris catalogue used in this work to produce the results in Chapter 4. The complete dataset was filtered to perigee and apogee values within the ranges 550 and 750 km, and inclinations between 96° and 100°.

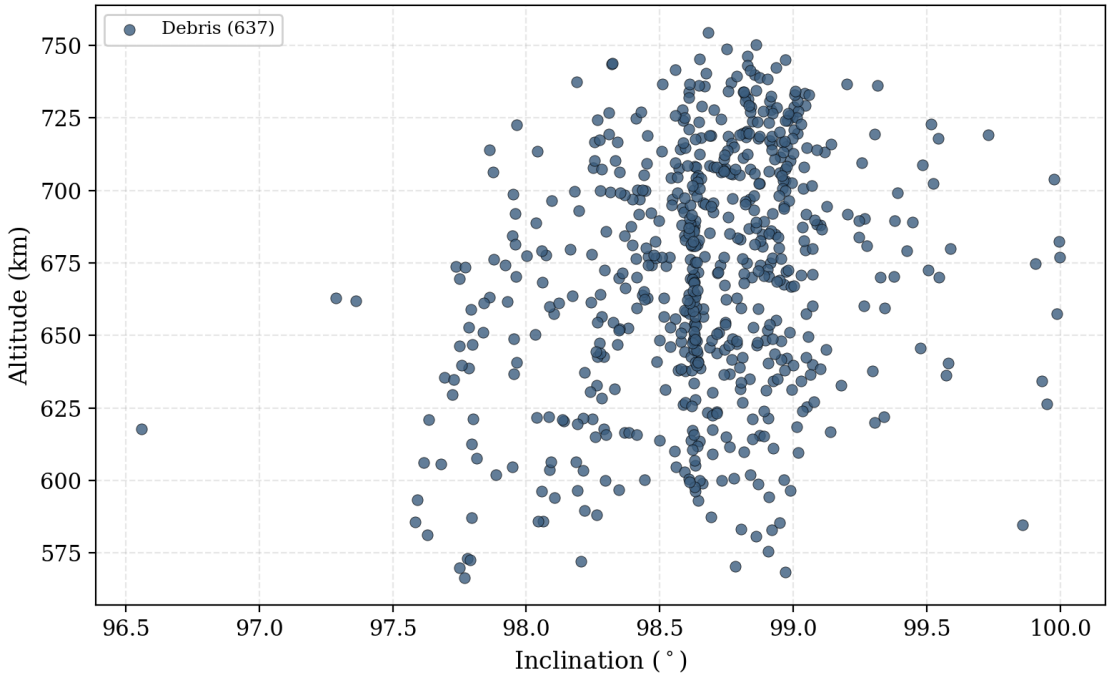

FIGURE 1.1. SSO Debris Catalogue: Inclination vs. Altitude

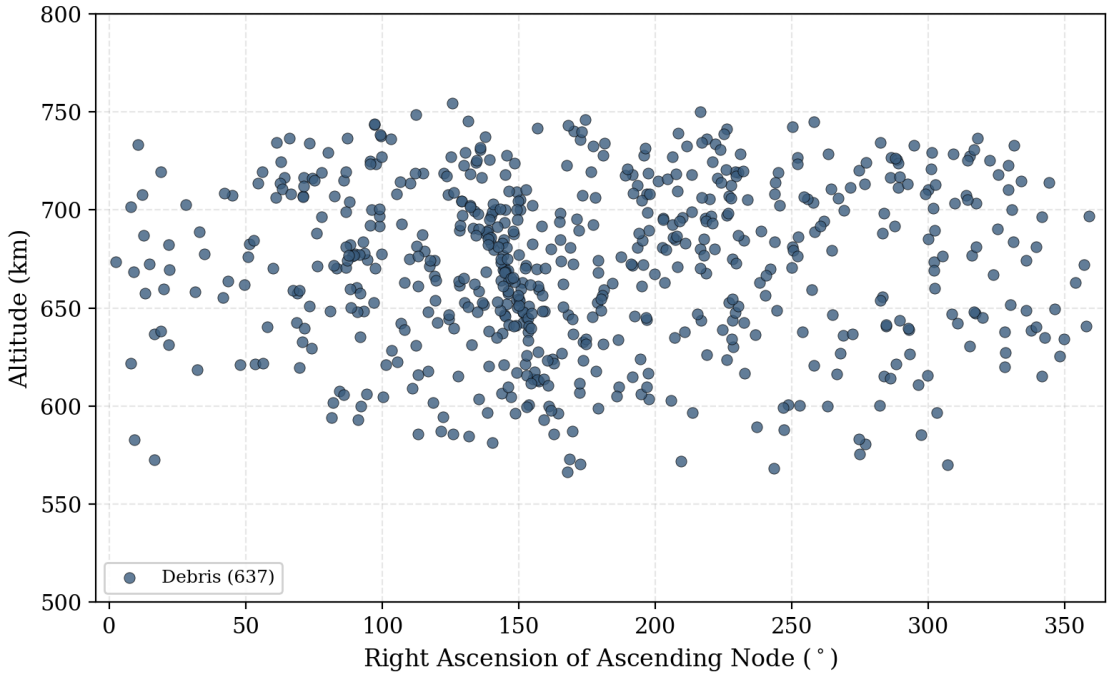

FIGURE 1.2. SSO Debris Catalogue: RAAN vs. Altitude

1.2 Multi-Target ADR Mission Concepts

Multi-target ADR must choose which debris to visit, in what order, and along what trajectory, all at the same time. The propellant cost of each leg depends on what is visited next, so the decisions cannot be made independently, and this coupling is what makes the problem hard.

Recent work has addressed it with global optimisation. Choi et al.[5] applied NSGA-II to five or more SSO debris, optimising target selection, order, and transfer parameters together against propellant cost and a hazard-weighted benefit score. Their framework also exploits J_2-induced RAAN drift for the same reason this work does. For near-polar orbits, RAAN precession rates differ between altitudes, so a small altitude change at the start of a leg buys days of passive RAAN alignment and avoids a far more expensive impulsive plane change.

The sequencer developed here takes the opposite trade-off in that it scales to the full 637-object catalogue and returns the same mission for fixed inputs, but it does not guarantee the cheapest target sequence because the greedy rule has no lookahead.

1.3 Catalogue Data and Propagation Model

Public debris catalogues are commonly distributed as Two-Line Element (TLE) sets, which are intended for use with the Simplified General Perturbations 4 (SGP4) propagator rather than as direct osculating Keplerian states. SGP4 is useful for catalogue propagation because it embeds an operational perturbation model in a compact mean-element format, but those mean elements are not the same variables propagated by the reduced model in this dissertation.[6] For mission design studies, TLEs are therefore best treated as a convenient source of initial orbital geometry unless the full SGP4 state reconstruction and error model are retained.

The present work follows this limited use. Space-Track.org TLE data define the initial orbital geometry of each object in the filtered Sun-synchronous catalogue. Subsequent analysis is then performed with a J_2 model derived from Gauss's Variational Equations (GVE), since the GVEs propagate orbital elements directly while allowing perturbing accelerations to be resolved in the LVLH frame.[7] This choice makes the RAAN drift mechanism explicit, which is the physical effect exploited by the transfer-leg optimiser. It also avoids presenting SGP4 as the mission propagator when the later chapters use a reduced analytical J_2 model.

1.4 Greedy Sequencing

A greedy algorithm picks the option that appears best at each stage, without looking ahead[8]. Applied here, the spacecraft starts at the parking orbit and evaluates the ΔV cost to every remaining target in the catalogue, heads to the cheapest, and repeats the procedure after each completed leg until the fuel budget runs out.

Greedy sequencing has prior form in space applications. Liu et al.[9] applied it to target selection for the Wide Field Optical Telescope Array, outperforming a fixed survey pattern at negligible computational cost. Nguyen et al.[10] applied a greedy algorithm to search-and-rescue UAV trajectory planning over probability score maps. At each step, the UAV is directed to the neighbouring grid node with the highest detection score, with visited nodes set to zero to avoid revisits. When all eight neighbours have been exhausted, and the greedy rule would otherwise stall, a small escape routine redirects the UAV to the nearest unvisited non-zero node. The implementation generates feasible trajectories in under 0.5 seconds, fast enough for real-time use. Their domains and this one share the key structural feature that each choice's cost depends on what preceded it.

The sequencer developed in this work applies the same principle to orbital manoeuvre planning. For every candidate target, the transfer-leg optimiser first selects the drift geometry that minimises a composite cost combining ΔV expenditure with a penalty on the drift time required under J_2. The greedy sequencer then ranks the feasible targets by that same

metric. The relative weighting of the two terms controls the trade between fuel cost and mission duration, and its effect is quantified through a parameter sweep in Chapter 4.

1.5 This Report

1.5.1 Aim and Objectives

The aim of this dissertation is to develop and evaluate a mission sequencer model for planning multi-target active debris removal missions in SSO, using J_2-induced RAAN precession to reduce fuel expenditure.

The objectives are:

(1) Implement an orbital propagator based on Gauss's Variational Equations (GVE) (Chapter 2) that captures the secular effects of J_2.

(2) Develop a transfer leg model that selects a drift orbit altitude and inclination to passively close the RAAN gap between the spacecraft and a target, and computes the total ΔV cost including Hohmann transfers and inclination corrections.

(3) Use a greedy sequencing algorithm that repeatedly selects the cheapest reachable target from a real debris catalogue sourced from Space-Track.org.

1.5.2 Scope and Constraints

Debris altitudes are restricted to 550–750 km in SSO, all targets are assumed near-circular, and the mothership is modelled as a point mass. All manoeuvres are impulsive, so low-thrust propulsion, finite burn-arcs, and in-plane phasing are not modelled, and transfer legs consist of Hohmann transfers and instantaneous inclination changes only.

The return to parking orbit after each leg is a deliberate design choice. In the context of the entire mission, chaser satellites must rendezvous back with the mothership after each removal. The time spent waiting during this chaser return and reuse phase is not modelled here.

J_2 is the sole gravitational perturbation. Higher-order zonal harmonics, atmospheric drag, solar radiation pressure, and third-body effects are excluded; over the altitude range and mission durations considered, these are second-order to J_2-driven RAAN drift.

Between mission legs, only the secular J_2 term of RAAN is propagated for the mothership and the remaining debris targets. The semi-major axis, eccentricity, and inclination are held fixed during passive drift, consistent with the reduced secular model adopted here; argument of perigee and argument of latitude are not propagated at mission level because in-plane phasing is out of scope.

The greedy sequencer selects the cheapest next target at each step[8], which is not the sequence that maximises total removals. Initial orbital elements are taken from Space-Track.org TLE data at a single epoch; subsequent propagation in the mission model uses the reduced secular J_2 formulation described in Chapter 2. The framework therefore addresses mission planning at the trajectory level only.

This dissertation contributes to the Autonomous Removal and Maintenance Architecture (ARMA), a multidisciplinary project. The workload comprises laser range finding, mothership mission planning (this dissertation), chaser satellite rendezvous, capture methods, and deorbit strategies, distributed across five group members.

1.5.3 Agentic-Assisted Development

The codebase was developed with Claude Code, an agentic AI assistant, in a strictly supervisory capacity. The agent's contributions were confined to documentation edits, formatting fixes, and file renaming. All such commits have a co-authorship tag in the GitHub history (see Appendix A). No orbital mechanics equations, propagation algorithms, manoeuvre cost models, or sequencing logic were authored or suggested by the agent.

Chapter 2

Orbital Propagation and Manoeuvre Modelling

2.1 LVLH Reference Frame

This study uses the Local-Vertical-Local-Horizontal (LVLH) frame, a rotating reference frame centred on the spacecraft and aligned with the local orbital geometry. The three orthogonal basis directions are radial (f_r), along-track (f_c), and cross-track (f_n). These are illustrated in Figure 2.1, and propulsion and perturbation accelerations are expressed in this basis throughout the remainder of the dissertation.

A rejected alternative was the Earth-Centered Inertial (ECI) frame[11]. This was because the perturbing and thrust accelerations are most naturally expressed relative to the local orbital geometry. Using the ECI frame would require these components to be rotated into the ECI reference frame at each time step.

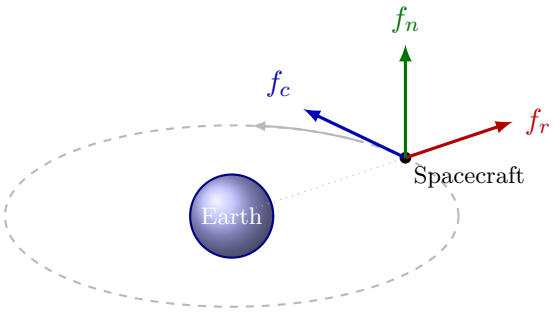

FIGURE 2.1. Local-vertical local-horizontal (LVLH) reference frame, centred on the spacecraft. The grey arrow marks the direction of orbital motion.

2.2 Gauss's Variational Equations

This dissertation adopts Gauss's Variational Equations because they propagate the orbital elements directly while allowing perturbing accelerations to be expressed in the LVLH frame of Section 2.1. This makes the secular effect of J_2 on RAAN explicit while remaining compatible with the transfer-cost modelling developed later in the report.

The full GVE give the time derivatives of the six elements $(a, e, i, \Omega, \omega, u)$ under J_2 and thrust resolved into a LVLH reference frame[7]:

$$\dot{a} = \frac{2a^2}{h}\left(e\sin\theta\, f_r + \frac{p}{r} f_c\right) \tag{2.1}$$

$$\dot{e} = \frac{1}{h}\left[p\sin\theta\, f_r + \left((p+r)\cos\theta + re\right) f_c\right] \tag{2.2}$$

$$\dot{i} = \frac{r\cos u}{h}\, f_n \tag{2.3}$$

$$\dot{\Omega} = -\frac{3R_E^2\, J_2\, n}{2p^2}\cos i + \frac{r\sin u}{h\sin i}\, f_n \tag{2.4}$$

$$\dot{\omega} = \frac{3R_E^2\, J_2\, n}{4p^2}\left(4 - 5\sin^2 i\right) + \frac{p}{he}\left[\left(1 + \frac{r}{p}\right)\sin\theta\, f_c - \cos\theta\, f_r\right] - \frac{r\sin u}{h\tan i}\, f_n \tag{2.5}$$

$$\dot{u} = \frac{h}{r^2} - \frac{3R_E^2 n\, J_2}{4p^2}\left[\sin^2 i\left(5 - 3\sqrt{1-e^2}\right) + \left(2\sqrt{1-e^2} - 4\right)\right] - \frac{r\sin u}{h\tan i}\, f_n \tag{2.6}$$

The semi-major axis, eccentricity, and inclination have no long-term J_2 drift and depend on thrust only. RAAN carries the J_2 precession that drives the drift strategy of Chapter 3. The argument of perigee and argument of latitude also evolve under J_2, but they do not affect

the cross-plane RAAN alignment used by the mission model, so only RAAN is propagated between transfer legs, as detailed in Chapter 4.

2.3 J_2 Secular Perturbation Implementation

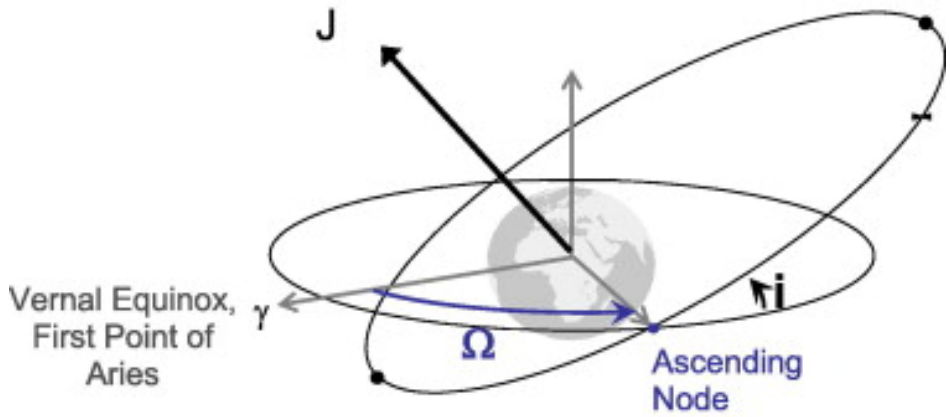

FIGURE 2.2. Eastward RAAN precession of the mothership parking orbit under J_2[12].

Figure 2.2 shows how RAAN changes naturally under J_2 perturbations. This occurs as the Earth is not perfectly spherical, its oblateness causes a torque to be exerted on the orbital plane. This is the phenomenon that is leveraged in this study to close RAAN gaps whilst minimising ΔV expenditure[6].

A sample calculation showing how the mothership's orbital plane precesses in its parking orbit is detailed as follows.

Setting the thrust components to zero in the GVE of Section 2.2 leaves only the J_2 secular terms on Ω, ω, and u. The RAAN rate takes the form

$$\dot{\Omega} = -\frac{3R_E^2 J_2 n}{2p^2}\cos i \tag{2.7}$$

which is negative for prograde and positive for retrograde. Across the filtered catalogue inclination range (96–100°), $\cos i$ is small and negative, so the RAAN precession is a slow

eastward drift. It is the small differential between precession rates at varying altitudes and inclinations that the framework exploits to close RAAN gaps without propellant.

Two limit cases confirm the expression. At $i = 90°$, $\dot{\Omega} = 0$ (no drift for strictly polar orbits), and at $i = 0$, $|\dot{\Omega}|$ is maximised at the equatorial precession rate of the classical literature.

The magnitude can be evaluated explicitly for the mothership parking orbit used throughout this work, with $a = R_E + 680$ km, $e = 10^{-4}$, $i = 98°$, $\Omega_0 = 45°$. The altitude was chosen as it sits close to the middle of the debris catalogue's range, to minimise the distance to be closed by Hohmann transfers. The inclination is also the median of the $96 - 100°$ range to reduce the magnitude of inclination burns.

Regarding placing RAAN at 45°, if the mothership was parked at the densest RAAN cluster (Figure 1.1), the initial RAAN gap to most early targets would be near-zero, and the early legs would collapse to Hohmann transfers with negligible plane changes. The J_2 drift mechanism outlined in this report would never be exercised in the first half of the mission. The 45° offset forces non-trivial transfers from leg 1 to test the effectiveness of the drift strategy in Chapter 4.

The dependent quantities are the semi-latus rectum,

$$p = a(1 - e^2) = 7.058 \times 10^6 \text{ m}, \tag{2.8}$$

and the mean motion,

$$n = \sqrt{\mu/a^3} = 1.065 \times 10^{-3} \text{ rad}\,\text{s}^{-1}, \tag{2.9}$$

corresponding to an orbital period

$$T = \frac{2\pi}{n} = 98.4 \text{ min}. \tag{2.10}$$

Substituting into the RAAN rate expression,

$$\dot{\Omega}_0 = -\frac{3\,(6.378\times10^{6})^2\,(1.08263\times10^{-3})\,(1.065\times10^{-3})}{2\,(7.058\times10^{6})^2}\cos 98^\circ = +1.965\times10^{-7}\ \mathrm{rad\,s^{-1}}, \tag{2.11}$$

and, converted to degrees per day,

$$\dot{\Omega}_0 = +0.973^\circ\ \mathrm{day}^{-1} \quad \text{(eastward)}. \tag{2.12}$$

The debris catalogue spans a narrow band of inclinations and altitudes around the mothership parking orbit, so target precession rates fall within a similarly narrow band of this value. The small differentials between mothership and target rates are the quantities the transfer strategy of Chapter 3 manipulates to close RAAN gaps without propellant.

The argument of perigee has a secular J_2 advance governed by $(4 - 5\sin^2 i)$, vanishing at the critical inclination 63.4°; at the near-polar inclinations of this work it is non-zero but plays no role in the cross-plane RAAN dynamics that drive the transfer strategy. The argument of latitude rate is dominated by the unperturbed h/r^2 and likewise contributes nothing to cross-plane motion.

2.4 ΔV Cost Models

The transfer strategy of Chapter 3 assembles each leg from four impulsive formulae. These are the coplanar Hohmann transfer, the pure inclination change at a node, the pure RAAN correction at altitude, and the combined plane change. Each is set out below in the form implemented in `manoeuvre.py`, with the assumptions under which it is valid. Their composition into the parking–drift–parking leg cost is deferred to Section 3.2.

2.4.1 Hohmann Transfer

Between circular, coplanar orbits of radii r_1 and r_2, the two-burn Hohmann transfer is used to price the altitude changes in this work.[6] The transfer ellipse has semi-major axis $a_t = (r_1 + r_2)/2$, and the two burn magnitudes follow from the vis-viva equation:

$$\Delta V_1 = \left| \sqrt{\mu\left(\tfrac{2}{r_1} - \tfrac{1}{a_t}\right)} - \sqrt{\tfrac{\mu}{r_1}} \right|, \qquad \Delta V_2 = \left| \sqrt{\tfrac{\mu}{r_2}} - \sqrt{\mu\left(\tfrac{2}{r_2} - \tfrac{1}{a_t}\right)} \right|. \tag{2.13}$$

The time of flight is half the transfer-ellipse period, $t_{\text{Hohmann}} = \pi\sqrt{a_t^3/\mu}$. A $680 \to 650\,\text{km}$ descent costs $16.02\,\text{m}\,\text{s}^{-1}$ over 49.0 minutes; this sets the scale for the plane-change costs below.

2.4.2 Inclination Change at a Node

An inclination change $\Delta i = i_2 - i_1$ is performed as a single impulsive rotation of the velocity vector at a nodal crossing. For a circular orbit of speed $v_c = \sqrt{\mu/a}$ the cost is[6]

$$\Delta V_{\Delta i} = \left| 2 v_c \sin\left(\tfrac{\Delta i}{2}\right) \right|. \tag{2.14}$$

For the small corrections of this work ($|\Delta i| \lesssim 1^\circ$) the $\sin(\Delta i/2)$ factor is close to linear, so the cost scales approximately linearly with the inclination difference.

2.4.3 RAAN Change at Altitude

A RAAN change at constant inclination rotates the orbital plane about Earth's polar axis. Under the small-angle, circular-orbit approximation the cost at a nodal crossing is[6]

$$\Delta V_{\Delta\Omega} = \left| v_c \, \Delta\Omega \, \sin i \right|. \tag{2.15}$$

The $\sin i$ factor makes pure RAAN corrections expensive in near-polar orbits. At 650 km and $i \approx 98^\circ$, a 1° RAAN slip costs $\sim 130\,\text{m}\,\text{s}^{-1}$. A 3° gap, would consume close to $400\,\text{m}\,\text{s}^{-1}$ of

a typical $500\,\mathrm{m\,s^{-1}}$ mission budget. This is the physical motivation for closing RAAN gaps passively via J_2 drift rather than impulsively.

2.4.4 Combined Plane Change

When Δi and $\Delta\Omega$ both differ between the initial and final orbits, a single combined burn is cheaper than two sequential ones. The cost is[7]

$$\Delta V_{\Delta i,\Delta\Omega} = v_c\sqrt{(\Delta\Omega \sin i)^2 + (\Delta i)^2}. \tag{2.16}$$

The formula reduces to the pure-inclination expression at $\Delta\Omega = 0$ and to the pure-RAAN expression at $\Delta i = 0$. This is used in Chapter 3 to price the direct transfer against which the J_2-assisted saving is measured.

Chapter 3

J_2 Assisted Transfer Optimisation

3.1 Drift Orbit Selection

The mothership's parking orbit precesses at $+0.973^\circ\,\mathrm{day}^{-1}$ (Section 2.3), but the debris targets precess at their own rates under the same J_2 perturbation. To close the RAAN gap in a reasonable time, the mothership drifts to an intermediate orbit whose altitude and inclination are chosen to produce the required differential $\dot{\Omega}$.

Substituting $n = \sqrt{\mu/a^3}$ and $p \approx a$ into the J_2 expression of Section 2.3 gives $\dot{\Omega} \propto a^{-7/2}\cos i$. The altitude dependence is weak, since a 1% altitude reduction raises $|\dot{\Omega}|$ by only 3.5%. Inclination, by contrast, has a stronger effect at near-polar values where $\cos i$ is small.

Figure 3.1 quantifies both dependencies over representative ranges around the parking orbit. Sweeping altitude from 550–750 km at $i = 98^\circ$ varies $\dot{\Omega}$ from 1.04 to $0.94^\circ\,\mathrm{day}^{-1}$ (a 10% spread); sweeping inclination from 95–105° at 680 km varies it from 0.61 to $1.81^\circ\,\mathrm{day}^{-1}$ (a factor of three). Inclination, as expected, imposes a bigger change on RAAN precession rate; altitude enters the search primarily to trade ΔV against transfer time.

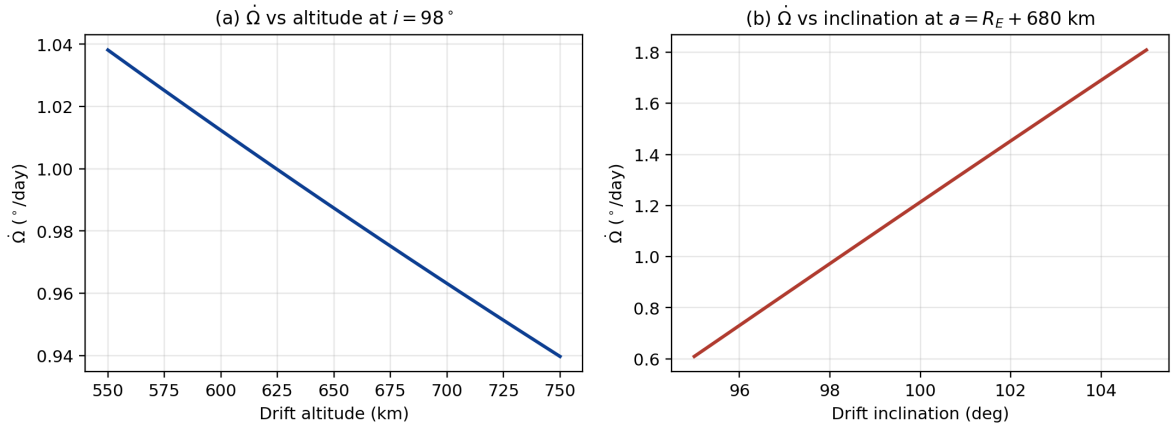

FIGURE 3.1. Sensitivity of the J_2 secular RAAN rate $\dot{\Omega}$ to drift orbit parameters. Figure 3.1a holds inclination at the parking value (98°) and varies altitude over the drift-orbit search range. Figure 3.1b holds altitude at the parking value (680 km) and varies inclination.

The drift rate required to close a RAAN gap $\Delta\Omega$ in time t_{drift} is

$$\dot{\Omega}_{\text{required}} = \dot{\Omega}_{\text{target}} + \frac{\Delta\Omega}{t_{\text{drift}}} \tag{3.1}$$

and the drift inclination that delivers it at a given altitude follows from inverting the J_2 rate expression:

$$\cos i_{\text{drift}} = \frac{\dot{\Omega}_{\text{required}}}{-\frac{3}{2}\, n_{\text{drift}}\, J_2 \left(\frac{R_E}{p_{\text{drift}}}\right)^2} \tag{3.2}$$

The per-leg algorithm is summarised in Figure 3.2.

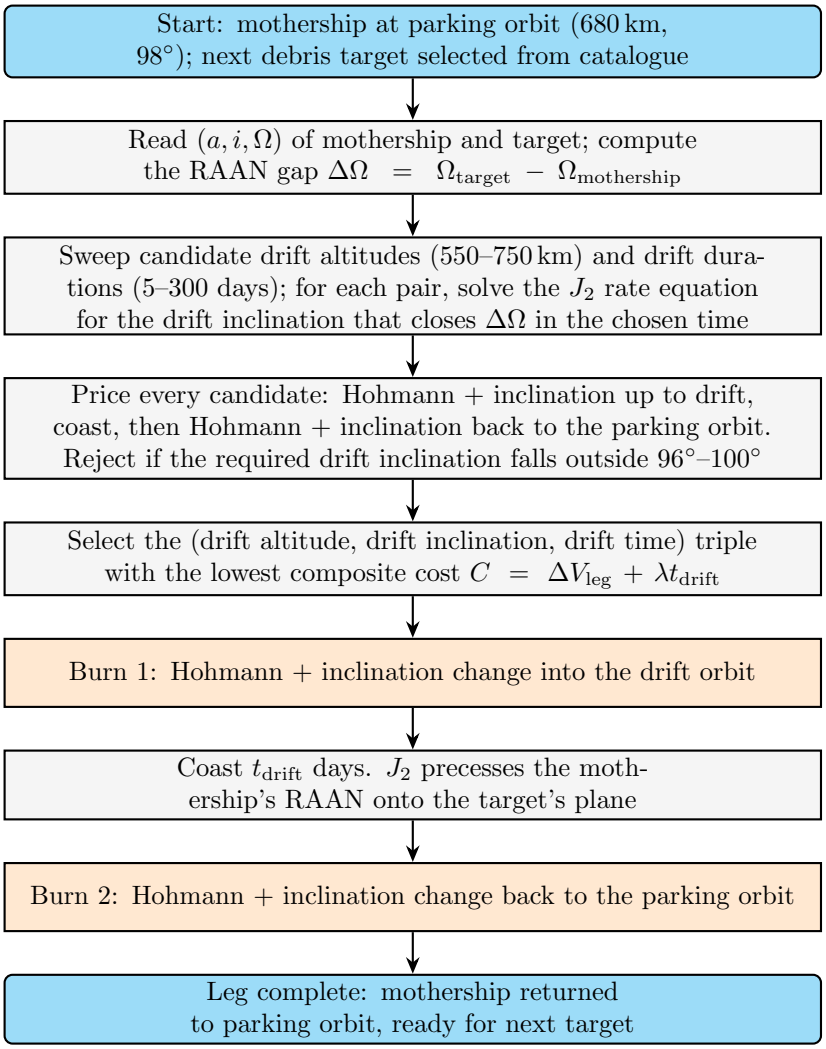

FIGURE 3.2. Per-leg algorithm for aligning the mothership's RAAN with a debris target using J_2-induced drift.

3.2 Transfer Leg Cost Evaluation

This section assembles the total leg cost, introduces a time penalty on long drift durations, and describes the method that selects the cheapest feasible $(a_{\text{drift}}, t_{\text{drift}})$.

A leg consists of two impulsive burns separated by a passive drift. The first manoeuvre transfers the mothership from the parking orbit to the drift orbit and changes its inclination.

The second returns the mothership to the parking orbit once the RAAN gap has closed. The total leg cost is therefore

$$\Delta V_{\text{leg}} = 2\,\Delta V_{\text{Hoh}} + \Delta V_{\Delta i}^{\text{drift}} + \Delta V_{\Delta i}^{\text{park}} \tag{3.3}$$

The Hohmann term appears twice because ascent and descent share the same altitude difference. However, the two inclination burns are priced separately because the first change is performed at the drift orbit and the second at the parking orbit. Since the inclination change cost scales with the circular speed, v_c, a lower drift orbit increases the ΔV cost of this burn. However, $\dot{\Omega} \propto a^{-7/2}$ means that lower drift orbits precess faster. The optimiser therefore trades boundary-burn cost against RAAN drift time.

Fuel cost alone does not rank candidate legs. A 10 m/s leg that takes three months is not chosen over a 15 m/s leg that takes three weeks. Therefore, to address the competing objectives of minimising propellant expenditure and mission duration, this work uses a weighted sum scalarisation of the candidate-leg cost[13]. The composite cost is defined as

$$C = \Delta V_{\text{leg}} + \lambda\, t_{\text{drift}}, \tag{3.4}$$

where the weighting coefficient λ converts drift duration into an equivalent ΔV penalty. This reduces the fuel–time trade-off to a single ranking metric for the greedy sequencer, with larger λ values favouring shorter drift times.

The candidate space is bounded by the SSO altitude window (550–750 km) and a drift duration of 5–300 days. The 300 day upper bound is a mission-planning parameter rather than a physical limit; it prevents the fuel-minimising case from selecting legs with very long drifts and keeps each removal under a year.

Within these bounds, 20 altitude values and 30 drift durations are swept. The codebase accelerates the selection process by assigning infeasible candidates an infinite cost. The optimal altitude-drift pair is the solution that minimises C.

3.3 Single Leg Demonstration and Savings

This section outlines the savings of the method incorporated in this report. The per-leg algorithm of Section 3.2 is compared to a direct impulsive round-trip baseline in which the RAAN and inclination gap is closed by combined plane changes at the parking orbit and then reversed to return the mothership to its original plane.

The mothership starts at $a_{\text{park}} = R_E + 680\,\text{km}$, $i_{\text{park}} = 98°$, $\Omega_{\text{park}} = 45°$. The target is at $a_{\text{target}} = R_E + 650\,\text{km}$, $i_{\text{target}} = 97.8°$, $\Omega_{\text{target}} = 48°$. This corresponds to a RAAN gap of 3° and an inclination difference of 0.2°. Both orbits are near-circular ($e \approx 10^{-4}$).

Without J_2 drift, the mothership would close the gap by executing a single combined plane change at its parking altitude. Substituting the test case into the combined plane change expression of Section 2.4, with $v_{\text{park}} = \sqrt{\mu/a_{\text{park}}} = 7515\,\text{m/s}$, the angular change is

$$\sqrt{(\Delta\Omega \sin i_{\text{park}})^2 + (\Delta i)^2} = 0.0520 \text{ rad}, \tag{3.5}$$

giving a plane-change cost of $\Delta V_{\text{plane}} = v_{\text{park}}\vartheta = 390.5\,\text{m/s}$. To transfer back to the original orbit, the same manoeuvre is repeated giving

$$\Delta V_{\text{direct}} = 781.0 \text{ m/s}. \tag{3.6}$$

Applying the sweeping search of Section 3.2 returns the drift orbit summarised in Table 3.1. Rather than closing the RAAN gap quickly with a large altitude or inclination offset, the planner selects a drift orbit that differs from the parking orbit by only 3.7 km and 0.001°, and holds the mothership there for just over eight months. With $\lambda = 0$, a small deviation requires minimal propellant use, and the 3° gap is closed by letting the small differential rate accumulate passively over a long drift.

TABLE 3.1. Output of the transfer leg cost function (Section 3.2) applied to the mothership–debris pair defined in this section. Values produced by `TransferLeg.optimise()`.

Quantity	**Value**
Drift altitude a_{drift}	$R_E + 676.3\,\text{km}$
Drift inclination i_{drift}	97.999°
Drift duration t_{drift}	259.3 days
Burn 1: Hohmann transfer from parking $\rightarrow$ drift	$1.96\,\text{m/s}$
Burn 1: Inclination change at a_{drift}	$0.14\,\text{m/s}$
Burn 2: Hohmann transfer from drift $\rightarrow$ parking	$1.96\,\text{m/s}$
Burn 2: Inclination change at a_{park}	$0.14\,\text{m/s}$
Total ΔV_{leg}	$\mathbf{4.20}\,\text{m/s}$

The J_2-enhanced transfer therefore costs $\Delta V_{J_2} = 4.20\,\text{m/s}$, a 99.5% saving over the direct baseline of Equation 3.6. With $\lambda = 0$, the fuel cost collapses to a small Hohmann round trip and the plane-change expense is avoided almost entirely.

This saving assumes drift time is free. Raising λ forces the optimiser to close the RAAN gap faster, which moves the drift orbit further from the parking orbit and inflates the boundary-burn ΔV; in the limit $\lambda \rightarrow \infty$ the drift phase vanishes and the cost recovers the direct-transfer value.

Chapter 4

Multi-Target Mission Sequencing

4.1 Greedy Sequencing Algorithm

The general properties of greedy sequencing and its suitability for multi-target debris removal have already been set out in Chapter 1; this section describes how the greedy algorithm is specialised to the mission considered here, namely what the sequencer is given, what it does on each iteration, and when it stops.

The sequencer's inputs are fixed across every run. The mothership begins at the parking orbit of Section 2.3, and its fuel load is expressed as a ΔV budget of 500 m/s, representative of a mothership carrying enough propellant for several rendezvous attempts. The target catalogue is the 637-object Sun-synchronous subset filtered in Chapter 1.

At each iteration, the sequencer prices every unvisited target by invoking the transfer-leg cost function of Section 3.2. That function returns either a feasible leg record or rejection, the latter when the sweeping search finds no admissible drift inclination within the 96–100° band. Each feasible candidate carries both its charged ΔV and the composite ranking cost C of Equation 3.4. The candidates are ranked from cheapest to most expensive by C, the cheapest is selected, and the leg is committed if its ΔV fits within the remainder of the mission budget. The visited target is appended to the sequence, the spent ΔV accumulated, and the loop continues until the ΔV budget is exhausted.

One aspect of this strategy differs from the more common rendezvous-and-capture formulation. At the end of every leg, the mothership returns to its parking-orbit shape rather than being left at the visited target. However, its orbital state is not fully reset because, during the committed drift interval, both the mothership and every unvisited target precess in RAAN under J_2, so the next iteration is priced from the propagated RAAN geometry. In the reduced secular model adopted here, the semi-major axis, eccentricity, and inclination are held fixed between legs and only RAAN is advanced, which is sufficient for the drift-based sequencing objective of this dissertation. This choice reflects the wider mission concept, as after a chaser is released, it is assumed to return to the mothership, hence the mothership's initial orbit being partially restored after each leg.

The argument of perigee and argument of latitude are not propagated because they affect in-plane geometry and phasing, rather than the orientation of the orbital plane priced by the transfer-leg model. Including them would not change the leg ranking unless rendezvous timing and capture dynamics were also modelled, both of which falls outside the scope of this work outlined in Section 1.5.2.

Require: M: mothership parking state (Section 2.3)
Require: T: filtered debris catalogue (Chapter 1)
Require: $B = 500\,\mathrm{m/s}$: ΔV budget
Require: $\lambda \geq 0$: time-penalty coefficient (Section 3.2)
Ensure: visited targets, total ΔV spent, per-leg log

Visited $\leftarrow$ [ ]; Spent $\leftarrow 0$; Log $\leftarrow$ [ ]
Remaining $\leftarrow$ copy of T
while Remaining is not empty **do**
    Candidates $\leftarrow$ [ ]
    for all targets t in Remaining **do**
        leg $\leftarrow$ TransferLeg(M, t, λ) ▷ Section 3.2
        if leg.feasible **then** Candidates.append$((t, \text{leg}))$
        end if
    end for
    if Candidates is empty **then break** ▷ no feasible target remains
    end if
    sort Candidates ascending by $C = \Delta V_{\text{leg}} + \lambda t_{\text{drift}}$
    $(t^\star, \text{leg}^\star) \leftarrow$ Candidates[0] ▷ lowest composite-cost leg
    if $\text{leg}^\star.\Delta V_{\text{leg}} > B - \text{Spent}$ **then break** ▷ budget exhausted
    end if
    Visited.append$(t^\star)$
    Spent $\leftarrow$ Spent $+ \text{leg}^\star.\Delta V_{\text{leg}}$; Log.append$(\text{leg}^\star)$
    advance $M.\Omega$ over $\text{leg}^\star.t_{\text{drift}}$
    advance every remaining target RAAN over $\text{leg}^\star.t_{\text{drift}}$
    Remaining.remove$(t^\star)$
end while
return Visited, Spent, Log

FIGURE 4.1. Greedy mission sequencer. The outer loop advances one target per iteration. Each candidate target is passed to the per-leg cost function of Section 3.2, which performs the $(a_{\text{drift}}, t_{\text{drift}})$ sweeping search and returns a feasible leg record with a ΔV cost, drift time, and component breakdown.

The loop terminates under one of two conditions. In the common case, the lowest composite-cost remaining feasible leg eventually exceeds the ΔV that is still available. Alternatively, the loop exits when no remaining target has a feasible drift solution at all, which occurs when every candidate's required drift inclination falls outside the admissible band and the sweeping search therefore rejects every candidate. Either way, the returned record contains the visited sequence, the total ΔV spent, and the per-leg log that feeds the mission-level results of Section 4.2.

4.2 Mission Results

Executing the sequencer of Section 4.1 on the filtered 637-object catalogue with $\lambda = 0$ and the 500 m/s budget removes 13 debris objects for a total expenditure of 363.0 m/s.

TABLE 4.1. Per-leg output of `run_mission()`. Each row corresponds to one committed transfer leg; the NORAD identifier distinguishes debris objects with the same catalogue name. The cumulative column tracks the spent ΔV against the budget. All ΔV in m/s; drift time in days.

Step	Object	NORAD	ΔV_{leg}	t_{drift}	$\sum \Delta V$
1	NOAA 17 DEB	48729	3.9	168	3.9
2	FENGYUN 1C DEB	31296	5.2	15	9.2
3	DELTA 1 DEB	19105	5.8	239	15.0
4	DELTA 1 DEB	42558	4.5	158	19.5
5	DELTA 1 DEB	18475	4.8	86	24.3
6	STRIX-BETA DEB	53460	4.6	290	29.0
7	SL-24 DEB	37796	11.1	300	40.0
8	DELTA 1 DEB	42555	30.3	300	70.3
9	DELTA 1 DEB	8958	40.6	300	110.9
10	DELTA 1 DEB	8951	7.6	290	118.5
11	COSMOS 2535 DEB	44640	77.3	300	195.8
12	CZ-4C DEB	43263	89.2	300	285.0
13	COSMOS 2535 DEB	44633	78.0	300	363.0

The per-leg cost in Table 4.1 does not show a monotonic increase in ΔV. The first six legs sit between 3.9 and 5.8 m/s before jumping to 11.1 m/s at leg 7, after which there is a general increase to mission end, aside from the drop at leg 10. This drop is the direct signature of the inter-leg RAAN propagation, since the target selected at leg 10 was not

the cheapest available at step 1 but became so once the catalogue had precessed by the accumulated drift durations of the first nine legs.

Table 4.2 shows the orbital-element offsets between each visited target and the mothership's parking orbit (Δh, Δi, $\Delta\Omega_{\text{init}}$), evaluated at the initial mission epoch. $\Delta\Omega_{\text{prop}}$ shows the accumulated RAAN propagation per target, which is the accumulation of the precession after the previous debris targets have been removed. The inclination offsets in Table 4.2 are uniformly small ($|\Delta i| < 1.1°$), reflecting the narrow inclination band of the filtered Sun-synchronous catalogue; the variation between legs therefore lives almost entirely in the RAAN gap. The three expensive legs at the bottom of Table 4.1, legs 11 through 13, are the consequence of the drift cap being hit. Since this report sets an upper bound of 300 days, the sweeping search is forced to give a solution with a higher ΔV to close the RAAN gap.

Across the thirteen legs, the 363 m/s total divides almost evenly between Hohmann transfers (170.3 m/s) and inclination burns (192.7 m/s). However, the composition is not uniform across the mission. The early legs (1–6) are dominated by Hohmann transfers, as the optimiser holds drift altitude at a single discrete value 3.7 km below parking and drift inclination within 0.01° of the parking value; their Hohmann sum is 23.5 m/s, against only 5.4 m/s of inclination cost. From here onwards, the legs are largely inclination-dominated, due to the drift cap being imposed on all legs aside from leg 10.

TABLE 4.2. Orbital-element offsets of each visited target with respect to the mothership's parking orbit. The NORAD identifier distinguishes debris objects with the same catalogue name. Δh and Δi are time-invariant; $\Delta\Omega_{\text{init}}$ is the RAAN gap at the initial mission epoch, and $\Delta\Omega_{\text{prop}}$ is the propagated RAAN gap at the moment the leg is priced, after the catalogue has precessed by the drift durations of all preceding legs. Rows are presented in the same order as Table 4.1.

Step	Object	NORAD	Δh (km)	Δi (°)	$\Delta\Omega_{\text{init}}$ (°)	$\Delta\Omega_{\text{prop}}$ (°)
1	NOAA 17 DEB	48729	−9.4	+0.48	−10.18	−10.18
2	FENGYUN 1C DEB	31296	−50.5	+1.06	−28.58	−2.35
3	DELTA 1 DEB	19105	+20.5	+0.75	−32.84	−18.33
4	DELTA 1 DEB	42558	−10.9	−0.12	+5.86	+1.83
5	DELTA 1 DEB	18475	+19.1	−0.12	+16.02	+2.18
6	STRIX-BETA DEB	53460	−114.6	−0.21	−28.62	−8.38
7	SL-24 DEB	37796	−2.6	−0.05	+7.86	+3.04
8	DELTA 1 DEB	42555	−28.3	−0.21	+22.35	+7.69
9	DELTA 1 DEB	8958	+1.6	+0.04	−12.10	−6.63
10	DELTA 1 DEB	8951	−18.7	+0.06	−36.13	−5.74
11	COSMOS 2535 DEB	44640	−47.5	−0.24	+26.59	+12.67
12	CZ-4C DEB	43263	−51.7	−0.31	+46.83	+16.16
13	COSMOS 2535 DEB	44633	−57.5	−0.28	+29.06	+12.62

The run also exposes what the greedy architecture does not achieve. Thirteen objects are removed from the filtered catalogue, approximately 2% of the Sun-synchronous debris population of Chapter 1, and the greedy rule has no guarantee that the sequence it returns is the cheapest one available. Each leg is chosen for its own cost alone, with no regard for the price of future legs, so a different ordering of the same 13 targets, one selected by evaluating all 13 legs at once rather than sequentially, could in principle remove a similar number of debris for a smaller ΔV spend.

The results of Table 4.1 use the fuel-minimising choice $\lambda = 0$. The composite cost of Equation 3.4 was introduced to expose the trade between fuel and duration, and that trade is quantified by re-running the sequencer across the sweep of λ values in Table 4.3. All runs share the 500 m/s budget and the filtered catalogue of Section 4.1; only the weighting of drift time in the cost function varies.

TABLE 4.3. Headline mission outcome as a function of the time-penalty coefficient λ. All runs use the 500 m/s budget and the filtered 637-object catalogue of Section 4.1. The baseline row ($\lambda = 0$) reproduces the per-leg sequence of Table 4.1.

λ (m s^{-1} day^{-1})	Targets removed	Total ΔV (m/s)	Total drift (days)	Duration (years)
0.00	13	363.0	3046	8.34
0.05	16	383.6	3050	8.35
0.10	19	434.5	2984	8.17
0.25	20	477.6	2511	6.87
0.50	18	459.8	1616	4.42
0.75	19	480.7	1041	2.85
1.00	18	448.8	843	2.31
1.25	18	397.1	609	1.67
1.50	19	440.7	604	1.65
1.75	19	464.4	593	1.62
2.00	18	477.2	487	1.33

Table 4.3 shows that the trade is not monotonic. Very small time penalties do not shorten the mission appreciably, but they do restructure the sequence, so raising λ from 0 to 0.05 increases the number of removals from 13 to 16 while leaving the duration essentially unchanged. By $\lambda = 0.25$ the sequencer removes 20 targets in 6.87 years for 477.6 m/s. In this low-λ regime the optimiser still tolerates long drifts, but the composite ranking metric

steers the greedy rule toward chains of targets that are cheaper in the mission-level sense than the purely fuel-minimising baseline.

Beyond $\lambda = 0.25$ the duration reduction steepens. Raising λ to 0.50 cuts the mission to 4.42 years with 18 removals, and by $\lambda = 0.75$ the duration has fallen to 2.85 years while still preserving 19 removals. The strongest compression occurs in the range $\lambda = 1.25$–2.00, where the mission duration falls to 1.33–1.67 years and the target count remains between 18 and 19. The sequence remains non-monotonic because the propagated-catalogue architecture couples every choice to the geometry of the remaining targets, so once the first few selections change, the downstream chain restructures as well.

As this is a conceptual mission-planning study, a 5% reserve above the expended ΔV is applied[14]. Applying this margin to the 500 m/s budget sets a ceiling on the total expenditure at 476.2 m/s. Table 4.3 places $\lambda = 0.25$, $\lambda = 0.75$, and $\lambda = 2.00$ outside this ceiling. The baseline $\lambda = 0$ configuration is comfortably compliant with 137.0 m/s of reserve, while the range $\lambda = 1.50$–1.75 emerges as the strongest time-constrained region among the compliant runs because both remove 19 targets in about 1.6 years, with total expenditures of 440.7 and 464.4 m/s respectively.

The $\lambda = 1.50$ run is selected as the headline solution for this dissertation, by virtue of having a lower ΔV at the expense of an eleven day increased duration compared to $\lambda = 1.75$. The per-leg sequence of this run is shown in Table 4.4.

Compared with the fuel-minimising baseline of Table 4.1, the selected case removes six additional targets and reduces the mission duration from 8.34 years to 1.65 years, albeit at the cost of an additional 77.7 m/s of ΔV. The trade-off is that several individual legs are more expensive, especially the later transfers, because the optimiser accepts higher manoeuvre costs over long passive RAAN alignment.

TABLE 4.4. Selected mission sequence for $\lambda = 1.5\ \mathrm{m\,s^{-1}\,day^{-1}}$. All ΔV in m/s; drift time in days.

Step	Object	NORAD	ΔV_{leg}	t_{drift}	$\sum \Delta V$
1	DMSP 5D-2 F11 DEB	28331	13.7	5	13.7
2	NOAA 17 DEB	48766	16.0	36	29.7
3	CZ-4 DEB	26194	23.5	15	53.2
4	PSLV DEB	27123	16.9	15	70.1
5	THORAD AGENA D DEB	4653	11.0	25	81.1
6	FENGYUN 1C DEB	30628	4.1	46	85.2
7	DELTA 1 DEB	7892	33.2	15	118.5
8	NOAA 17 DEB	48729	25.4	5	143.9
9	FENGYUN 1C DEB	31631	23.4	25	167.3
10	FENGYUN 1C DEB	30106	18.4	25	185.6
11	NOAA 16 DEB	41372	14.6	25	200.3
12	FENGYUN 1C DEB	30141	55.3	15	255.5
13	NOAA 16 DEB	42371	12.8	56	268.3
14	FENGYUN 1C DEB	31376	48.7	15	317.0
15	FENGYUN 1C DEB	30838	22.7	15	339.7
16	DELTA 1 DEB	19105	4.0	76	343.7
17	FENGYUN 1C DEB	31202	9.8	76	353.5
18	DELTA 1 DEB	42558	34.5	46	388.0
19	DELTA 1 DEB	18475	52.7	66	440.7

Chapter 5

Conclusion

This dissertation develops a mission sequencer model for planning multi-target active debris removal missions in Sun-synchronous orbit, in which the cost of closing the right ascension of the ascending node (RAAN) between successive targets is delivered by secular J_2 precession rather than impulsive plane changes. The method comprises an analytically priced transfer-leg model that selects a drift altitude and duration through a sweeping search over the debris catalogue, and a greedy sequencer that orders the legs. The per-leg saving is decisive, with the report's method leading to a saving of 776.8 m/s, a 99.5% reduction.

The composite cost sweep of Section 4.2 quantifies the trade between propellant and mission duration. The fuel-minimising baseline at $\lambda = 0$ removes 13 debris objects for 363.0 m/s, but requires 3046 drift days, or 8.34 years. Introducing a time penalty both restructures the greedy sequence and compresses the mission duration substantially. Among the compliant time-constrained runs, $\lambda = 1.50$ is considered the primary solution. It removes 19 debris objects in 1.65 years for 440.7 m/s, while still preserving a comfortable propellant reserve under the conceptual design ΔV margin[14].

A methodological limitation of this work is that the greedy sequencer has no lookahead. Each transfer leg is priced in isolation, so the selected target at the time may not have led to a cheaper sequence later in the mission. Running the same mission with an alternative sequencer on a smaller debris dataset, could reveal a solution with a lower ΔV or a higher removal count.

Future work could explore an architectural change to the deployment concept. Holding the mothership at the drift orbit after burn 1 (Figure 3.2) and releasing the chaser satellites would increase the number of removals per unit ΔV even further, since the mothership would not have to return to the parking orbit prior to releasing the chasers. However, a challenge of this approach would be modelling the chaser-to-mothership rendezvous to ensure the chasers can be reused.

Bibliography

[1] Y. Liu, Y. Jiang, and H. Li, "Analytical Propagation of Space Debris Density for Collisions near Sun-Synchronous Orbits," *Space: Science & Technology*, vol. 2022, Sep. 2022. [Online]. Available: https://spj.science.org/doi/10.34133/2022/9825763

[2] M. Arshad, M. C. F. Bazzocchi, and F. Hussain, "Emerging strategies in close proximity operations for space debris removal: A review," *Acta Astronautica*, vol. 228, pp. 996–1022, Mar. 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576524007665

[3] J. Nicholas, "The Collision of Iridium 33 and Cosmos 2251: The Shape of Things to Come," Seoul, Oct. 2009, nTRS Author Affiliations: Johnson Space Center NTRS Report/Patent Number: Report Number: JSC-CN-18971 NTRS Document ID: 20100002023 NTRS Research Center: Johnson Space Center (JSC). [Online]. Available: https://ntrs.nasa.gov/citations/20100002023

[4] L. Anselmo and C. Pardini, "Analysis of the Consequences in Low Earth Orbit of the Collision Between Cosmos 2251 and Iridium 33." [Online]. Available: https://www.researchgate.net/publication/228975104_Analysis_of_the_Consequences_in_Low_Earth_Orbit_of_the_Collision_Between_Cosmos_2251_and_Iridium_33

[5] J.-H. Choi, C. Park, and J. Lee, "Mission planning for active removal of multiple space debris in low Earth orbit," *Advances in Space Research*, vol. 73, no. 9, pp. 4800–4812, 2024.

[6] David A. Vallado, *Fundamentals of Astrodynamics and Applications, 4th ed.*, 4th ed. Microcosm Press, 2013.

[7] M. A. Alandihallaj and M. R. Emami, "CubeSat Orbit Insertion Maneuvering Using J2 Perturbation," May 2025. [Online]. Available: http://xplorestaging.ieee.org/ielx8/11068190/11068395/11068444.pdf?arnumber=11068444

[8] J. Zhang, H. Huang, and X. Wang, "Resource provision algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 64, pp. 23–42, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804516000643

[9] S. Liu, S. Hu, J. Du, H. Cao, B. Zhang, Y. Jiang, and S. Feng, "Space debris sky survey observation strategy based on healpix and greedy algorithm," *Aerospace*, vol. 12, no. 3, 2025. [Online]. Available: https://www.mdpi.com/2226-4310/12/3/168

[10] D. Nguyen, A. Nguyen, and L. Nguyen, "The application of a greedy algorithm to search-and-rescue trajectory planning for UAV systems," *International Journal of Sustainable Aviation*, vol. 11, Jan. 2025.

[11] A. Ogundele, "Nonlinear Dynamics and Control of Spacecraft Relative Motion," Ph.D. dissertation, Aug. 2017.

[12] R. P. Patera, "Energy angular momentum closed-loop guidance," *Advances in Space Research*, vol. 55, no. 5, pp. 1495–1511, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0273117714007959

[13] A. Shirazi, J. Ceberio, and J. A. Lozano, "Spacecraft trajectory optimization: A review of models, objectives, approaches and solutions," *Progress in Aerospace Sciences*, vol. 102, pp. 76–98, Oct. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0376042118300198

[14] European Cooperation for Space Standardization, "ECSS-E-TM-10-25A – Engineering design model data exchange," Oct. 2010. [Online]. Available: https://ecss.nl/hbstms/ecss-e-tm-10-25a-engineering-design-model-data-exchange-cdf-20-october-2010/

Chapter A

Source Code Repository

The complete source code for the orbital mechanics simulation model developed in this work is publicly available on GitHub.

The repository implements a modular Python codebase for modelling active debris removal (ADR) mission planning in Sun-synchronous (SSO) low Earth orbit (LEO), exploiting J_2 perturbations to minimise fuel cost. The source files listed below are the modules used directly in the methodology and mission results of this dissertation.

Dependencies are NumPy, SciPy, and pandas. All simulations were executed using Python 3.14.3.

TABLE A.1. Repository structure and module descriptions.

Module	**Description**
`constants.py`	Defines physical constants and unit conversion factors used across the model.
`orbitalstate.py`	Stores the classical orbital state and derived quantities including mean motion, circular velocity, and J_2 RAAN precession rate.
`propagator.py`	Numerically integrates Gauss' Variational Equations using DOP853 and validates the analytical J_2 RAAN drift.
`manoeuvre.py`	Implements the impulsive ΔV cost models used for Hohmann transfers, inclination changes, RAAN changes, combined plane changes, and full transfer-leg costing.
`transferleg.py`	Searches drift altitude and duration, then solves the required drift inclination from the J_2 rate equation.
`mission.py`	Runs the greedy multi-target sequencer and generates the baseline mission sequence.
`timepenalty.py`	Re-runs the greedy sequencer across λ values to generate the time-penalty sweep.
`spacetrackdata.py`	Inspects the filtered Space-Track.org debris catalogue and reports the orbital-element ranges used by the mission model.

A.1 Python Source Listings

A.1.1 constants.py

```
# Shared constants
# Physical constants used throughout the ADR mission model.
# All values are stored in SI units unless the name says otherwise.

import numpy as np

"""
1) Define the Earth and gravity constants that are reused by the orbital
state, propagator, manoeuvre, transfer-leg, and mission-sequencing modules.

2) Keep the conversion factors in one place so the rest of the code can
convert between catalogue units (km and degrees) and propagation units
(metres and radians) without duplicating constants.
"""

# Standard gravitational parameter of Earth
MU = 3.986004418e14  # m^3/s^2
R_E = 6378.137e3  # m (WGS-84 equatorial radius)
J2 = 1.08263e-3  # Dimensionless

# Unit conversion factors
DEG_TO_RAD = np.pi / 180.0
RAD_TO_DEG = 180.0 / np.pi
DAY_TO_SEC = 86400.0
SEC_TO_DAY = 1.0 / DAY_TO_SEC
KM_TO_M = 1000.0
M_TO_KM = 0.001
```

LISTING A.1. constants.py

A.1.2 orbitalstate.py

```
import numpy as np
from constants import MU, R_E, J2, RAD_TO_DEG, DAY_TO_SEC

"""
1) OrbitalState is the common container for the six classical orbital
elements used throughout the project. Angles are stored in radians and
semi-major axis is stored in metres.

2) The class also provides derived quantities such as mean motion, orbital
period, circular velocity, and the secular J2 RAAN precession rate. This
keeps the equations in the transfer and mission modules shorter and reduces
the chance of unit mistakes.

3) The to_vector and from_vector helpers exist because scipy's ODE solver
expects raw numpy arrays, while the rest of the code is easier to read when
using named orbital-state fields.
"""

class OrbitalState:
  def __init__(self, a, e, i, raan, omega, u, epoch=0.0):
    self.a = a # Semi-major axis (m)
    self.e = e # Eccentricity (Dimensionless)
    self.i = i # Inclination (rad)
    self.raan = raan # Right Ascension of Ascending Node (rad)
    self.omega = omega # Argument of perigee (rad)
    self.u = u # Argument of latitude (rad)
    self.epoch = epoch # Time (s)

  @property
  def n(self):
    """Mean motion (rad/s)."""
    return np.sqrt(MU / self.a**3)

  @property
  def period(self):
    """Orbital period (s)."""
    return 2 * np.pi / self.n

  @property
  def p(self):
    """Semi-latus rectum (m)."""
    return self.a * (1 - self.e**2)

  @property
  def h(self):
    """Specific angular momentum (m^2/s)."""
    return np.sqrt(MU * self.p)

  @property
  def v_circular(self):
    """Circular orbital velocity (m/s)"""
    return np.sqrt(MU / self.a)

  @property
  def altitude(self):
```

```
56      """Altitude above Earth surface (m)."""
57      return self.a - R_E
58
59    @property
60    def raan_dot_j2(self):
61      """Secular J2 RAAN precession rate (rad/s)."""
62      return -1.5 * self.n * J2 * (R_E / self.p)**2 * np.cos(self.i)
63
64    # Turns the state into the raw array format expected by the ODE integrator
65    def to_vector(self):
66      """Return state as numpy array (a, e, i, raan, omega, u)."""
67      return np.array([self.a, self.e, self.i, self.raan, self.omega, self.u])
68
69    @classmethod
70    def from_vector(cls, vec, epoch=0.0):
71      """Create OrbitalState from numpy array (a, e, i, raan, omega, u)."""
72      return cls(a=vec[0], e=vec[1], i=vec[2], raan=vec[3], omega=vec[4], u=vec[5], epoch=epoch)
73
74    def copy(self):
75      """Return a copy of the current state."""
76      return OrbitalState(self.a, self.e, self.i, self.raan, self.omega, self.u, self.epoch)
```

LISTING A.2. orbitalstate.py

A.1.3 propagator.py

```
# Phase 1 - Orbital Propagator using J2 perturbation

import numpy as np
from scipy.integrate import solve_ivp
from constants import MU, R_E, J2, RAD_TO_DEG, DAY_TO_SEC
from orbitalstate import OrbitalState

"""
1) GVEPropagator integrates Gauss's Variational Equations for the orbital
elements (a, e, i, RAAN, omega, u). This is the full element-level model
used to validate the secular J2 behaviour described in the dissertation.

2) The thrust_func argument is optional. When it is not supplied, all thrust
components are set to zero and the propagation reduces to the natural J2
drift used to check the analytical RAAN precession rate.

3) The __main__ block is a representative test case. It propagates a
Sun-synchronous orbit for 30 days and compares the numerical RAAN drift
against the analytical secular expression from OrbitalState.raan_dot_j2.
"""

class GVEPropagator:
    def __init__(self, rtol=1e-10, atol=1e-12, max_step=300):
        self.rtol = rtol # Relative tolerance
        self.atol = atol # Absolute tolerance
        self.max_step = max_step # Maximum step size (s)

    def _gve_rhs(self, t, y, thrust_func):
        a, e, i, raan, omega, u = y

        # Common orbital quantities used by the GVE
        p = a * (1 - e**2) # Semi-latus rectum
        n = np.sqrt(MU / a**3) # Mean motion
        h = np.sqrt(MU * p) # Specific angular momentum
        theta = u - omega  # True anomaly
        r = p / (1 + e * np.cos(theta))  # Radius

        # Thrust accelerations in the LVLH frame. They are zero for the
        # J2-only validation runs used in this dissertation.
        if thrust_func is not None:
            f_r, f_c, f_n = thrust_func(t, y)  # Radial, tangential, normal thrust components
        else:
            f_r, f_c, f_n = 0.0, 0.0, 0.0

        sin_i = np.sin(i)
        cos_i = np.cos(i)
        sin_u = np.sin(u)
        cos_u = np.cos(u)
        sin_theta = np.sin(theta)
        cos_theta = np.cos(theta)

        # Eq (1): da/dt
        da = (2 * a**2 / h) * (e * sin_theta * f_r + (p / r) * f_c)

        # Eq (2): de/dt
```

```
        de = (1 / h) * (p * sin_theta * f_r
                        + ((p + r) * cos_theta + r * e) * f_c)

        # Eq (3): di/dt
        di = (r * cos_u / h) * f_n

        # Eq (4): dΩ/dt
        draan = (-3 * R_E**2 * J2 * n / (2 * p**2)) * cos_i
        if abs(sin_i) > 1e-12:
            draan += (r * sin_u / (h * sin_i)) * f_n

        # Eq (5): dω/dt
        domega = (3 * R_E**2 * J2 * n / (4 * p**2)) * (4 - 5 * sin_i**2)
        if abs(e) > 1e-10:
            domega += (p / (h * e)) * (
                (1 + r / p) * sin_theta * f_c - cos_theta * f_r
            )
        if abs(np.tan(i)) > 1e-12:
            domega -= (r * sin_u / (h * np.tan(i))) * f_n

        # Eq (6): du/dt
        e_factor = np.sqrt(1 - e**2)
        du = h / r**2 \
             - (3 * R_E**2 * n * J2 / (4 * p**2)) * (
                 sin_i**2 * (5 - 3 * e_factor)
                 + (2 * e_factor - 4))
        if abs(np.tan(i)) > 1e-12:
            du -= (r * sin_u / (h * np.tan(i))) * f_n

        return np.array([da, de, di, draan, domega, du])

    def propagate(self, initial_state, duration, thrust_func=None):
        """Propagate an OrbitalState for the requested duration in seconds."""
        y0 = initial_state.to_vector()

        sol = solve_ivp(
            fun=lambda t, y: self._gve_rhs(t, y, thrust_func),
            t_span=(0.0, duration),
            y0=y0,
            method='DOP853', # 8th order method
            rtol=self.rtol,
            atol=self.atol,
            max_step=self.max_step,
            dense_output=True
        )

        if not sol.success:
            raise RuntimeError(f"Integration failed: {sol.message}")

        final_state = OrbitalState.from_vector(
            sol.y[:, -1],
            epoch=initial_state.epoch + duration
        )

        return final_state, sol

if __name__ == "__main__":
```

```
    # Define a sun-synchronous orbit at 700 km
    state = OrbitalState(
        a=R_E + 700e3,
        e=0.0001,
        i=np.radians(98.19),
        raan=np.radians(45.0),
        omega=0.0,
        u=0.0
    )

    print(f"Initial orbit: {state.altitude/1e3:.0f} km, "
          f"i = {np.degrees(state.i):.2f} deg")
    print(f"Analytical RAAN rate: "
          f"{state.raan_dot_j2 * RAD_TO_DEG * DAY_TO_SEC:.6f} deg/day")

    # Propagate 30 days under J2 only (no thrust)
    prop = GVEPropagator()
    final, sol = prop.propagate(state, 30 * DAY_TO_SEC)

    # Compare numerical result to analytical prediction
    numerical = np.degrees(final.raan - state.raan)
    analytical = np.degrees(state.raan_dot_j2) * 30 * DAY_TO_SEC

    print(f"\n30-day RAAN drift:")
    print(f"Numerical (GVE): {numerical:.6f} deg")
    print(f"Analytical (secular): {analytical:.6f} deg")
    print(f"Difference: {abs(numerical - analytical):.8f} deg")

    # Check for 0 change in a, e, i
    print(f"\nChecks for 0 thrust propagation:")
    print(f"Delta-a: {(final.a - state.a)/1e3:.6f} km")
    print(f"Delta-e: {final.e - state.e:.8f}")
    print(f"Delta-i: {np.degrees(final.i - state.i):.8f} deg")
```

LISTING A.3. propagator.py

A.1.4 manoeuvre.py

```
# Manoeuvre cost models
# Equations used to price the impulsive burns in each transfer leg.

import numpy as np
from constants import MU, R_E, J2

"""
1) This module contains the analytical Delta-V formulae used in the report:
Hohmann transfer, pure inclination change, pure RAAN change, combined plane
change, and the assembled parking-drift-parking transfer leg.

2) The functions assume circular or near-circular orbits and impulsive burns.
Those assumptions match the scope stated in the dissertation and keep the
mission-level optimiser focused on cross-plane RAAN alignment rather than
full rendezvous dynamics.

3) compute_leg_dv is the function called by transferleg.py. It prices the
mothership moving from parking to a drift orbit, waiting for J2 to close the
RAAN gap, and returning to the parking-orbit shape.
"""

def hohmann_delta_v(r1, r2):
    """Calculate the delta-V for a Hohmann transfer between two circular orbits."""
    # Calculate the semi-major axis of the transfer orbit
    a_transfer = (r1 + r2) / 2

    # Vis-Viva equation for a circular orbit. It gives the velocity of the mothership in its
     initial circular orbit.
    v1_circular = np.sqrt(MU / r1)
    # Full Vis-Viva equation. Gives the velocity needed at radius r1 to enter the transfer orbit.
    v1_transfer = np.sqrt(MU * (2 / r1 - 1 / a_transfer))

    dv1 = abs(v1_transfer - v1_circular)  # Delta-V for the first burn

    v2_circular = np.sqrt(MU / r2)  # Velocity of the target in its circular orbit
    v2_transfer = np.sqrt(MU * (2 / r2 - 1 / a_transfer))  # Velocity needed at radius r2 to enter
     the transfer orbit
    dv2 = abs(v2_circular - v2_transfer)  # Delta-V for the second burn

    # The value I'm interested in is the sum. The breakdown is helpful for debugging
    return dv1, dv2, dv1 + dv2

def hohmann_transfer_time(r1, r2):
    """Calculate the time of flight for a Hohmann transfer between two circular orbits."""
    # A Hohmann transfer traverses exactly half the transfer ellipse, so the flight time is half
     the orbital period
    a_transfer = (r1 + r2) / 2
    period_transfer = 2 * np.pi * np.sqrt(a_transfer**3 / MU)
    return period_transfer / 2  # Time to go from r1 to r2 is half the period of the transfer orbit

"""
1) A pure inclination change is modelled as a single impulsive rotation of
the velocity vector at a node. This is the inclination correction used when
the mothership enters and exits the selected drift orbit.
"""
```

```
53
54 def pure_inclination_dv(v_circular, delta_i):
55     """Calculate the delta-V for a pure inclination change at a node."""
56     return abs(2 * v_circular * np.sin(delta_i / 2))
57
58 """
59 1) A pure RAAN change is expensive in near-polar orbit because the burn has
60 to rotate the orbital plane about Earth's polar axis. The report uses this
61 function to show why passive J2 drift is preferable to impulsive RAAN
62 correction.
63 """
64
65 def pure_raan_change_dv(v_circular, delta_raan, inclination):
66     return abs(v_circular * delta_raan * np.sin(inclination))
67
68 """
69 1) The combined plane-change formula prices a single burn that changes RAAN
70 and inclination together under the small-angle circular-orbit approximation.
71 It is used for the direct no-J2 baseline comparison.
72 """
73
74 def combined_plane_change_dv(v_circular, delta_i, delta_raan, inclination):
75     """Calculate the delta-V for a combined plane change at a node."""
76     # This is an approximation that assumes small angles. For larger angles, the geometry becomes
       more complex.
77     plane_change_angle = np.sqrt(delta_i**2 + (delta_raan * np.sin(inclination))**2)
78     dv = abs(v_circular * plane_change_angle)
79
80     if abs(delta_i) > 1e-14:
81         u_star = np.arctan2(delta_raan * np.sin(inclination), delta_i)
82     else:
83         u_star = np.pi / 2 if delta_raan > 0 else -np.pi / 2
84         # The nested if statement prevents division by 0 in cases there may be 0 inclination change
85
86     # Return both the cost and the optimal burn location
87     return dv, u_star
88
89 def compute_leg_dv(a_mothership, i_mothership, a_drift, i_drift, a_target,
90                    i_target, delta_raan_residual):
91     """Compute the full parking-drift-parking leg cost and return a breakdown."""
92     # Mothership: parking -> drift
93     _, _, dv_hohmann_to_drift = hohmann_delta_v(a_mothership, a_drift)
94     v_at_drift = np.sqrt(MU / a_drift)
95     dv_inc_to_drift = pure_inclination_dv(v_at_drift, i_drift - i_mothership)
96
97     # Mothership: drift -> parking (returns to its own orbit, not the target)
98     _, _, dv_hohmann_return = hohmann_delta_v(a_drift, a_mothership)
99     v_at_mothership = np.sqrt(MU / a_mothership)
100    dv_inc_return = pure_inclination_dv(v_at_mothership, i_mothership - i_drift)
101
102    # Residual RAAN term is retained for completeness. In the nominal
103    # optimiser this is zero because transferleg.py solves the drift rate
104    # required to close the RAAN gap exactly.
105    v_at_target = np.sqrt(MU / a_target)
106    dv_residual = pure_raan_change_dv(v_at_target, delta_raan_residual, i_target)
107
```

```
    total = (dv_hohmann_to_drift + dv_inc_to_drift + dv_hohmann_return + dv_inc_return +
    dv_residual)

    breakdown = {
        'hohmann_to_drift' : dv_hohmann_to_drift,
        'inc_to_drift' : dv_inc_to_drift,
        'hohmann_return' : dv_hohmann_return,
        'inc_return' : dv_inc_return,
        'raan_residual' : dv_residual,
        'total' : total
    }

    return total, breakdown

if __name__ == "__main__":
    # Test: Hohmann transfer from 680 km to 650 km
    r1 = R_E + 680e3
    r2 = R_E + 650e3
    dv1, dv2, total = hohmann_delta_v(r1, r2)
    tof = hohmann_transfer_time(r1, r2)

    print("Hohmann transfer: 680 km -> 650 km")
    print(f"Burn 1: {dv1:.2f} m/s")
    print(f"Burn 2: {dv2:.2f} m/s")
    print(f"Total:  {total:.2f} m/s")
    print(f"Transfer time: {tof/60:.1f} minutes")

    # Test: Direct plane change cost WITHOUT J2 drift
    v = np.sqrt(MU / r2)  # velocity at 650 km
    delta_raan = np.radians(3.0)  # 3 degree RAAN gap
    delta_i = np.radians(0.2)     # 0.2 degree inclination difference

    dv_raan = pure_raan_change_dv(v, delta_raan, np.radians(98.0))
    dv_inc = pure_inclination_dv(v, delta_i)
    dv_combined, u_star = combined_plane_change_dv(v, delta_i, delta_raan, np.radians(98.0))

    print(f"\nPlane change costs at 650 km (3° RAAN, 0.2° inc):")
    print(f"Pure RAAN change:       {dv_raan:.2f} m/s")
    print(f"Pure inclination change: {dv_inc:.2f} m/s")
    print(f"Sequential (sum):       {dv_raan + dv_inc:.2f} m/s")
    print(f"Combined (single burn): {dv_combined:.2f} m/s")
    print(f"Savings from combining: {(dv_raan + dv_inc) - dv_combined:.2f} m/s")
    print(f"Optimal burn location:  {np.degrees(u_star):.2f} deg")

    # Test: Full leg cost
    print(f"\nFull leg cost (680 km -> drift at 720 km -> target at 650 km):")
    total, bd = compute_leg_dv(
        a_mothership=R_E + 680e3,
        i_mothership=np.radians(98.0),
        a_drift=R_E + 720e3,
        i_drift=np.radians(97.9),
        a_target=R_E + 650e3,
        i_target=np.radians(97.8),
        delta_raan_residual=np.radians(0.5)  # 0.5 deg residual
    )

    print(f"Hohmann to drift:    {bd['hohmann_to_drift']:.2f} m/s")
```

```
164     print(f"Inc to drift:        {bd['inc_to_drift']:.2f} m/s")
165     print(f"Hohmann return:      {bd['hohmann_return']:.2f} m/s")
166     print(f"Inc return:          {bd['inc_return']:.2f} m/s")
167     print(f"RAAN residual:       {bd['raan_residual']:.2f} m/s")
168     print(f"TOTAL:               {bd['total']:.2f} m/s")
```

LISTING A.4. manoeuvre.py

A.1.5 transferleg.py

```
# Phase 2 - Transfer leg
# Finds the optimal drift orbit to reach a debris target

import numpy as np
from constants import MU, R_E, J2, DAY_TO_SEC
from orbitalstate import OrbitalState
from propagator import GVEPropagator
from manoeuvre import compute_leg_dv

"""
1) TransferLeg prices one possible mothership-to-target leg. For each target,
the class searches over drift altitude and drift duration, then solves for
the drift inclination that gives the required J2 RAAN precession rate.

2) The returned total_dv is the Delta-V charged to the mission budget. The
ranked_cost adds the optional time penalty, so the same optimiser can be used
for the fuel-minimum baseline and the lambda sweep.

3) The mothership is assumed to return to its parking-orbit shape after each
leg. The RAAN state is advanced by mission.py once the selected drift time is
committed.
"""

class TransferLeg:
    def __init__(self, mothership_state, debris_state, min_drift_alt=550e3, max_drift_alt=750e3,
                 min_drift_days=5, max_drift_days=300, time_penalty=0):
        self.mothership_state = mothership_state.copy()
        self.debris_state = debris_state.copy()
        self.min_drift_alt = min_drift_alt
        self.max_drift_alt = max_drift_alt
        self.min_drift_days = min_drift_days
        self.max_drift_days = max_drift_days
        self.time_penalty = time_penalty  # Penalty weight in m/s per drift day
        self.propagator = GVEPropagator()

    """
    1) For a chosen drift duration, compute the RAAN rate the mothership
    must have while drifting so that it arrives in the target plane at the
    same time as the target.
    """

    def _required_raan_rate(self, drift_time_s):
        """What RAAN precession rate does the drift orbit need to close the gap
        with the debris in the given time?"""
        raan_gap = self.debris_state.raan - self.mothership_state.raan

        debris_raan_rate = self.debris_state.raan_dot_j2

        required_rate = debris_raan_rate + raan_gap / drift_time_s

        return required_rate, raan_gap

    """
    1) Rearrange the J2 RAAN-rate equation to find the inclination that
    produces the required drift rate at the chosen altitude.
```

```
56
57      2) If the required cosine lies outside [-1, 1], that altitude-time pair
58      cannot close the RAAN gap and is rejected by returning None.
59      """
60
61      def _drift_inclination(self, a_drift, required_raan_rate):
62          """What inclination makes the drift orbit precess at the required rate?"""
63          p_drift = a_drift  # Assumes circular orbit (e ≈ 0), where p = a
64          n_drift = np.sqrt(MU / a_drift**3)
65
66          cos_i = required_raan_rate / (-1.5 * n_drift * J2 * (R_E / p_drift)**2)
67
68          if abs(cos_i) > 1.0:
69              return None  # No solution, geometry is impossible. No inclination at that altitude
        can achieve the required RAAN rate.
70
71          return np.arccos(cos_i)
72
73      """
74      1) For one candidate drift altitude and drift time, solve the required
75      drift inclination, reject infeasible geometries, and price the resulting
76      parking-drift-parking transfer.
77
78      2) The returned cost is the composite ranking cost. The Delta-V-only cost
79      remains inside the breakdown and is the value charged to the mission
80      budget by mission.py.
81      """
82
83      def compute_cost(self, drift_alt, drift_time_days):
84          """For a specific drift altitude and time, compute the total delta-V cost of the transfer
85          and the composite ranking cost."""
86          drift_time_s = drift_time_days * DAY_TO_SEC # Seconds
87          a_drift = R_E + drift_alt
88
89          required_rate, raan_gap = self._required_raan_rate(drift_time_s)
90
91          i_drift = self._drift_inclination(a_drift, required_rate)
92          if i_drift is None:
93              return np.inf, None  # No solution, geometry is impossible. Return infinite cost to
        exclude this option.
94
95          drift_inc_deg = np.degrees(i_drift)
96          if drift_inc_deg < 96.0 or drift_inc_deg > 100.0:
97              return np.inf, None  # Exclude drift orbits outside the filtered SSO inclination band.
98
99          mothership_raan_after = self.mothership_state.raan + required_rate * drift_time_s
100         debris_raan_after = self.debris_state.raan + self.debris_state.raan_dot_j2 * drift_time_s
101
102         delta_raan_residual = debris_raan_after - mothership_raan_after
103         delta_raan_residual = (delta_raan_residual + np.pi) % (2 * np.pi) - np.pi  # Wrap to [-pi,
        pi]
104
105         # Price out the full leg using manoeuvre.py
106         total_dv, breakdown = compute_leg_dv(
107             a_mothership=self.mothership_state.a,
108             i_mothership=self.mothership_state.i,
109             a_drift=a_drift,
```

```
            i_drift=i_drift,
            a_target=self.debris_state.a,
            i_target=self.debris_state.i,
            delta_raan_residual=delta_raan_residual
        )

        # Add metadata to the breakdown
        breakdown['drift_alt_km'] = drift_alt / 1e3
        breakdown['drift_inc_deg'] = np.degrees(i_drift)
        breakdown['drift_time_days'] = drift_time_days
        breakdown['raan_gap_initial_deg'] = np.degrees(raan_gap)
        breakdown['raan_residual_deg'] = np.degrees(delta_raan_residual)

        # Composite ranking cost used to compare candidate drift solutions for
        # this target. Only the delta-V component is charged to the mission
        # budget; the time-penalty term is a sequencing preference.
        total_cost = total_dv + self.time_penalty * drift_time_days

        return total_cost, breakdown

    def optimise(self, n_time_samples=30, n_alt_samples=20):
        """
        Search over drift altitudes and drift times to find the
        cheapest transfer. Returns a dict with the optimal parameters.
        """
        drift_times = np.linspace(self.min_drift_days, self.max_drift_days, n_time_samples)
        altitudes = np.linspace(self.min_drift_alt, self.max_drift_alt, n_alt_samples)

        best_cost = np.inf
        best_breakdown = None

        for t_days in drift_times:
            for alt in altitudes:
                cost, breakdown = self.compute_cost(alt, t_days)
                if cost < best_cost and breakdown is not None:
                    best_cost = cost
                    best_breakdown = breakdown

        if best_breakdown is None:
            return {'feasible': False}

        return {
            'feasible': True,
            'drift_alt_km': best_breakdown['drift_alt_km'],
            'drift_inc_deg': best_breakdown['drift_inc_deg'],
            'drift_time_days': best_breakdown['drift_time_days'],
            'total_dv': best_breakdown['total'],
            'ranked_cost': best_cost,
            'raan_gap_initial_deg': best_breakdown['raan_gap_initial_deg'],
            'raan_residual_deg': best_breakdown['raan_residual_deg'],
            'breakdown': best_breakdown
        }

if __name__ == "__main__":
    # Mothership at 680 km
    mothership = OrbitalState(
        a=R_E + 680e3, e=0.0001, i=np.radians(98.0),
```

```
        raan=np.radians(45.0), omega=0.0, u=0.0
    )

    # Debris at 650 km, 3 degrees ahead in RAAN
    debris = OrbitalState(
        a=R_E + 650e3, e=0.0001, i=np.radians(97.8),
        raan=np.radians(48.0), omega=0.0, u=0.0
    )

    print(f"Mothership: {mothership.altitude/1e3:.0f} km, "
          f"i={np.degrees(mothership.i):.2f} deg, "
          f"RAAN={np.degrees(mothership.raan):.2f} deg")
    print(f"Debris:     {debris.altitude/1e3:.0f} km, "
          f"i={np.degrees(debris.i):.2f} deg, "
          f"RAAN={np.degrees(debris.raan):.2f} deg")
    print(f"RAAN gap:   {np.degrees(debris.raan - mothership.raan):.2f} deg")

    # Fair direct baseline (no J2):
    # Match the J2-assisted leg's endpoint (mothership returns to its parking
    # orbit). The direct equivalent is therefore two combined plane changes at
    # the parking altitude: the first rotates the mothership onto the debris
    # plane so that mothership and debris are momentarily coplanar, the second
    # rotates it back to the original parking orientation. Altitude is not
    # changed in either manoeuvre, because the J2 leg does not transit to the
    # target's altitude either. Both manoeuvres are computed at parking-orbit
    # velocity, and both have the same angular magnitude, so the total is
    # simply twice the single-plane-change cost.
    from manoeuvre import combined_plane_change_dv
    delta_i = debris.i - mothership.i
    delta_raan = debris.raan - mothership.raan
    dv_one_plane_change, _ = combined_plane_change_dv(
        mothership.v_circular, delta_i, delta_raan, mothership.i
    )
    dv_direct = 2.0 * dv_one_plane_change

    print(f"\nDirect baseline (no J2, round-trip plane change): "
          f"{dv_direct:.2f} m/s")
    print(f"  Single plane change: {dv_one_plane_change:.2f} m/s "
          f"(forward and reverse at parking altitude)")

    # Optimise the J2-enhanced transfer
    print(f"\nOptimising J2-enhanced transfer...")
    leg = TransferLeg(mothership, debris)
    result = leg.optimise()

    if result['feasible']:
        print(f"\nOptimal solution:")
        print(f"  Drift altitude:    {result['drift_alt_km']:.1f} km")
        print(f"  Drift inclination: {result['drift_inc_deg']:.4f} deg")
        print(f"  Drift time:        {result['drift_time_days']:.1f} days")
        print(f"  Total delta-V:     {result['total_dv']:.2f} m/s")
        print(f"  RAAN residual:     {result['raan_residual_deg']:.4f} deg")

        savings = (1 - result['total_dv'] / dv_direct) * 100
        print(f"\n  Delta-V savings vs direct: {savings:.1f}%")

        bd = result['breakdown']
```

```
        print(f"\n  Breakdown:")
        print(f"    Hohmann to drift:     {bd['hohmann_to_drift']:.2f} m/s")
        print(f"    Inc to drift:         {bd['inc_to_drift']:.2f} m/s")
        print(f"    Hohmann return:       {bd['hohmann_return']:.2f} m/s")
        print(f"    Inc return:           {bd['inc_return']:.2f} m/s")
        print(f"    RAAN residual:        {bd['raan_residual']:.2f} m/s")
    else:
        print("  No feasible solution found")
```

Listing A.5. transferleg.py

A.1.6 mission.py

```
# Phase 3 + 4 - Mission Leg Optimisation
# Greedy approach - Always picks the cheapest next target under the active
# ranking metric. With time_penalty = 0 this reduces to pure Delta-V ranking.

import numpy as np
import pandas as pd
from constants import MU, R_E, J2, DEG_TO_RAD, DAY_TO_SEC
from orbitalstate import OrbitalState
from transferleg import TransferLeg

"""
1) Converts each row into an OrbitalState - the CSV stores angles in degrees and semi-major axis in km, while
OrbitalState expects radians and metres

2) Handle the fact that the CSV gives us MEAN_ANOMALY and ARG_OF_PERICENTER separately, but OrbitalState
uses argument of latitude u = omega + theta. For near-circular orbits (which debris in SSO mostly are),
mean anomaly ≈ true anomaly, so u ≈ omega + M is a reasonable approximation.

3) Return a list of OrbitalState objects alongside the debris names/IDs so we can track which is which
"""

def load_debris_catalogue(csv_path="plots/ch1_debris_environment/debris-catalogue-full.csv"):
    """Load debris targets from CSV and convert to OrbitalState objects."""
    df = pd.read_csv(csv_path)

    targets = []
    for _, row in df.iterrows():
        # CSV: km to m, deg to rad
        a = row['SEMIMAJOR_AXIS'] * 1e3
        e = row['ECCENTRICITY']
        i = row['INCLINATION'] * DEG_TO_RAD
        raan = row['RA_OF_ASC_NODE'] * DEG_TO_RAD
        omega = row['ARG_OF_PERICENTER'] * DEG_TO_RAD
        M = row['MEAN_ANOMALY'] * DEG_TO_RAD
        u = omega + M

        state = OrbitalState(a=a, e=e, i=i, raan=raan, omega=omega, u=u)

        targets.append({
            'state' : state,
            'norad_id' : row['NORAD_CAT_ID'],
            'name' : row['OBJECT_NAME']
        })

    return targets

"""
1) Set the mission constraints - Define where the mothership starts and how much fuel it has (expressed as
a total Delta-V budget in m/s). This function just bundles those together so the sequencing loop has
```

```
everything it needs.

2) Using a function as allows for experimentation with different starting conditions and fuel budgets
without changing the sequencing logic.
"""

def create_mission(dv_budget, a=R_E + 680e3, e=0.0001, i_deg=98.0,
                   raan_deg=45.0, omega=0.0, u=0.0):
    """Define the mothership's initial state and Delta-V budget."""
    mothership = OrbitalState(
        a=a, e=e, i=np.radians(i_deg), raan=np.radians(raan_deg),
        omega=omega, u=u
    )

    return {
        'mothership': mothership, # Current orbital state, which gets updated after each transfer leg
        'dv_budget': dv_budget, # Total fuel available in m/s
        'dv_spent': 0.0, # Cumulative Delta-V spent, starting at 0
        'visited' : [], # Array to accumulate targets removed in order
        'log' : [] # Array to record details of each transfer leg
    }

"""
1) The compute_all_costs function takes the mothership's current state and the list of unvisited targets,
then uses the TransferLeg class to price out every possible next transfer. It returns the results sorted
cheapest-first, so the greedy algorithm picks the top one.
"""

def compute_all_costs(mothership, targets, time_penalty=0):
    """Compute and rank the transfer cost from the current mothership state to each target."""
    costs = []

    for target in targets:
        leg = TransferLeg(mothership, target['state'], time_penalty=time_penalty)
        result = leg.optimise()

        if result['feasible']:
            costs.append({
                'target': target,
                'result': result
            })

    '''Rank by the same composite cost used inside TransferLeg. When
    time_penalty = 0 this is identical to sorting by pure delta-V. '''
    costs.sort(key=lambda x: x['result']['ranked_cost'])

    return costs

"""
2) The run_mission function ties everything together. It repeatedly finds the cheapest target, check if
the fuel budget allows for the transfer, visit the target, update the mothership, and repeat until
I'm out of fuel or targets.
```

```
"""

def run_mission(mission, targets, time_penalty=0):
    """Greedy sequencing loop: always picks the cheapest next target"""
    remaining = list(targets) # Creates a copy to preserve original
    step = 0

    print(f"Mission Start: {len(remaining)} targets, "
          f"{mission['dv_budget']:.1f} m/s Delta-V budget")

    while remaining:
        # Compute costs to all remaining targets
        ranked = compute_all_costs(mission['mothership'], remaining, time_penalty)

        if not ranked:
            print("No feasible targets remain.")
            break

        # Greedy pick: cheapest first under the active ranking metric
        best = ranked[0]
        dv_cost = best['result']['total_dv']
        dv_remaining = mission['dv_budget'] - mission['dv_spent']

        # Is it affordable?
        if dv_cost > dv_remaining:
            print(f"No more Delta-V remaining. Next target costs {dv_cost:.1f} m/s "
                  f"but only {dv_remaining:.1f} m/s remains.")
            break

        # Visit the target
        step += 1
        mission['dv_spent'] += dv_cost
        mission['visited'].append(best['target'])
        mission['log'].append(best['result'])

        '''Advance the catalogue by the committed drift duration. The
        mothership returns to the same parking-orbit shape after the
        leg, but that orbit has precessed under J2 while the leg was
        in progress; every unvisited target has also precessed at its
        own rate. Updating all of them preserves the true relative
        RAAN geometry when the next leg is priced.
        '''
        leg_dt_s = best['result']['drift_time_days'] * DAY_TO_SEC
        mission['mothership'].raan += (
            mission['mothership'].raan_dot_j2 * leg_dt_s
        )
        for tgt in remaining:
            tgt['state'].raan += tgt['state'].raan_dot_j2 * leg_dt_s

        # Remove from remaining
        remaining.remove(best['target'])

        # Print progress
        print(f"Step {step}: -> {best['target']['name']} "
              f"(NORAD {best['target']['norad_id']})")
        print(f"Delta-V: {dv_cost:.1f} m/s |"
              f"Drift: {best['result']['drift_time_days']:.0f} days |"
```

```
159                 f"Spent: {mission['dv_spent']:.1f} / "
160                 f"{mission['dv_budget']:.1f} m/s")
161
162     # Summary
163     print(f"\n{'='*50}")
164     print(f"Number of targets removed: {len(mission['visited'])}")
165     print(f"Delta-V spent: {mission['dv_spent']:.1f} m/s")
166     print(f"Budget remaining: "
167           f"{mission['dv_budget'] - mission['dv_spent']:.1f} m/s remaining")
168
169     return mission
170
171 if __name__ == "__main__":
172     # Load targets from CSV
173     targets = load_debris_catalogue()
174
175     # Create mission with 500 m/s Delta-V budget
176     mission = create_mission(dv_budget=500.0)
177
178     print(f"Mothership: {mission['mothership'].altitude/1e3:.0f} km, "
179           f"i={np.degrees(mission['mothership'].i):.2f} deg, "
180           f"RAAN={np.degrees(mission['mothership'].raan):.2f} deg\n")
181
182     # Run the greedy sequencer
183     result = run_mission(mission, targets, time_penalty=0)
```

LISTING A.6. `mission.py`

A.1.7 timepenalty.py

```
# Phase 4 - Time Penalty Sweep
# Runs the greedy sequencer of mission.py across a set of time-penalty
# coefficients lambda to expose the fuel / duration trade-off.

import io
import sys

from mission import load_debris_catalogue, create_mission, run_mission

"""
1) The lambda sweep is expressed as a list of coefficients rather than a
single value so that a single invocation of this module produces the full
summary table consumed by Section 4.2 of the dissertation. Each lambda in
the list is run independently with a freshly-loaded catalogue and a
freshly-created mission state, so no information leaks between runs.

2) lambda = 0 reproduces the fuel-minimum baseline of mission.py's default
run. Larger values bias the search toward shorter drifts at the expense of
fuel; values above roughly 1 m/s/day make drift time prohibitively
expensive and collapse most legs onto direct transfers.

3) The dv_budget is held fixed at the mission.py default of 500 m/s across
the sweep; sweeping both budget and lambda is out of scope for this module.
"""

LAMBDAS = [0.00, 0.05, 0.10, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00]
DV_BUDGET = 500.0

"""
1) sweep_time_penalty loops over the supplied lambda list and captures the
headline mission outcome for each run: number of targets removed, total
Delta-V spent, and total drift time summed across legs. Per-leg detail is
discarded, since the purpose of the sweep is the aggregate trade-off
rather than the sequence itself.

2) mission.py's run_mission prints progress to stdout on every leg. Those
prints are redirected through io.StringIO for the duration of each run to
keep the sweep's own output legible; the pattern matches that already used
in montecarlo.py.
"""

def sweep_time_penalty(lambdas=LAMBDAS, dv_budget=DV_BUDGET):
    """Run the greedy sequencer once per lambda and collect headline stats."""
    results = []

    for lam in lambdas:
        # Fresh catalogue and mission state each iteration avoids carry-over
        targets = load_debris_catalogue()
        mission = create_mission(dv_budget=dv_budget)

        # Silence run_mission's per-leg print output
        quiet = io.StringIO()
        saved_stdout = sys.stdout
        sys.stdout = quiet
        try:
```

```
            run_mission(mission, targets, time_penalty=lam)
        finally:
            sys.stdout = saved_stdout

        n_removed = len(mission['visited'])
        total_dv = mission['dv_spent']
        total_drift = sum(leg['drift_time_days'] for leg in mission['log'])

        results.append({
            'lambda': lam,
            'n_removed': n_removed,
            'total_dv': total_dv,
            'total_drift': total_drift
        })

        print(f"lambda={lam:>5.2f}: removed {n_removed:>3}, "
              f"DV={total_dv:>6.1f} m/s, drift={total_drift:>5.0f} days")

    return results

"""
1) print_summary_table formats the sweep results as a plain-text table
suitable for direct transcription into the LaTeX source. Columns match the
layout used in Section 4.2: time penalty lambda, targets removed, total
Delta-V spent, total drift time, and mission duration in years.
"""

def print_summary_table(results):
    """Print a formatted summary table of the sweep results."""
    print()
    print(f"{'lambda':>8}  {'removed':>8}  {'DV (m/s)':>10}  "
          f"{'drift (d)':>10}  {'duration (yr)':>14}")
    print("-" * 60)
    for r in results:
        years = r['total_drift'] / 365.25
        print(f"{r['lambda']:>8.2f}  {r['n_removed']:>8}  "
              f"{r['total_dv']:>10.1f}  {r['total_drift']:>10.0f}  "
              f"{years:>14.2f}")


if __name__ == "__main__":
    print(f"Time-penalty sweep: {len(LAMBDAS)} values, "
          f"dv_budget={DV_BUDGET} m/s\n")

    results = sweep_time_penalty()
    print_summary_table(results)
```

LISTING A.7. `timepenalty.py`

A.1.8 spacetrackdata.py

```
import pandas as pd
import numpy as np

"""
1) This script inspects the filtered Space-Track.org debris catalogue used by
the mission sequencer. It is not an optimiser, but it verifies the columns
and orbital-element ranges before the catalogue is passed into mission.py.

2) The output is a quick sanity check for the dataset used in the dissertation:
object count, available columns, sample rows, and basic statistics for the
altitude, inclination, RAAN, eccentricity, RCS size, and country fields.
"""

# Load the CSV
df = pd.read_csv("plots/ch1_debris_environment/debris-catalogue-full.csv")

print(f"Loaded {len(df)} debris objects\n")
print(f"Columns available: {list(df.columns)}\n")

# Show the key orbital elements for the first 10 objects
cols = ['NORAD_CAT_ID', 'OBJECT_NAME', 'SEMIMAJOR_AXIS',
        'ECCENTRICITY', 'INCLINATION', 'RA_OF_ASC_NODE',
        'ARG_OF_PERICENTER', 'MEAN_ANOMALY', 'PERIAPSIS',
        'APOAPSIS', 'RCS_SIZE', 'COUNTRY_CODE']

# Only show columns that actually exist in the CSV
available = [c for c in cols if c in df.columns]

print("First 10 objects:")
print(df[available].head(10).to_string(index=False))

# Basic statistics
print(f"\n{'='*50}")
print("Summary Statistics")
print(f"{'='*50}")

if 'PERIAPSIS' in df.columns:
    print(f"  Perigee altitude:  {df['PERIAPSIS'].min():.1f} - {df['PERIAPSIS'].max():.1f} km")

if 'APOAPSIS' in df.columns:
    print(f"  Apogee altitude:   {df['APOAPSIS'].min():.1f} - {df['APOAPSIS'].max():.1f} km")

if 'INCLINATION' in df.columns:
    print(f"  Inclination:       {df['INCLINATION'].min():.2f} - {df['INCLINATION'].max():.2f}°")

if 'RA_OF_ASC_NODE' in df.columns:
    print(f"  RAAN:              {df['RA_OF_ASC_NODE'].min():.2f} -
    {df['RA_OF_ASC_NODE'].max():.2f}°")

if 'ECCENTRICITY' in df.columns:
    print(f"  Eccentricity:      {df['ECCENTRICITY'].min():.6f} - {df['ECCENTRICITY'].max():.6f}")

if 'RCS_SIZE' in df.columns:
    print(f"\n  RCS size breakdown:")
    print(df['RCS_SIZE'].value_counts().to_string())
```

```
55
56 if 'COUNTRY_CODE' in df.columns:
57     print(f"\n  Top 5 countries of origin:")
58     print(df['COUNTRY_CODE'].value_counts().head(5).to_string())
```

LISTING A.8. spacetrackdata.py