Technical Report

Peder Ward

Cybernetic Engineer

Trondheim, Norway

November 12, 2024

Contents

Chapter 1

Introduction

This document provides an overview of various algorithms and their significance in computational problem-solving. It aims to set the foundation for understanding the design, analysis, and application of different algorithms across various domains.

Chapter 2

Algorithms

This chapter presents detailed analyses of different algorithms. It explores their mechanisms, use cases, and the theoretical concepts that underlie their effectiveness. Each algorithm is discussed with practical insights to highlight its strengths and potential limitations.

2.1 Optimization of Spacecraft Trajectory

2.1.1 Introduction

In this section, we present an algorithm designed to optimize a spacecraft's trajectory by maximizing fuel efficiency and utilizing gravitational assists. The algorithm calculates the optimal time and thrust parameters to minimize the cost function, which incorporates several penalties and efficiencies.

2.1.2 Variables

- m_{fuel}: Total mass of fuel onboard in kilograms (kg).
- T_{initial}: Initial thrust in Newtons (N).
- d_{closest}: Closest approach distance to the celestial body in meters (m).
- t: Time for which the thrust is applied in seconds (s).
- T: Thrust applied in Newtons (N).
- Δv: Change in velocity (delta-v) in meters per second (m/s).
- E_{grav}: Effect of gravitational assist.
- d_{traveled}: Distance traveled by the spacecraft in meters (m).
- m_{used}: Fuel mass used in kilograms (kg).
- P_{time}: Penalty for the elapsed time.
- $P_{\text{efficiency}}$: Penalty based on efficiency related to the closest approach.
- C: Total cost function value.

2.1.3 Formulas

$$\begin{aligned}
\Delta v &= \frac{T \cdot t}{m_{\text{fuel}}} \\
E_{\text{grav}} &= \begin{cases} \ln(d_{\text{closest}}) & \text{if } d_{\text{closest}} > 1 \\ 0 & \text{otherwise} \end{cases} \\
d_{\text{traveled}} &= \frac{\Delta v \cdot t \cdot E_{\text{grav}}}{2} \\
m_{\text{used}} &= m_{\text{fuel}} - \frac{T \cdot t}{\Delta v} \\
P_{\text{time}} &= 0.05 \cdot t^{1.2} \\
P_{\text{efficiency}} &= 0.01 \cdot (d_{\text{closest}} - 7 \times 10^6)^2 \\
C &= m_{\text{used}} + P_{\text{time}} + P_{\text{efficiency}}
\end{aligned}$$

2.1.4 Example

The following example illustrates the optimization of spacecraft trajectory, focusing on minimizing the total cost function C by varying the closest approach distances d_{closest}.

We start by defining the range of the closest approach distances that we will evaluate:

$$d_{\text{closest}} = [5 \times 10^6, 2 \times 10^7] \text{ meters}$$

For each value of d_{closest} within the specified range, the algorithm calculates the optimal cost C using the provided formulas and parameters:

$$\begin{aligned} m_{\text{fuel}} &= 500000 \text{ kg} \\ T_{\text{initial}} &= 20000 \text{ N} \end{aligned}$$

The calculations involve determining the effect of the closest approach distance on the total cost and identifying both the minimum and maximum cost points within this range.

The goal is to observe the behavior of the optimal costs as a function of d_{closest} when gravitational assists are employed for trajectory optimization. The results are visualized to aid in understanding and enhancing the performance of the spacecraft's trajectory design by selecting the optimal closest approach distance.

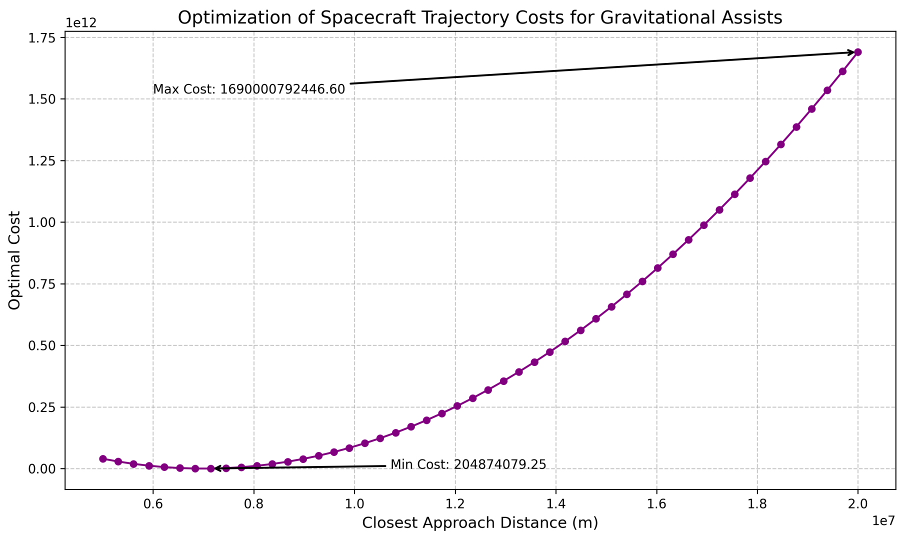

Figure 2.1: Example plot of Optimize Trajectory.

2.1.5 Code Listing

```python
import numpy as np
from scipy.optimize import minimize


# Function to optimize the spacecraft's trajectory based on fuel efficiency and gravitational assist
def optimize_trajectory(fuel_mass, initial_thrust, closest_approach):
    """
    Objective function for trajectory optimization using gravitational assist.

    Parameters:
    - fuel_mass (float): Total mass of fuel onboard in kg.
    - initial_thrust (float): Initial thrust in Newtons.
    - closest_approach (float): Closest approach distance to the celestial body in meters.

    Returns:
    - Optimal parameters (list): [time, thrust] values that minimize the cost function.
    - Cost (float): Minimized cost function value.
    """

    # Define the cost function for optimization
    def objective(params):
        time, thrust = params  # Unpack parameters: time and thrust

        # Calculate delta-v (velocity change) based on thrust and time
        delta_v = thrust * time / fuel_mass

        # Calculate the effect of gravitational assist based on closest approach distance
        gravitational_assist_effect = np.log(closest_approach) if closest_approach > 1 else 0

        # Simulate distance traveled using delta-v and gravitational effect
        distance_traveled = delta_v * time * gravitational_assist_effect / 2

        # Calculate the cost considering fuel usage, time penalty, and gravitational assist efficiency
        fuel_used = fuel_mass - (thrust * time) / delta_v
        time_penalty = 0.05 * time**1.2  # Small non-linear penalty for extended time
        efficiency_penalty = 0.01 * (closest_approach - 7e6) ** 2  # Optimal closest approach is around 7 million meters

        # Total cost combines all penalties and fuel usage
        cost = fuel_used + time_penalty + efficiency_penalty
        return cost

    # Initial guess for time and thrust
    initial_guess = [1e6, initial_thrust]

    # Define bounds for time and thrust to avoid unrealistic values
    bounds = [(1e5, 1e7), (5000, 3e5)]

    # Minimize the objective function to find the optimal time and thrust
    result = minimize(objective, initial_guess, bounds=bounds)
    return result.x, result.fun  # Return optimal parameters and the minimized cost
```

2.2 Predator-Prey System

2.2.1 Introduction

The predator-prey system is a set of differential equations known as the Lotka-Volterra equations, which describe the dynamics of biological systems in which two species interact: a prey and a predator.

2.2.2 Variables

- p_{prey}: Population of prey at time t.
- p_{predator}: Population of predators at time t.
- t: Time variable.
- α: Growth rate of the prey.
- β: Rate at which predators consume prey.
- δ: Growth rate of predators due to consumption of prey.
- γ: Natural death rate of predators.

2.2.3 Formulas

$$\begin{aligned}\frac{dp_{\text{prey}}}{dt} &= \alpha p_{\text{prey}} - \beta p_{\text{prey}} p_{\text{predator}} \\ \frac{dp_{\text{predator}}}{dt} &= \delta p_{\text{prey}} p_{\text{predator}} - \gamma p_{\text{predator}}\end{aligned}$$

2.2.4 Example

Consider a predator-prey system with the following parameters and initial conditions:

$$\begin{aligned}\alpha &= 0.1 \quad \text{(prey growth rate)} \\ \beta &= 0.02 \quad \text{(rate at which predators consume prey)} \\ \delta &= 0.01 \quad \text{(rate at which prey consumption converts into predator growth)} \\ \gamma &= 0.1 \quad \text{(predator death rate)} \\ p_{\text{prey, initial}} &= 40 \quad \text{(initial population of prey)} \\ p_{\text{predator, initial}} &= 9 \quad \text{(initial population of predators)}\end{aligned}$$

The time variable t represents the temporal progression of the system:

$$t = [0, 200] \quad \text{(time points for simulation at intervals)}$$

These equations describe how the populations of prey and predators change over time based on the initial conditions and parameters defined. The solution of these differential equations provides insights into the dynamics of the predator and prey populations within the specified time frame.

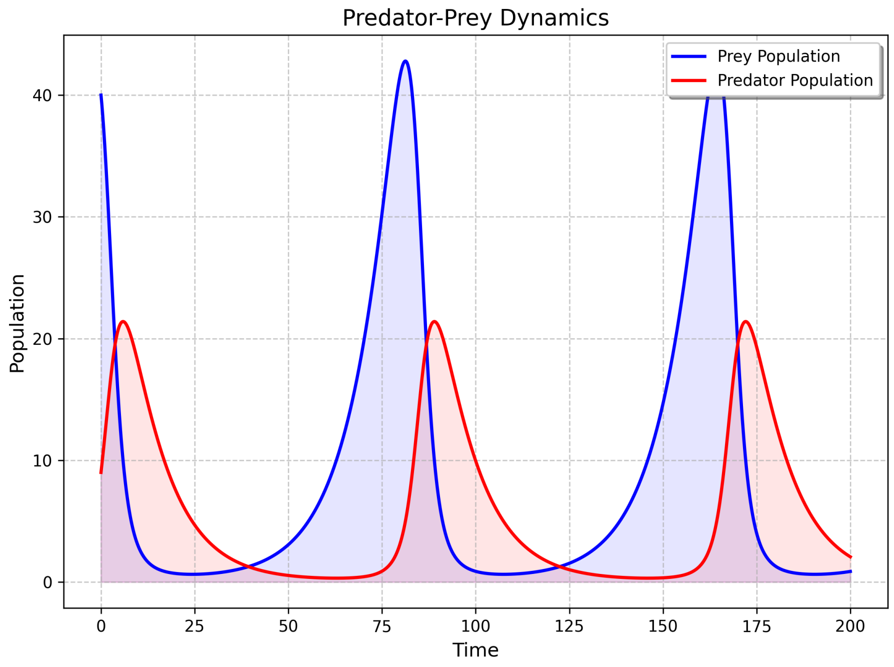

Figure 2.2: Example plot of Predator Prey System.

2.2.5 Code Listing

```python
# Define the predator-prey system as a system of differential equations
def predator_prey_system(populations, t, alpha, beta, delta, gamma):
    """
    Calculate the rate of change for prey and predator populations based on Lotka-Volterra equations.

    Parameters:
    - populations (list): A list with two elements: [prey, predator] populations at time t.
    - t (float): Time variable, required for odeint even though it's unused here.
    - alpha (float): Prey growth rate.
    - beta (float): Rate at which predators consume prey.
    - delta (float): Predator growth rate from consuming prey.
    - gamma (float): Natural predator death rate.

    Returns:
    - list: Contains the rate of change [dprey_dt, dpredator_dt] for prey and predator populations.
    """
    prey, predator = (
        populations  # Unpack the populations array into prey and predator variables
    )
    dprey_dt = alpha * prey - beta * prey * predator  # Rate of change for prey
    dpredator_dt = (
        delta * prey * predator - gamma * predator
    )  # Rate of change for predator
    return [dprey_dt, dpredator_dt]
```

2.3 Gravitational Assist Algorithm

2.3.1 Introduction

This section outlines an algorithm for calculating the final velocity of a spacecraft after a gravitational assist with a planet. The calculation considers the initial velocities, masses, and distances involved.

2.3.2 Variables

- v_{sc}: Initial velocity of the spacecraft in m/s.
- v_{p}: Velocity of the planet in m/s.
- m_{p}: Mass of the planet in kg.
- d_{closest}: Distance of closest approach in meters.
- $v_{\text{rel, init}}$: Relative initial velocity between spacecraft and planet.
- v_{∞}: Velocity at infinity (far from the planet) after gravitational interaction.
- v_{final}: Spacecraft's final velocity after the gravitational assist.
- G: Gravitational constant, $6.67430 \times 10^{-11}\,\text{m}^3\,\text{kg}^{-1}\,\text{s}^{-2}$.

2.3.3 Formulas

$$\begin{aligned} v_{\text{rel, init}} &= |v_{\text{sc}} - v_{\text{p}}| \\ v_{\infty} &= \sqrt{v_{\text{rel, init}}^2 + \frac{2 \cdot G \cdot m_{\text{p}}}{d_{\text{closest}}}} \\ v_{\text{final}} &= \sqrt{v_{\text{sc}}^2 + 2 \cdot (v_{\infty}^2 - v_{\text{sc}} \cdot v_{\infty} \cdot \cos(\pi))} \end{aligned}$$

2.3.4 Example

The following example demonstrates the calculation of the spacecraft's final velocity after a gravitational assist maneuver using specific initial conditions.

Given :

$v_{\text{sc}} = 1.2 \times 10^4\,\text{m/s}$,

$v_{\text{p}} = 2.4 \times 10^4\,\text{m/s}$,

$m_{\text{p}} = 5.972 \times 10^{24}\,\text{kg}$,

$d_{\text{closest}} = 6.7 \times 10^6\,\text{m}$,

First, find the relative initial velocity:

$v_{\text{rel, init}} = |v_{\text{sc}} - v_{\text{p}}| = |1.2 \times 10^4 - 2.4 \times 10^4|\,\text{m/s}$.

Next, calculate the velocity at infinity:

$v_\infty = \sqrt{v_{\text{rel, init}}^2 + \dfrac{2 \cdot G \cdot m_{\text{p}}}{d_{\text{closest}}}}$,

where $G = 6.67430 \times 10^{-11}\,\text{m}^3\,\text{kg}^{-1}\,\text{s}^{-2}$.

Finally, compute the spacecraft's final velocity:

$v_{\text{final}} = \sqrt{v_{\text{sc}}^2 + 2 \cdot (v_\infty^2 - v_{\text{sc}} \cdot v_\infty \cdot \cos(\pi))}$.

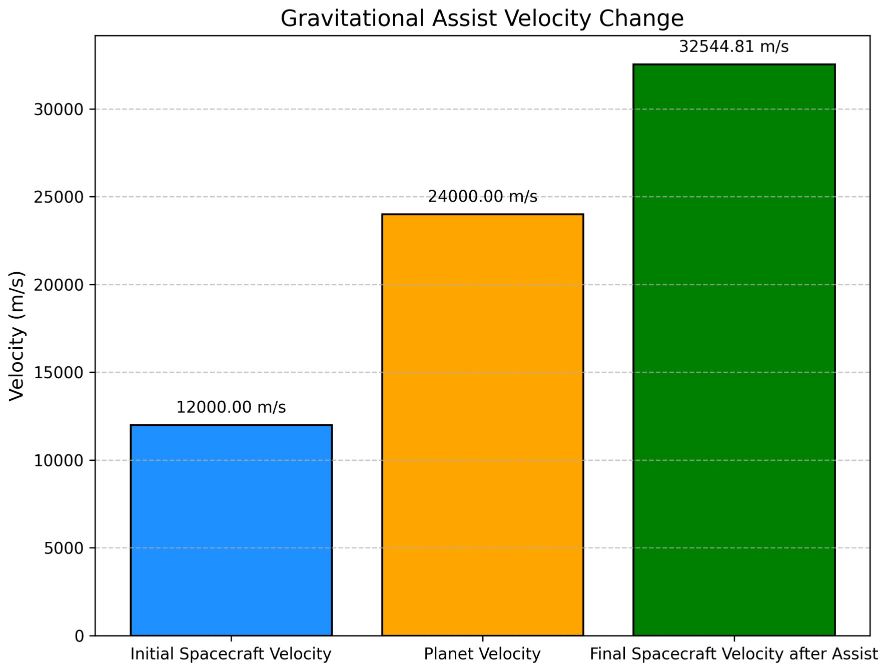

Figure 2.3: Example plot of Gravitational Assist.

2.3.5 Code Listing

```python
import numpy as np

# Function to calculate the final velocity of a spacecraft after gravitational assist
def gravitational_assist(spacecraft_velocity, planet_velocity, planet_mass, closest_approach_distance):
    """
    Calculate the final velocity of a spacecraft after a gravitational assist with a planet.

    Parameters:
    - spacecraft_velocity (float): Initial velocity of the spacecraft in m/s.
    - planet_velocity (float): Velocity of the planet in m/s.
    - planet_mass (float): Mass of the planet in kg.
    - closest_approach_distance (float): Distance of closest approach in meters.

    Returns:
    - v_final (float): The spacecraft's final velocity after the gravitational assist.
    """
    G = 6.67430e-11  # Gravitational constant in m^3 kg^-1 s^-2

    # Relative initial velocity between the spacecraft and the planet
    v_rel_initial = abs(spacecraft_velocity - planet_velocity)

    # Calculate velocity at infinity (far from the planet) after gravitational interaction
    v_inf = np.sqrt(v_rel_initial**2 + 2 * G * planet_mass / closest_approach_distance)

    # Calculate final velocity considering energy changes from the approach and assist
    v_final = np.sqrt(spacecraft_velocity**2 + 2 * (v_inf**2 - spacecraft_velocity * v_inf * np.cos(np.pi)))

    return v_final
```

2.4 Hohmann Transfer Orbit Calculation

2.4.1 Introduction

The Hohmann transfer algorithm calculates the velocity changes (delta-v) required to transfer a spacecraft from Earth's orbit to Mars's orbit. This is achieved through the use of gravitational interactions, specifically the Sun's gravitational influence.

2.4.2 Variables

- r_{earth}: Radius of Earth's orbit around the Sun in meters.
- r_{mars}: Radius of Mars's orbit around the Sun in meters.
- μ_{sun}: Gravitational constant for the Sun in m^3/s^2.
- v_{earth}: Velocity of Earth in its orbit.
- v_{mars}: Velocity of Mars in its orbit.
- a_{transfer}: Semi-major axis of the Hohmann transfer orbit.
- $v_{\text{transfer\_earth}}$: Velocity required at Earth for the transfer orbit.
- $v_{\text{transfer\_mars}}$: Velocity required at Mars for the transfer orbit.
- Δv_{earth}: Delta-v required to escape Earth's orbit and enter the transfer orbit.
- Δv_{mars}: Delta-v required to enter Mars's orbit from the transfer orbit.
- Δv_{total}: Total delta-v for the entire transfer.

2.4.3 Formulas

$$\mu_{\text{sun}} = 1.32712440018 \times 10^{20}\,\text{m}^3/\text{s}^2$$
$$v_{\text{earth}} = \sqrt{\frac{\mu_{\text{sun}}}{r_{\text{earth}}}}$$
$$v_{\text{mars}} = \sqrt{\frac{\mu_{\text{sun}}}{r_{\text{mars}}}}$$
$$a_{\text{transfer}} = \frac{r_{\text{earth}} + r_{\text{mars}}}{2}$$
$$v_{\text{transfer\_earth}} = \sqrt{\frac{2\mu_{\text{sun}}}{r_{\text{earth}}} - \frac{\mu_{\text{sun}}}{a_{\text{transfer}}}}$$
$$v_{\text{transfer\_mars}} = \sqrt{\frac{2\mu_{\text{sun}}}{r_{\text{mars}}} - \frac{\mu_{\text{sun}}}{a_{\text{transfer}}}}$$
$$\Delta v_{\text{earth}} = v_{\text{transfer\_earth}} - v_{\text{earth}}$$
$$\Delta v_{\text{mars}} = v_{\text{mars}} - v_{\text{transfer\_mars}}$$
$$\Delta v_{\text{total}} = |\Delta v_{\text{earth}}| + |\Delta v_{\text{mars}}|$$

2.4.4 Example

To demonstrate the Hohmann transfer orbit calculation, consider the following scenario:

$$r_{\text{earth}} = 1.496 \times 10^{11}\,\text{m}$$
$$r_{\text{mars}} = 2.279 \times 10^{11}\,\text{m}$$
$$\Delta v_{\text{earth}}, \Delta v_{\text{mars}}, \Delta v_{\text{total}} = \text{calculate\_hohmann\_transfer}(r_{\text{earth}}, r_{\text{mars}})$$

To visualize the orbits involved in the transfer, consider the following coordinates:

1. Earth's orbit around the Sun is circular, with coordinates (x, y) given by:

$$x_{\text{earth}} = r_{\text{earth}} \cdot \cos(\theta)$$
$$y_{\text{earth}} = r_{\text{earth}} \cdot \sin(\theta)$$

2. Similarly, Mars's orbit coordinates are given by:

$$x_{\text{mars}} = r_{\text{mars}} \cdot \cos(\theta)$$
$$y_{\text{mars}} = r_{\text{mars}} \cdot \sin(\theta)$$

For the transfer orbit:

- The semi-major axis a_{transfer} and eccentricity e_{transfer} are calculated as follows:

$$a_{\text{transfer}} = \frac{r_{\text{earth}} + r_{\text{mars}}}{2}$$
$$e_{\text{transfer}} = \frac{r_{\text{mars}} - r_{\text{earth}}}{r_{\text{earth}} + r_{\text{mars}}}$$

- The coordinates (x, y) for the Hohmann transfer orbit are:

$$x_{\text{transfer}} = a_{\text{transfer}} \cdot (\cos(\theta) - e_{\text{transfer}})$$
$$y_{\text{transfer}} = a_{\text{transfer}} \cdot \sqrt{1 - e_{\text{transfer}}^2} \cdot \sin(\theta)$$

The total velocity change Δv_{total} required for the transfer orbit is given by:

$$\Delta v_{\text{total}} = \Delta v_{\text{earth}} + \Delta v_{\text{mars}}$$

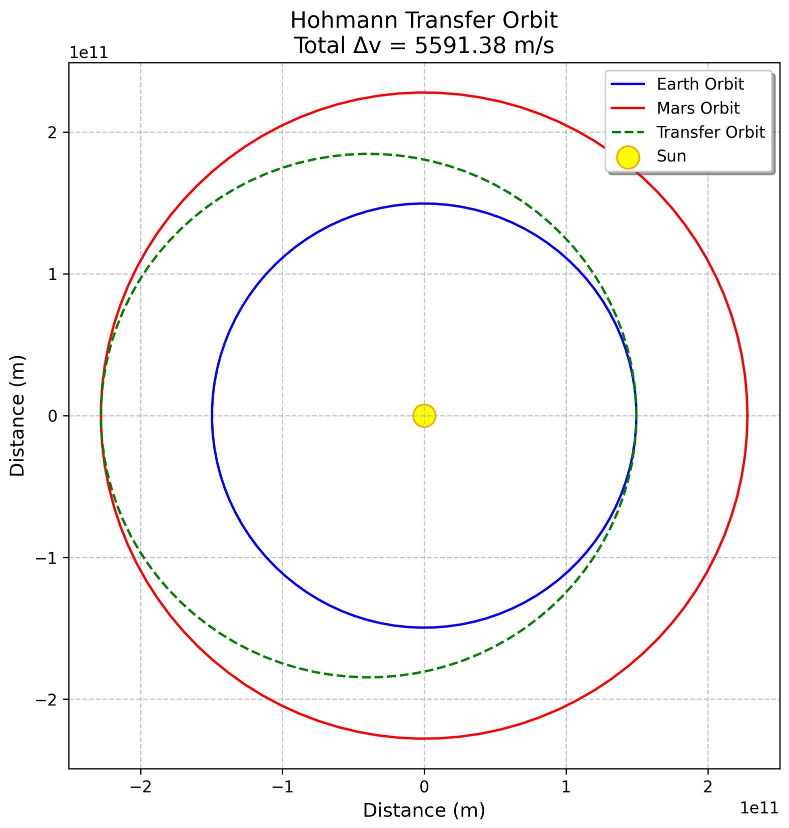

Figure 2.4: Example plot of Calculate Hohmann Transfer.

2.4.5 Code Listing

```python
import numpy as np


def calculate_hohmann_transfer(earth_orbit_radius, mars_orbit_radius):
    """
    Calculate the delta-v (velocity change) required for a Hohmann transfer orbit from Earth to Mars.

    Parameters:
    - earth_orbit_radius (float): Radius of Earth's orbit around the Sun in meters.
    - mars_orbit_radius (float): Radius of Mars's orbit around the Sun in meters.

    Returns:
    - delta_v_earth (float): Delta-v required to escape Earth's orbit and enter the transfer orbit.
    - delta_v_mars (float): Delta-v required to enter Mars's orbit from the transfer orbit.
    - total_delta_v (float): Total delta-v for the entire transfer.
    """

    # Gravitational constant for the Sun in m^3/s^2
    mu_sun = 1.32712440018e20

    # Calculate orbital velocities of Earth and Mars using circular orbit approximation
    v_earth = np.sqrt(mu_sun / earth_orbit_radius)  # Velocity of Earth in its orbit
    v_mars = np.sqrt(mu_sun / mars_orbit_radius)  # Velocity of Mars in its orbit

    # Calculate the semi-major axis of the Hohmann transfer orbit (average distance between Earth and Mars)
    transfer_orbit_semi_major_axis = (earth_orbit_radius + mars_orbit_radius) / 2

    # Calculate the velocity required at Earth's position for the transfer orbit
    v_transfer_at_earth = np.sqrt(2 * mu_sun / earth_orbit_radius - mu_sun / transfer_orbit_semi_major_axis)

    # Calculate the velocity required at Mars's position for the transfer orbit
    v_transfer_at_mars = np.sqrt(2 * mu_sun / mars_orbit_radius - mu_sun / transfer_orbit_semi_major_axis)

    # Delta-v at Earth: difference between the transfer orbit velocity and Earth's orbital velocity
    delta_v_earth = v_transfer_at_earth - v_earth

    # Delta-v at Mars: difference between Mars's orbital velocity and the transfer orbit velocity at Mars
    delta_v_mars = v_mars - v_transfer_at_mars

    # Total delta-v required for the Hohmann transfer (sum of absolute delta-v values at Earth and Mars)
    total_delta_v = abs(delta_v_earth) + abs(delta_v_mars)

    return delta_v_earth, delta_v_mars, total_delta_v
```