SALTO: AN EXPERT-INFORMED GLOBAL TRAJECTORY DESIGN AND OPTIMISATION TOOLKIT

Waldemar Martens [(1)], Johannes Schoenmaekers [(1)], Gábor Varga [(2)], Colin Baumgard [(3)], Olga Ramírez Torralba [(4)], Pablo Muñoz [(1)], Moritz von Looz [(1)]

[(1)] ESA/ESOC, Robert-Bosch-Str. 5, 64293 Darmstadt, Germany, waldemar.martens@esa.int
[(2)]IMS Space Consultancy, Am Steinern Kreuz 26, 64297 Darmstadt
[(3)] CS Group - Germany GmbH, Berliner Allee 65 D-64295 Darmstadt, Germany
[(4)] Solenix Engineering GmbH, Spreestr. 3, 64295 Darmstadt, Germany

ABSTRACT

A novel trajectory design algorithm suitable for interplanetary, lunar and libration point missions with high and low thrust is described. Unlike techniques that aim at generating initial guesses in a problem-agnostic way, the method proposed here is fully tailored to the trajectory design problem and exploits expert insight in orbital motion.
The Python implementation of this algorithm, called SALTO (Search Algorithm using Linked Trajectory Optimisation), is based on two main ideas.
Firstly, the initial guess is only required to solve the problem from an energetic perspective and does not need to solve the phasing problem yet. This allows for large state and time gaps along the trajectory, which are systematically closed in a second step that exploits time shifts by full orbital periods and/or body rotations.
Secondly, the algorithm is not expected to autonomously find all local minima in the cost function. Instead, the user defines "tags," or category variables, that guide SALTO's search for solutions by indicating the expected structure of the search space.
SALTO is developed at ESOC as part of the MIDAS package and is Community Open Source (ESA Community License).

1 INTRODUCTION

Trajectory design for interplanetary missions is a challenging problem for which a wide range of algorithms have been proposed and used in the past. These algorithms range from completely problem-agnostic solvers, such as evolutionary algorithms [1], to methods that incorporate some degree of domain knowledge, like branch and pruning [2][3][4][5]. In this paper, a novel algorithm is presented that is at the other end of this spectrum in the sense that it is expert-informed and fully tailored to the trajectory design problem. This algorithm was first developed at ESOC in an in-house prototype, called SWING. It was designed for interplanetary missions with high and low-thrust and was successfully applied for reproducing the trajectories of JUICE, SOLO, BepiColombo and Rosetta. Later, it was realised that the method used in SWING is generalisable to a wider class of missions, including lunar low-energy transfers, libration point missions and any combination of these.

This triggered the development of SALTO (Search Algorithm using Linked Trajectory Optimisation) as a generic, user-extensible tool. SALTO is a module in the mission analysis Python package, MIDAS, and is based on the ESOC flight dynamics infrastructure, GODOT [6][7]. It supports the direct generation of multiple-shooting trajectories in the GODOT format from initial guesses. These can be used in local optimisation and navigation analysis and therefore seamlessly integrates into the end-to-end mission analysis workflow at ESOC. SALTO is freely available to the European industry and academia as part of the MIDAS package under the ESA Community License (Community Open Source). It is accessible through the space-codev.org platform.

The general concepts and ideas behind the SALTO algorithm will be introduced in section 2 using the JUICE mission as an example. A more detailed look into the software implementation is then provided in section 3. Potential use cases and ways for users to extend SALTO are discussed in section 4, followed by conclusions and future work in section 5.

2 GENERAL CONCEPTS

The general working principles in SALTO are explained in the following using the JUICE interplanetary transfer [8] as an example.

2.1 Non-phased Initial Guesses

To understand the SALTO algorithm, one can start by observing that optimal transfers for a given planet sequence look close to identical in the Tisserand[1] graph [9]. As an illustration for the launch-Earth-Venus-Earth-Earth-Jupiter (launch-e1-v1-e2-e3-joi) flyby sequence, the JUICE baseline transfer is depicted in Figure 1.

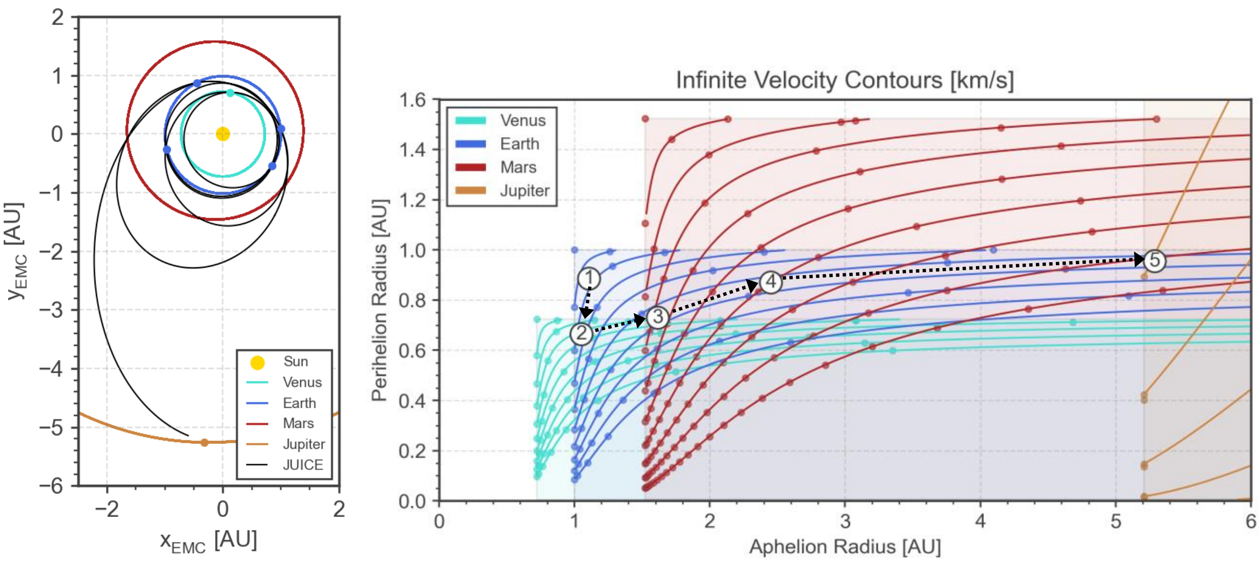


Figure 1: The JUICE baseline interplanetary transfer trajectory (left) and the corresponding path in the Tisserand graph (right). The infinite velocity contours in the Tisserand graph are separated by 2 km/s. The marker spacing on the contours correspond to flybys of 300 km altitude. The planet sequence is: launch, Earth (①→②), Venus (②→③), Earth (③→④), Earth (④→⑤), Jupiter.

[1] The Tisserand graph depicts contour lines of constant infinite velocity relative to the different planets and assumes a planar, circular orbit model for the planets. It is a useful and widespread tool for the design of multi-gravity-assist trajectories. To make this paper as self-contained as possible, a primer on Tisserand graphs is included in the Annex (section 6).

Even though the depicted transfer corresponds to specific flyby dates, any other transfer solution with the same planet sequence will look almost indistinguishable in the Tisserand graph. This is because the Tisserand graph shows the solution only from the energetic point of view and doesn't hold any information about the planet phasing and flyby dates. Therefore, it is very easy to analytically construct transfers in the Tisserand graph when the planet sequence is given.

But how useful are such initial guesses for the construction of "phased" solutions? Note that the path in the Tisserand graph fixes the semi-major axes and eccentricities of all intermediate orbits during the transfer. To fully define the trajectory in a planar non-phased orbit model, the only additional parameter that needs to be adjusted for the intermediate orbits is the argument of perihelion. This can easily be done by solving a sequence of one-dimensional root-finding problems for the intermediate orbits starting from the Earth-Jupiter arc and proceeding backwards towards launch. This is a numerically fast and robust procedure, and the resulting trajectory is continuous in space, i.e., it is almost indistinguishable from that in Figure 1 (left). However, it still assumes that the planets are at the right position when the spacecraft arrives to do the flyby. A way of looking at such a solution is to say that it is continuous in space, but not in time. That can be seen by plotting the Sun distance evolution as shown in Figure 2.

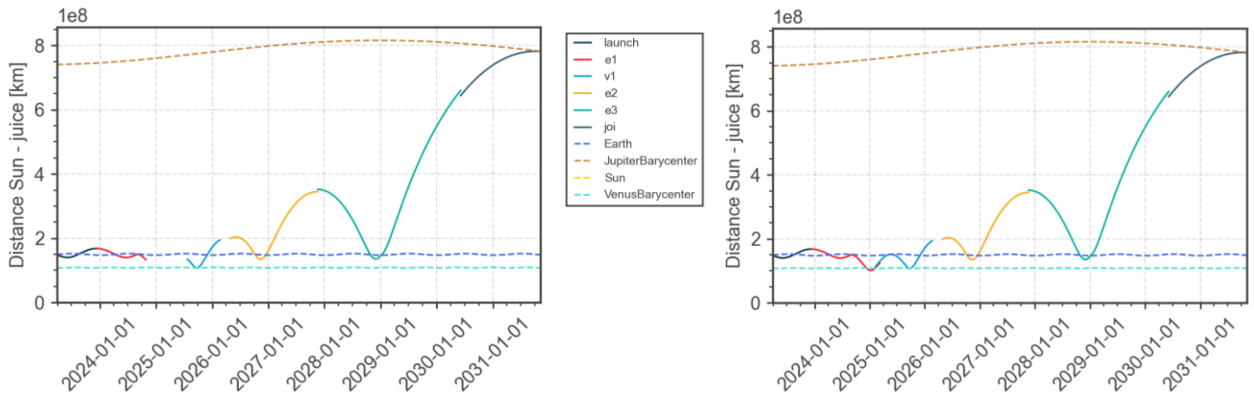


Figure 2: Sun distance evolution of two Tisserand initial guesses. These have zero (left) and one (right) full spacecraft revolutions in the e1 to v1 arc.

Here, we have fixed the Jupiter arrival date to result in a Hohmann transfer from Earth to Jupiter and adjusted the other flyby dates to match the closest epoch where the planet is at the correct position for the flyby. Moreover, in this step we have also moved from a planar, circular planet model to the full ephemeris model, which is apparent from the varying Sun distance evolution of the planets. Also, the plot already uses a numerical integration of the multiple-shooting trajectory in a dynamical model where both the flyby body and the Sun gravities are considered. Note that the choice of epochs done in this step is not unique! More precisely, any flyby epoch can be moved forward or backward in time by integer numbers of planet orbital periods. Any candidate solution resulting from such an operation will look identical in the trajectory plot (Figure 1), but will exhibit different time gaps in the radius plot (Figure 2). This property can be used to our advantage to construct feasible, time-continuous solutions, as will be described in the following.

The difference between the two plots in Figure 2 is the number of full spacecraft revolutions (these are examples of "niche tags" in SALTO jargon) in the e1 to v1 arc: zero revolutions in the left plot and one full revolution in the right plot. The initial guess fixes the true longitude of the flybys, but not the epochs: there is a remaining ambiguity modulo full body orbital periods. Any shift of a flyby epoch by a full body orbital period leads to another valid initial guess with different time gaps. A shift of the launch date and e1 date by one full Earth orbital period on the left figure would result in a decrease of the time gap in the e1 to v1 arc and thus a better initial guess. The initial guess in the right plot already has very small time gaps. The procedure of how to fully close these time gaps will be described in section 2.3.

2.2 Niche Tags and Diversity of Solutions

Before delving into how to solve the phasing problem, it shall be noted that non-phased initial guesses are created for every expected niche (local optimum) in the solution space. This is to generate a diverse pool of solutions, rather than just one global optimum. The knowledge of where to find these niches is drawn purely from experience and does not rely on any automatic algorithms. E.g., in the context of multi-flyby trajectories, every arc can have an inward or outward departure and arrival asymptote at the planets, as shown in Figure 3. Each combination of departure and arrival directions at all the planets will lead to a qualitatively different transfer, both in terms of geometry and timing, thus a different local optimum - or niche.
The departure direction and arrival direction are examples of “niche tags” in SALTO. Another example is the number of full spacecraft revolutions during a transfer arc. Niche tags are used to explicitly enumerate the different niches in the solution space where a solution is looked for.

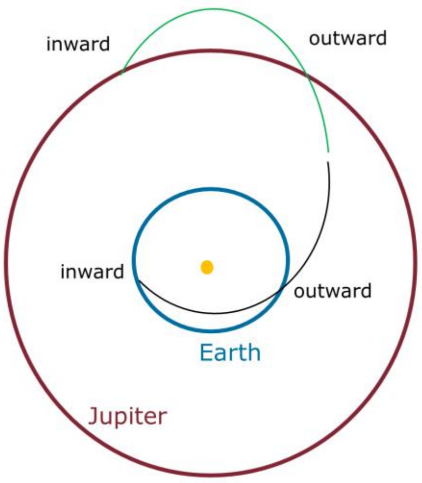


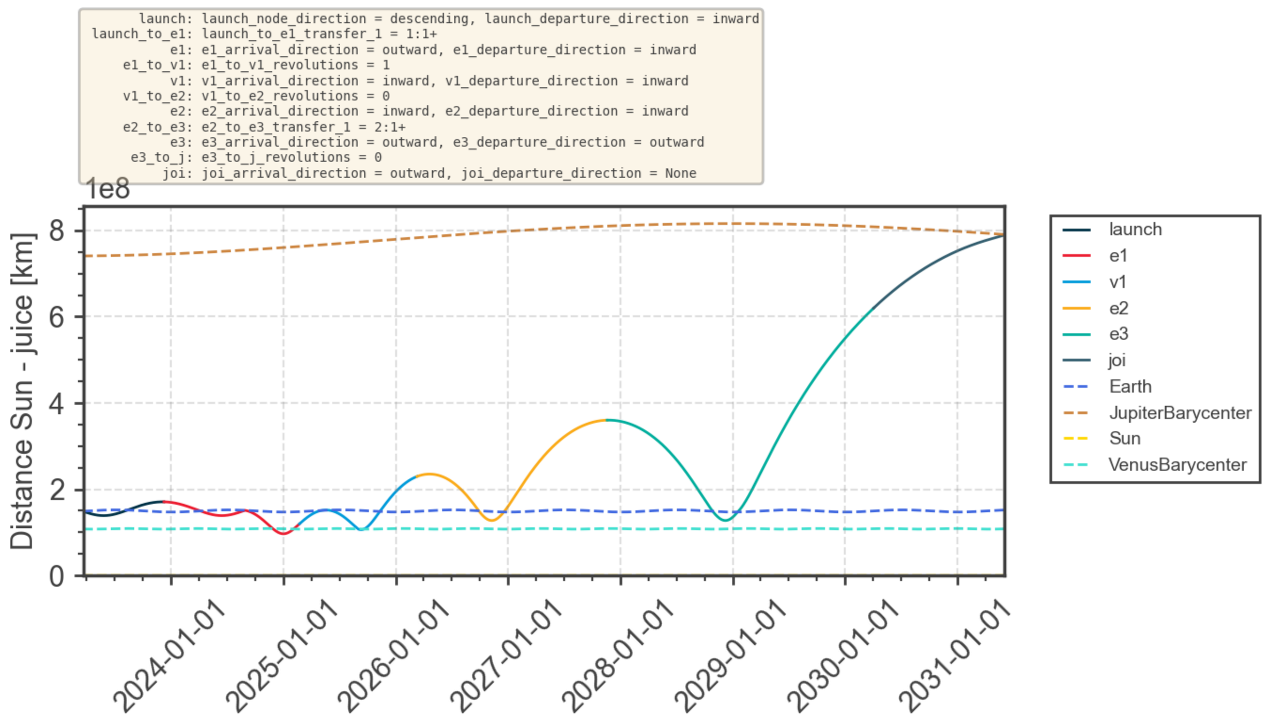
Figure 3: Departure and arrival direction tags for the Earth-to-Jupiter transfer.

2.3 Solving the Phasing Problem

In section 2.1 it was noted that in the candidate solutions constructed so far, any flyby epoch can be moved forward or backward in time by integer numbers of planet orbital periods. This operation does not affect the continuity of the trajectory in space, but only alters the time gaps between the flybys. To construct a fully continuous trajectory, an algorithm consisting of flyby epoch time shifts and local optimisation runs is applied to the candidate solution. It starts at the Jupiter arrival epoch and moves backwards through the trajectory towards launch. The algorithm can be summarised as follows:

1. For the current planet, select the number of orbital periods to move the flyby epoch by. This can be any integer (positive or negative) number. The number is called a “time shift tag”.
2. If the resulting time gap in the previous transfer arc is lower than a user-given threshold, run a local optimisation with a matching constraint to close the time gap. If the optimisation fails to converge, discard the previously selected time shift tag combination.
3. If the optimisation is successful, move on to the next planet in the sequence and proceed with step 1.

At the end of this procedure, we end up with a continuous (in both space and time) trajectory for every feasible niche tag combination. Figure 4 shows the Sun distance evolution of such a solution. It corresponds to the flown JUICE trajectory (with the exception that it uses a standard Earth flyby at e1 whereas JUICE uses a combined Lunar-Earth gravity assist) and results from the initial guess from Figure 2 (right). The initial guess from Figure 2 (left) was discarded by the algorithm, because the time gap in the e1 to v1 arc could not be closed.
Note that a comprehensive search for trajectories in SALTO encompasses a consideration of all possible niche tag combinations. In the JUICE example, there are 10 departure/arrival direction tags where each can take two values ("inward" or "outward"). Even neglecting the niche tags coming from the resonance ratios and full spacecraft revolutions, that would result in 2^{10}=1024 niche tag combinations! An efficient implementation of the described algorithm therefore needs to discard unfeasible tag combinations as early as possible in the process to remain practically useful.



```
      launch: launch_node_direction = descending, launch_departure_direction = inward
launch_to_e1: launch_to_e1_transfer_1 = 1:1+
          e1: e1_arrival_direction = outward, e1_departure_direction = inward
    e1_to_v1: e1_to_v1_revolutions = 1
          v1: v1_arrival_direction = inward, v1_departure_direction = inward
    v1_to_e2: v1_to_e2_revolutions = 0
          e2: e2_arrival_direction = inward, e2_departure_direction = inward
    e2_to_e3: e2_to_e3_transfer_1 = 2:1+
          e3: e3_arrival_direction = outward, e3_departure_direction = outward
     e3_to_j: e3_to_j_revolutions = 0
         joi: joi_arrival_direction = outward, joi_departure_direction = None
```

Figure 4: Sun distance evolution of the fully converged JUICE baseline trajectory. All time gaps have been closed by the algorithm described in the main text. The yellow box indicates all niche tag values that resulted in this trajectory. These include the departure/arrival directions at the flybys, the full spacecraft revolutions in the transfer arcs and the (pseudo-)resonance ratios of the Earth-to-Earth arcs.

Note that due to the nature of this algorithm, it is not limited to a particular dynamical model or a patched conics approximation. In fact, it is very suitable for a gradual increase of fidelity, starting with simple and fast models and moving on towards fully realistic models. Moreover, the concepts exploited in SALTO are not limited to multi-flyby trajectories, but can be extended to any scenario where:

1. There exists a simple and fast method for initial guess generation for one trajectory arc (e.g. the Tisserand graph). This initial guess does not need to solve the phasing problem yet.
2. There is a periodicity in the system due to planet revolutions around the central body or planet rotations around its axis.

Such problems include lunar (low-energy) missions, libration point missions, moon tours and any combination of the different types. The SALTO software package is set up in such a way as to be agnostic about the underlying transfer problem, which allows for a high level of flexibility and future extensibility.

3 SOFTWARE PACKAGE DESCRIPTION

This section describes the structure of the SALTO software module inside MIDAS (`midas.salto`) and explains the relation between the different classes to the algorithm presented in the previous section. More details on the software and tutorials can be found in the MIDAS documentation [**10**].
The objective is to make SALTO a top-level framework that allows the user to easily concatenate mission phases that have been designed with low-level algorithms from the `midas.design` module and to convert them to a multiple-shooting trajectory configuration for further processing. As such, SALTO is based on an abstraction of the algorithms described in the previous section.

3.1 Segments, Missions and Individuals

In SALTO, (candidate) solutions for a mission part are represented by `SegmentIndividual` objects, which are concatenated to a `MissionIndividual` to make up an end-to-end trajectory solution (see Figure 5). Each `SegmentIndividual` contains a `ParameterBook` and `TagBook`, which, when initialised, fully define the solution.

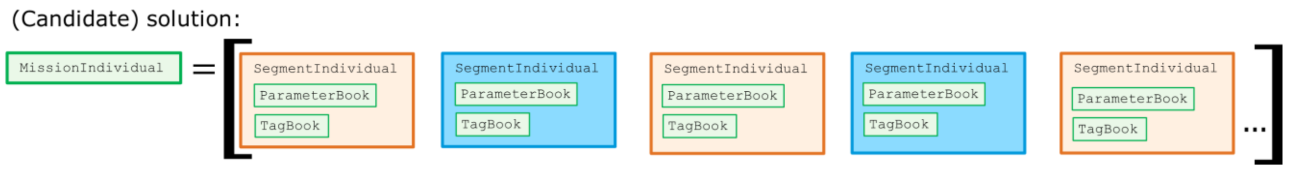


Figure 5: `SegmentIndividuals` concatenated to a `MissionIndividual` in SALTO.

The underlying “scenario” or template for a `SegmentIndividual` is defined by different `Segment` types. These can be configured and used to represent various mission parts, like launches, flybys, ballistic arcs or (pseudo-)resonant arcs. `Segment` objects can be of two types: `StateSegment` and `TransferSegment`. The general rule is that a `Mission` must contain an alternating sequence of `StateSegment` and `TransferSegment` objects with a `StateSegment` in the beginning and the end (see Figure 6). `StateSegments` represent a spacecraft state in any parameterisation (Keplerian, Cartesian, Equinoctial…) and an epoch. `TransferSegments` connect these `StateSegments` with ballistic arcs and/or manoeuvres (both high and low thrust).

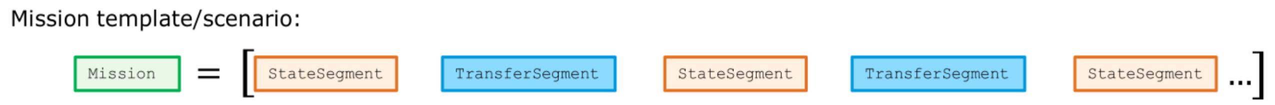
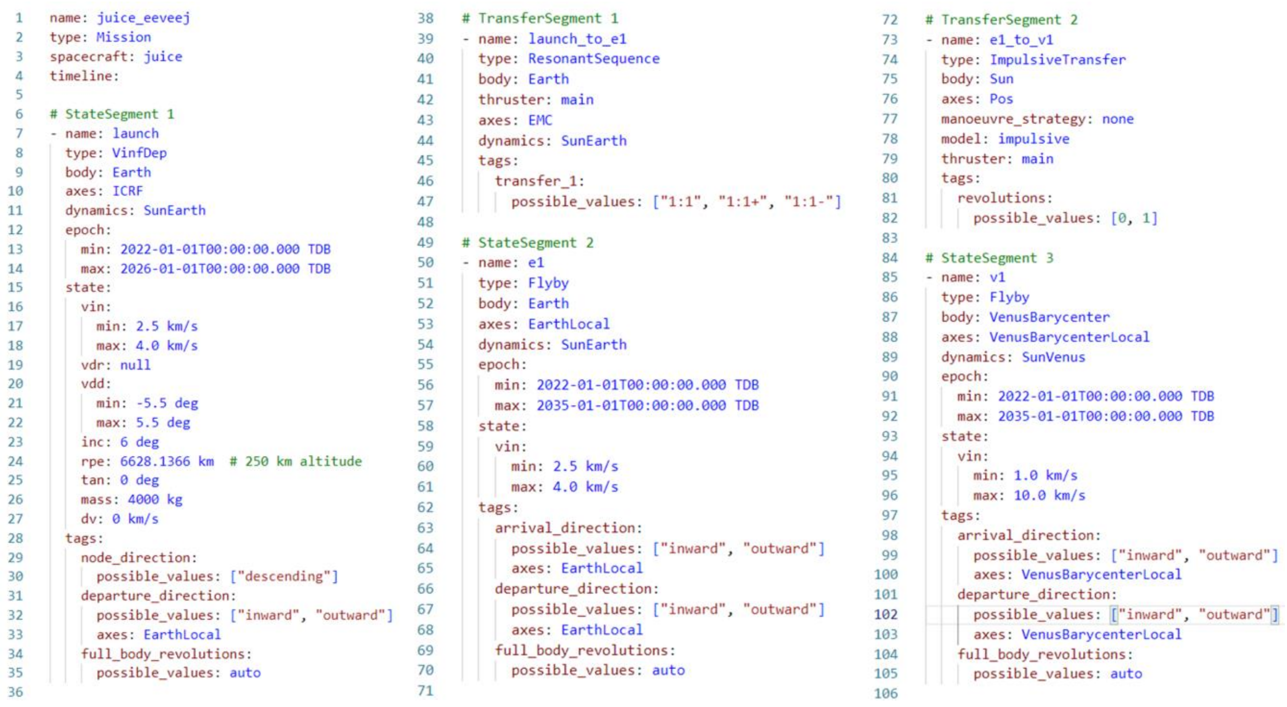


Figure 6: `Segments` concatenated to a `Mission` in SALTO.

A `Mission` can also be constructed programmatically, but in most cases, it is more convenient to do that from a `yaml` configuration file using the factory method `Mission.from_config()`. An example of such a configuration file is shown in
Figure 7.



```yaml
name: juice_eeveej
type: Mission
spacecraft: juice
timeline:

# StateSegment 1
- name: launch
  type: VinfDep
  body: Earth
  axes: ICRF
  dynamics: SunEarth
  epoch:
    min: 2022-01-01T00:00:00.000 TDB
    max: 2026-01-01T00:00:00.000 TDB
  state:
    vin:
      min: 2.5 km/s
      max: 4.0 km/s
    vdr: null
    vdd:
      min: -5.5 deg
      max: 5.5 deg
    inc: 6 deg
    rpe: 6628.1366 km  # 250 km altitude
    tan: 0 deg
    mass: 4000 kg
    dv: 0 km/s
  tags:
    node_direction:
      possible_values: ["descending"]
    departure_direction:
      possible_values: ["inward", "outward"]
      axes: EarthLocal
    full_body_revolutions:
      possible_values: auto

# TransferSegment 1
- name: launch_to_e1
  type: ResonantSequence
  body: Earth
  thruster: main
  axes: EMC
  dynamics: SunEarth
  tags:
    transfer_1:
      possible_values: ["1:1", "1:1+", "1:1-"]

# StateSegment 2
- name: e1
  type: Flyby
  body: Earth
  axes: EarthLocal
  dynamics: SunEarth
  epoch:
    min: 2022-01-01T00:00:00.000 TDB
    max: 2035-01-01T00:00:00.000 TDB
  state:
    vin:
      min: 2.5 km/s
      max: 4.0 km/s
  tags:
    arrival_direction:
      possible_values: ["inward", "outward"]
      axes: EarthLocal
    departure_direction:
      possible_values: ["inward", "outward"]
      axes: EarthLocal
    full_body_revolutions:
      possible_values: auto

# TransferSegment 2
- name: e1_to_v1
  type: ImpulsiveTransfer
  body: Sun
  axes: Pos
  manoeuvre_strategy: none
  model: impulsive
  thruster: main
  tags:
    revolutions:
      possible_values: [0, 1]

# StateSegment 3
- name: v1
  type: Flyby
  body: VenusBarycenter
  axes: VenusBarycenterLocal
  dynamics: SunVenus
  epoch:
    min: 2022-01-01T00:00:00.000 TDB
    max: 2035-01-01T00:00:00.000 TDB
  state:
    vin:
      min: 1.0 km/s
      max: 10.0 km/s
  tags:
    arrival_direction:
      possible_values: ["inward", "outward"]
      axes: VenusBarycenterLocal
    departure_direction:
      possible_values: ["inward", "outward"]
      axes: VenusBarycenterLocal
    full_body_revolutions:
      possible_values: auto

```

Figure 7: The `Mission` configuration file (from launch up to the Venus flyby, v1) in SALTO for the JUICE problem discussed in section 2. Each `Segment` configuration mainly defines the ranges and allowed values for parameters and tags in the respective `ParameterBook` and `TagBook`.

It is important to realise that the actual (candidate) solution is represented by a `MissionIndividual` in SALTO, not by a `Mission`. The latter functions only as a template or blueprint that defines the mission scenario.

3.2 Segmented Problem and Phasing Problem

SALTO provides two main classes to find solutions to global trajectory optimisation problems. These are:

- `SegmentedProblem`: Finds `MissionIndividual` solutions from scratch using `BaseSegmentGuessGenerator` child classes. This corresponds to the non-phased initial guesses generation described in section 2.1.
- `PhasingProblem`: Requires a `MissionIndividual` as an input and uses a combination of tree search and local optimisation described in section 2.3 to expand and improve the search.

As described earlier, the main idea behind this two-step approach is that the `MissionIndividual` objects obtained from calling `SegmentedProblem.solve()` are not required to provide a continuous end-to-end trajectory but must only solve the problem from the energetic point of view using simple techniques typically based on the Tisserand graph. Those `MissionIndividual` objects will often have large state and time gaps in their `TransferSegments`. The main task of the `PhasingProblem.branch_and_optimise()` method is to close those gaps by time-shifting the different `StateSegment` solutions by integer numbers of planet orbital periods or rotation periods and attempting a local optimisation to improve and create related solutions from it. The `PhasingProblem` provides configurability on which variables shall be matched (state variables and/or time).

For simple mission scenarios, like transfers to libration points or Mars transfers, the SegmentedProblem will already provide a good enough initial guess. In that case, the MissionIndividual can directly be converted to a multiple-shooting trajectory configuration to be read by the respective GODOT classes. Alternatively, PhasingProblem.optimise() provides a convenience method to directly run a local optimisation on a given MissionIndividual to improve it. The PhasingProblem supports a number of user configurable constraints, like flyby altitudes. Moreover, any custom user-defined constraint can be added programmatically from Python to the PhasingProblem instance and considered in optimisation runs.

All problem classes are most conveniently constructed from a yaml file using the factory method BaseProblem.from_config().

There are plans to implement a third problem class in the future, the GlobalProblem, which can be used to find initial guesses for a Mission as a whole from scratch using genetic algorithms and a black-box transcription of the problem.

3.3 Guess Generators

The initial guesses obtained from SegmentedProblem.solve() make use of a palette of guess generators that all inherit from the BaseSegmentGuessGenerator class. A guess generator is responsible for initialising the individual of one TransferSegment and the two neighbouring StateSegment individuals. If the Mission scenario contains more than one TransferSegment, the algorithm will move through the sequence of TransferSegment objects and initialise them as indicated in Figure 8. In each initialisation step the individuals of one triplet, (StateSegment, TransferSegment, StateSegment) are being initialised using a particular GuessGenerator class. The StateSegment at the interface between the two initialisation steps is only required to be fully initialised after both guess generator calls. In this way, a communication between the different TransferSegment objects is facilitated.

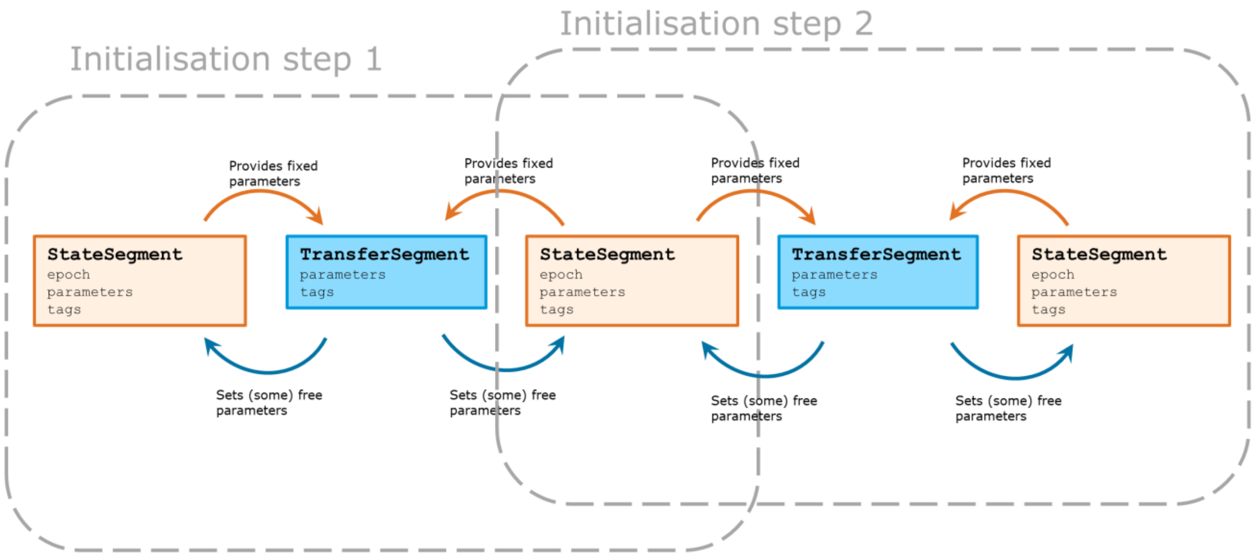

Initialisation step 2
Initialisation step 1
Provides fixed parameters
Provides fixed parameters
Provides fixed parameters
Provides fixed parameters
StateSegment
epoch
parameters
tags
TransferSegment
parameters
tags
StateSegment
epoch
parameters
tags
TransferSegment
parameters
tags
StateSegment
epoch
parameters
tags
Sets (some) free parameters
Sets (some) free parameters
Sets (some) free parameters
Sets (some) free parameters

Figure 8: Initialisation algorithm using GuessGenerators in the SegmentedProblem.

SALTO comes with a range of guess generators for various mission types. However, users can easily implement their own guess generators by inheriting from the `GuessGenerator` class and implementing the `solve` method using their custom algorithms. The procedure described above makes it easy to combine different `GuessGenerators` within the same `Mission` scenario.

3.4 Parameters and ParameterBook

A solution to a trajectory optimisation problem is defined by parameter values such as encounter epochs, Keplerian elements and manoeuvre directions. In SALTO, the `ParameterBook` class is used to manage these. It allows configuring free and fixed parameters and defining their allowed ranges. The `ParameterBook` is one of the two main constituents of a `SegmentIndividual` (the other is the `TagBook`).

3.5 Tags and TagBook

A central concept in SALTO is the utilisation of category variables or `Tags`. These are used to indicate a region in the parameter space where a local minimum is expected. The concept has been introduced in section 2.2.
Mission designers are often not only interested in a global optimum, but like being offered options with qualitatively different trajectories in terms of geometry, transfer duration and Δv. In SALTO, these options are generated by trying to find one `MissionIndividual` for every combination of `Tag` values of the used `Segment` types. In the example of section 2.2, the `Tags` are called *departure_direction* and *arrival_direction* and they are examples of `NICHE` Tags.

The other of the two `Tag` types in SALTO is the `TIMESHIFT` type. Whereas the `NICHE` Tags are taken into account at the level of the `SegmentedProblem` (i.e. the `SegmentedProblem` tries to find an initial guess for every possible `NICHE Tag` value), the `TIMESHIFT Tags` are ignored by the `SegmentedProblem`. Instead, `TIMESHIFT Tags` are used in the `PhasingProblem` to shift the epochs of `StateSegment` individuals by integer planet orbital periods in order to close large time gaps in `TransferSegments`. Examples of `TIMESHIFT` Tags are *full_body_revolutions* and *full_body_rotations*.

The `TagBook` is a collection of all `Tags` belonging to a `Segment` class and allows configuring the active `Tags` and their allowed values. It is one of the two main constituents of a `SegmentIndividual` (the other is the `ParameterBook`).

3.6 Collection of `Segment` types in SALTO

Each `StateSegment` type in SALTO has a different parameterisation suitable for different applications. E.g. a state for a direct escape launch is conveniently parameterised using the `KepC3 StateSegment` because it allows specifying the inclination with respect to the Earth's equator as well as the direction of the escape infinite velocity. The `Flyby StateSegment` is useful for modelling planetary flybys where the incoming and outgoing infinite velocities are used.

`StateSegments` are connected by `TransferSegments` in SALTO. These represent the actual propagation of the spacecraft. Simple ballistic arcs that optionally contain impulsive manoeuvres are modelled by the `ImpulsiveTransfer` segment. Special `TransferSegment` types exist for same-planet (resonant, pseudo-resonant, backflip, v-infinity leveraging) transfers and multi-revolution low-thrust arcs. For a detailed description of the different segment types, please refer to the online documentation [**10**].

3.7 Collection of initial guess generators in SALTO

The currently implemented guess generator types include algorithms based on the Lambert problem, the Tisserand graph, resonant, pseudo-resonant, backflip, v-infinity leveraging transfer searches, bisection for libration point transfers and multi-revolution low-thrust transfers based on a collocation method. More algorithms are continuously added. Again, a more detailed description of all the algorithms can be found in the online documentation [**10**].

4 USE CASES AND USER EXTENSIBILITY

4.1 Supported Mission Types

At the current state of development, SALTO supports the following mission types:

- Interplanetary missions with high-thrust initial guesses: these includes multi-gravity assist trajectories with chemical propulsion like the ones used by JUICE, SOLO and Rosetta and low-thrust interplanetary missions, like BepiColombo.
- Libration point missions: one-leg transfers from Earth to any of the libration points, like the Sun-Earth L2 point used by Euclid and Ariel, followed up by a station-keeping leg to maintain the Lissajous orbits.
- Multi-revolution low-thrust transfers: the method described in [11] based on collocation and averaging out the fast variable has been implemented. It is applicable to any low-thrust transfer scenario, where the mean orbital elements change slowly over many spacecraft revolutions.

Developments for lunar (low-energy) transfers, few-revolution low-thrust transfers based on collocation and support of non-Keplerian insertion orbits (e.g. NRHOs) are ongoing.

4.2 User Extensions

As indicated in section 3, care has been taken to design the software in a modular, user-extensible way, such that specific future mission needs can be implemented without changing the SALTO source code. The main places where user extensions are possible are:

1. `Segment` types: segments are the mission building blocks in SALTO. The most common types are provided, but specific mission needs might require custom `Segment` types. As an example, the transfer design for the LISA mission requires a specific three-spacecraft state representation for the so-called cartwheel formation, which has been implemented as the `Cartwheel StateSegment` and used for the LISA transfer design [12].
2. Initial guess generators: even though the most relevant guess generators for the `SegmentedProblem` are provided in SALTO, a vast variety of algorithms exist in the literature. To meet specific user needs, it is easy to implement a guess generator for a given triplet of `(StateSegment, TransferSegment, StateSegment)`. Such user-implemented guess generators can then be mixed and matched with other guess generators.

4.3 Integration of SALTO in the End-to-end Mission Analysis Workflow

At ESOC, an end-to-end mission analysis workflow is considered to follow the following steps:

1. Early mission design: generation of initial guess trajectories.
2. Trajectory optimisation: refinement of trajectories in the full dynamical model by adding all operational and scientific constraints.
3. Navigation analysis: simulation of the orbit determination and guidance process.
4. (Planetary protection analysis)
5. Product generation: OEM files, AOS/LOS data, geometry plots…
6. Flight dynamics operations

The common low-level software infrastructure of all these steps is GODOT [6]. SALTO is the software to tackle step 1 of the workflow. Being based on GODOT, it allows a seamless integration with all the following steps. The main mechanism by which this integration is achieved is the use of the GODOT `Trajectory` class as the backend for all orbit propagations. The `Trajectory` class is based on a highly configurable multiple-shooting setup that supports a wide range of state representations, manoeuvre models, matching points and dynamical models. It internally employs GODOT's automatic differentiation module, `autodif`, and is mainly used as the propagation engine for trajectory optimisation in GODOT. For more information, please refer to the GODOT online documentation [7].

All solutions generated by SALTO can directly be saved as a `Trajectory` configuration `yaml` file that is used for construction of that class. Therefore, no tedious manual work is required to convert initial guesses to a format suitable for high-fidelity local optimisation. The same `Trajectory` can also be used for the rest of the above workflow, in particular, as an input to the navigation analysis (step 3).

Both GODOT and SALTO use pygmo [13] as an interface to local optimisation algorithms. Therefore, any algorithm that is supported by pygmo can also be used for the local optimisation tasks in SALTO. This includes NLOPT, SNOPT, WORHP, IPOPT and Scipy algorithms. However, it was found that SALTO works in a particularly robust way with Pyoptgra [14]. This is an Open Source Python-wrapped version of the in-house local optimiser, OPTGRA. It was specifically developed at ESOC for close-to-linear optimisation problems with many constraints and is thus tailored for multiple-shooting trajectory optimisation problems. The distinguishing feature of OPTGRA is the focus on efficient constraint satisfaction. The minimisation of the cost function is only attempted once all constraints are satisfied. This is different in most other gradient-based optimisers but is exactly what is required to solve the phasing problem in SALTO.

5 CONCLUSIONS AND FUTURE WORK

The novel mission design framework, SALTO, has been introduced and its main working principles, as well as the software design were described. SALTO is a very flexible toolset that was initially conceived for the design of multi-gravity assist trajectories, but can be applied to other mission types, like lunar and libration point missions as well. The SALTO algorithm is distinctly different from other popular initial guess generation tools because it is fully based on a physical understanding and intuition of the trajectory design problem, which was developed at ESOC over the last decades. Its seamless integration into the end-to-end mission analysis workflow via GODOT will significantly increase the efficiency of trajectory design and analysis in ESA. SALTO is also freely available to the European industry and academia as part of the MIDAS package under

the ESA Community License (Community Open Source). It is accessible through the space-codev.org platform.
For brevity reasons, the current paper can only give a high-level overview of such a complex software tool. The interested reader is advised to explore the online documentation, which is constantly expanded [10].
SALTO is being continuously developed and improved. Ongoing and planned developments include the following:

- A fast, low-fidelity trajectory backend: as described in section 4.3, all trajectory propagation is currently being done via the GODOT `Trajectory` class, which uses numerical integration. For an initial guess generation tool such a high fidelity is not needed and unnecessarily slows down the process. A low-fidelity version of the `Trajectory` class using Keplerian propagation and other fast methods, such as Sims-Flanagan low-thrust arcs [15], is currently being developed. The user will be able to configure the used `Trajectory` backend in the `PhasingProblem` depending on their fidelity and speed needs.
- Few-revolution low-thrust transfers: the current support of low-thrust trajectories in SALTO is limited to many-revolution transfers, where the change of orbital elements over one revolution is small enough to be averaged over, and low-thrust from impulsive initial guesses. It is envisioned to add a `GuessGenerator` that can create low-thrust initial guesses based on a collocation method for few revolution transfer. This addition will be useful for interplanetary low-thrust missions.
- Lunar transfers: support of lunar Hohmann transfers between Earth and Moon will be rather straightforward to implement with the existing code base. Additionally, low-energy transfers that use lunar resonances [16] or go via the Weak Stability Boundary [17] shall also be supported.
- Non-Keplerian target orbits: currently only Keplerian insertion orbits are supported via the implemented `StateSegment` types. For some mission types, such as lunar missions involving an NRHO, this is not sufficient. A support of non-Keplerian target orbits shall the added to SALTO to cover such missions.
- Linking of mission phases: the software structure is very suitable for designing different parts of a mission (e.g. interplanetary transfer and moon tour) separately and then linking them together by closing potential state and time gaps between the phases.

6 ANNEX: TISSERAND GRAPHS

To make this paper as self-contained as possible, this annex explains the basics of the Tisserand graph, which is extensively used in SALTO: when designing multi-flyby trajectories, it can be useful to start with a model where all planet orbits are circular and planar and the spacecraft trajectory is also confined to the plane. Then, all spacecraft orbits crossing a planet at a given infinite velocity can conveniently be represented as a contour line in a graph with the aphelion and perihelion radii on its axes (see Figure 9). Since a planetary flyby does not change the infinite velocity magnitude, orbits connected by a flyby will all be located on the same contour line. This is very handy because it shows at a single glance what aphelion and perihelion radii can be reached using flybys.

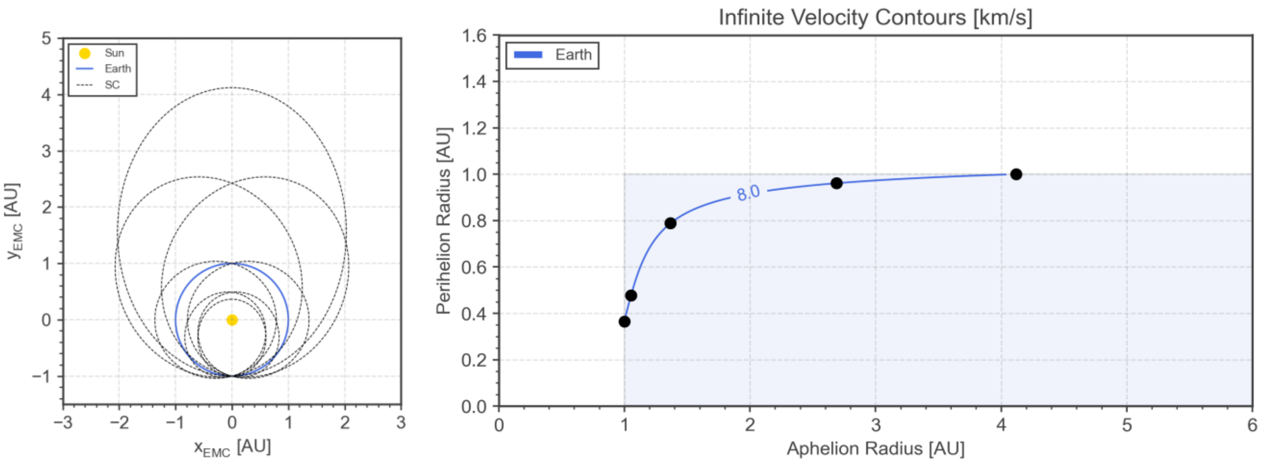


Figure 9: The family of orbits crossing the Earth with an infinite velocity of 8 km/s (left) and the corresponding contour line in the Tisserand graph (right). The black dots indicate the orbits shown on the left.

This aphelion-perihelion representation is called a Tisserand graph. Its full potential for trajectory design becomes apparent, when contour lines of all planets are superimposed in the same plot and the maximum allowed travel distances (due to the minimum flyby altitude constraint) along the contour lines are also indicated (see Figure 10). A point in the plot represents a spacecraft orbit. It is immediately apparent which planet orbits are crossed by a given orbit and at what infinite velocity the encounter occurs. A displacement along a contour line represents a flyby.

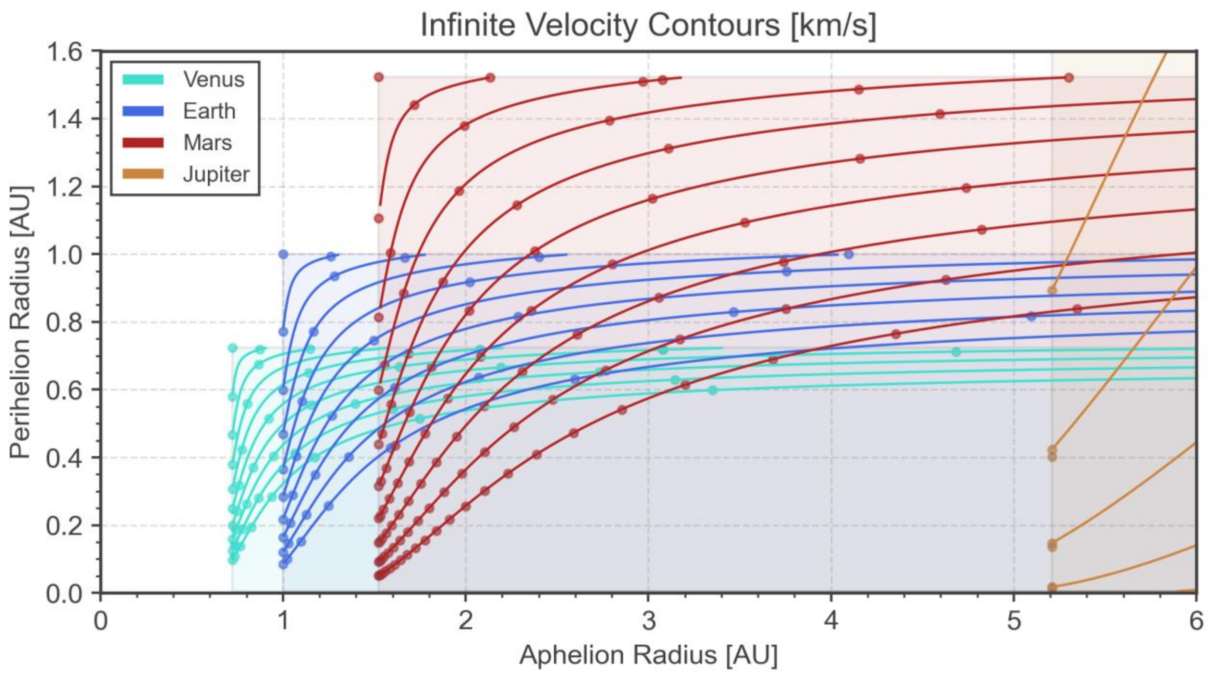


Figure 10: Tisserand graph for Venus, Earth, Mars and Jupiter. The contour lines are spaced by 2 km/s in infinite velocity. The dots on the contour lines indicate the maximum distance by which a 300 km altitude flyby can change the orbit.

A path along the different contour lines represents an interplanetary trajectory with flybys, such as the one shown in Figure 1. The graph is very helpful for finding suitable planet sequences for interplanetary missions. For instance, the JUICE transfer aims at starting at Earth with an infinite velocity around 3 km/s and arriving at Jupiter minimising the infinite velocity. This fixes the start and end point of the trajectory in the graph. By manually testing different ways of connecting these

two points, it quickly becomes apparent that certain planet sequences, like the famous Earth-Venus-Earth-Earth-Jupiter sequence is expected to work particularly well.
Tisserand graphs are tools to solve the transfer problem from the energetic perspective, i.e. they neglect the phasing between the planets and the spacecraft, but their simplicity has made them a popular means for preliminary trajectory design.
Tisserand graphs can be extended for the use with low-energy transfers, i.e. weak stability boundary transfers, lunar resonances and weak capture (see [**18**] for details). They can therefore support initial guess generation for a wide class of mission types in a tool like SALTO, where large state and time gaps are allowed at the initial stage.

7 REFERENCES

[1] Vikhar P. A., *Evolutionary Algorithms: A Critical Review and its Future Prospects*, 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Jalgaon, India, 2016, pp. 261-265

[2] Becerra V. M., Myatt D. R., Nasuto S. J., Bishop J. M., Izzo, D., *An Efficient Pruning Technique for the Global Optimisation of Multiple Gravity Assist Trajectories*, Proceedings of the GO 2005 Conference, Almeria, Spain, 2005

[3] Izzo D., *Advances in Global optimisation For Space Trajectory Design*, Paper ISTS 2006-d-45, 25th International Symposium on Space Technology and Science, Japan, 2006

[4] Vasile M., Ceriotti M., Radice G., Becerra V. M., Nasuto S., Anderson J., *Global Trajectory Optimisation: Can We Prune the Solution Space when Considering Deep Space Manoeuvres?*, European Space Agency, the Advanced Concepts Team, Ariadna Final Report (06-4101c), 2007

[5] Boutonnet A., Martens W. and Schoenmaekers J., *SOURCE: A Matlab-oriented Tool for Interplanetary Trajectory Global Optimization, Fundamentals (Part I)*, Paper AAS 13-300, AAS/AIAA Astrodynamics Specialist Conference and Exhibit, Kauai, Hawaii, Feb 2013

[6] Mackenzie R. and Varga G. 2023 *GODOT, ESA astrodynamics infrastructure software for operations and mission analysis*, Proceedings of the 9th International Conference on Astrodynamics Tools and Techniques, 214

[7] GODOT software documentation https://godot.io.esa.int/docs

[8] Boutonnet A, Langevin Y. and Rocchi A., *JUICE Interplanetary Phase: Trajectory Design and Navigation*, Paper AAS 23-204, AAS/AIAA Space Flight Mechanics Meeting, Austin, Texas, Jan 2023

[9] Strange N. J., and Longuski J. M., *A Graphical Method for Gravity-Assist Trajectory Design*, Journal of Spacecraft and Rockets 39.1 (2002), 9-16

[10] MIDAS software documentation, https://midas.io.esa.int/midas/

[11] Olikara, Z. P., *Framework for Optimizing Many-Revolution Low-Thrust Transfers*, Paper AAS 18–332. 2018, AAS/AlAA Astrodynamics Specialist Conference

[12] Martens, W, Joffre, E., *Trajectory Design for the ESA LISA Mission*, The Journal of the Astronautical Sciences 68.2 (2021): 402-443.

[13] Biscani F., Izzo, D., *A Parallel Global Multiobjective Framework for Optimization: pagmo*. Journal of Open Source Software, 5(53), 2338, 2020

[14] Pyoptgra github repository: https://github.com/esa/pyoptgra

[15] Sims, J. A., Flanagan S. A., *Preliminary Design of Low-Thrust Interplanetary Missions* AAS/AIAA Astrodynamics Specialist Conference, Girdwood, Alaska, Aug. 1999.

[16] Schoenmaekers, J., Horas, D., Pulido, J. (2001). *SMART-1: With Solar Electric Propulsion to the Moon*, in Proceeding of the 16th International Symposium on Space Flight Dynamics, Pasadena, CA, 3–7.

[17] Belbruno, E. (1987), *Lunar Capture Orbits, a Method of Constructing Earth Moon Trajectories and the Lunar GAS Mission*, in 19th International Electric Propulsion Conference, 1054

[18] Martens, W. Bucci, L., *Double Tisserand Graphs for Low-Energy Lunar Transfer Design*, Frontiers in Space Technologies, 30 September 2022, 3