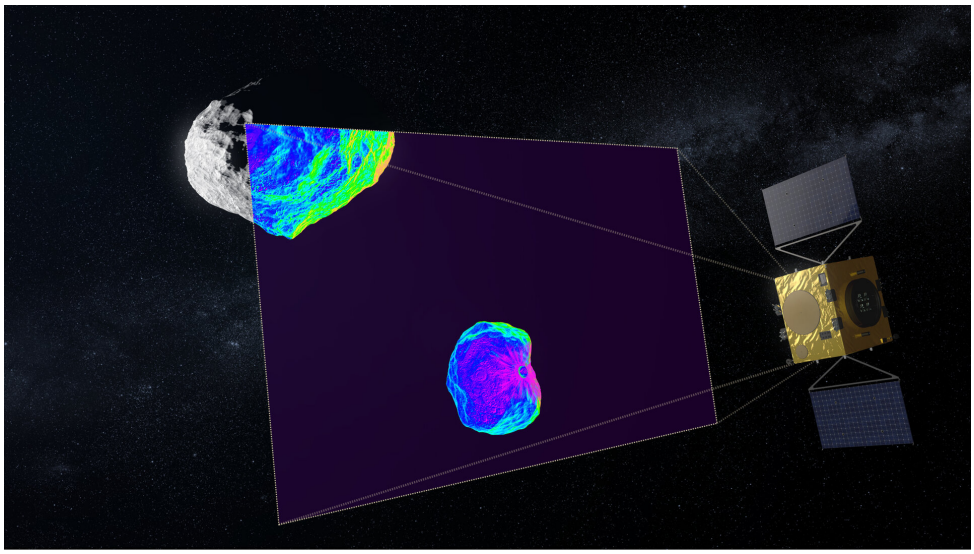
ARISTOTLE UNIVERSITY OF THESSALONIKI

MASTER THESIS

Image-based orbit determination of the Didymos-Dimorphos binary asteroid system using the Hera spacecraft



Author:
Anastasios-Faidon RETSELIS

Supervisor:
Prof. Kleomenis TSIGANIS

A thesis submitted in fulfillment of the requirements
for the Master of Science (MSc) in Computational Physics

July 6, 2022

https://gitlab.com/retse/hera-orbit-determination

Front page image: Hera uses infrared to scan impact crater, © ESA - ScienceOffice.org

ARISTOTLE UNIVERSITY OF THESSALONIKI

Faculty of Sciences

Physics Department

Abstract

Image-based orbit determination of the Didymos-Dimorphos binary asteroid system using the Hera spacecraft

by Anastasios-Faidon RETSELIS

Asteroids in the solar system arguably pose one of the greatest threats to planet Earth. An impact between a sufficiently large asteroid and planet Earth will lead to catastrophic consequences for the Earth and its inhabitants. To avoid such a scenario, several planetary defense missions and techniques have been proposed over the years in literature. One solution which shows sufficient maturity to be implemented is the kinetic impact technique, which utilizes a spacecraft crashing into an asteroid at high speeds. Based on the law of conservation of momentum, the velocity of the asteroid is then changed just a little due to the impact which -if planned correctly- will be sufficient to ensure that the asteroid will miss the Earth entirely and nullify the probability of collision.

The Asteroid Impact and Deflection Assessment (AIDA) mission is humanity's first experimental attempt in planetary defense and will attempt to investigate the kinetic impact approach for the binary asteroid system of Didymos-Dimorphos. It consists of two spacecraft: a) DART, which will crash into Dimorphos (the secondary body of the binary asteroid system) and b) Hera, which will arrive at the system after DART's impact with Dimorphos to investigate the collision further, determine the composition of the two asteroids and accurately characterize the dynamical behavior of the two bodies by performing orbit determination using its instruments.

Inspired by Hera's mission objectives, the possibility of performing image-based orbit determination using solely photographs captured on-board Hera is examined. Considering the lack of actual images due to the fact that Hera is expected to arrive at the binary asteroid system in late 2026, simulated sets of images containing Didymos and Dimorphos are used instead. The orbit determination problem is then transformed into an optimization problem, where the global minimum of a cost function is searched using a metaheuristic algorithm called the Evolutionary Centers Algorithm (ECA). Results of this approach are extremely promising, demonstrating that the algorithm can accurately determine the orbit of Dimorphos with a mean absolute percentage error below 1% for each osculating orbital element. Finally, using Didymos' wobble, the mass of Dimorphos can also be determined with this process with an accuracy lower than 0.1%.

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

Σχολή Θετικών Επιστημών

Τμήμα Φυσικής

Περίληψη

Προσδιορισμός της τροχιάς του συστήματος διπλού αστεροειδή Δίδυμος-Δίμορφος με χρήση φωτογραφιών από το διαστημόπλοιο Hera

από Αναστάσιος-Φαίδων Ρετσελης

Οι αστεροειδείς του ηλιακού συστήματος αποτελούν αναμφισβήτητα έναν από τους μεγαλύτερους κινδύνους για τον πλανήτη Γη. Μια σύγκρουση με έναν αρκούντως μεγάλο αστεροειδή και τη Γη θα έχει καταστροφικές συνέπειες για τη Γη και τους κατοίκους της. Προς αποφυγή ενός τέτοιου σεναρίου, αρκετές αποστολές και τεχνικές πλανητικής άμυνας έχουν προταθεί στη βιβλιογραφία. Μια λύση που δείχνει ωριμότητα ως προς την δυνατότητα υλοποίησης της είναι η τεχνική κινητικής πρόσκρουσης, στην οποία ένα διαστημόπλοιο προσκρούει σε έναν αστεροειδή με υψηλή ταχύτητα. Σύμφωνα με την αρχή διατήρηση της ορμής, η ταχύτητα του αστεροειδή αλλάζει ελάχιστα λόγω της πρόσκρουσης, αν σχεδιαστεί όμως σωστά είναι αρκετή για να διασφαλίσει πως ο αστεροειδής θα αποφύγει την σύγκρουση με τη Γη.

Η αποστολή AIDA είναι η πρώτη πειραματική προσπάθεια της ανθρωπότητας στο πεδίο της πλανητικής άμυνας και θα εξετάσει την τεχνική της κινητικής πρόσκρουσης στο σύστημα διπλού αστεροειδή Δίδυμος-Δίμορφος. Αποτελείται από δύο διαστημόπλοια: α) Το DART, το οποίο θα προσκρούσει στον Δίμορφο (ο οποίος είναι το δευτερεύον σώμα του συστήματος διπλού αστεροειδή) και β) το Hera, το οποίο θα φτάσει στο σύστημα Δίδυμος-Δίμορφος μετά την πρόσκρουση του DART στον Δίμορφο, με στόχο να διερευνήσει περαιτέρω τη σύγκρουση, να ανακαλύψει την σύσταση των δύο αστεροειδών και να πραγματοποιήσει τον δυναμικό χαρακτηρισμό τους συστήματος εκτελώντας προσδιορισμό τροχιάς με τα όργανα του.

Ορμώμενη από τους στόχους της αποστολής Hera, η διπλωματική αυτή εξετάζει την πιθανότητα να πραγματοποιηθεί ο προσδιορισμός της τροχιάς χρησιμοποιώντας αποκλειστικά φωτογραφίες που τραβήχτηκαν από το Hera. Δεδομένης της τωρινής έλλειψης φωτογραφιών, μιας και το Hera θα φτάσει στο σύστημα διπλού αστεροειδή στα τέλη του 2026, χρησιμοποιήθηκαν προσομοιωμένες εικόνες. Το πρόβλημα του προσδιορισμού της τροχιάς στη συνέχεια εκφράζεται ως ένα πρόβλημα βελτιστοποίησης, στο οποίο αναζητείται το ολικό ελάχιστο μιας συνάρτησης με τη χρήση του ευριστικού αλγορίθμου των εξελικτικών κέντρων (ECA). Τα αποτελέσματα είναι αρκετά ενθαρρυντικά, καθώς ο αλγόριθμος ταυτοποιεί την τροχιά του Δίμορφου με μέσο απόλυτο ποσοστιαίο σφάλμα για κάθε μεταβαλλόμενο τροχιακό στοιχείο κάτω του

1%. Τέλος, χρησιμοποιώντας το wobble του Δίδυμου, η μάζα του Δίμορφου μπορεί να ταυτοποιηθεί με ακρίβεια κάτω από 0.1%

Acknowledgements

I started this thesis in a very peculiar moment of my life, during which I faced rejection and uncertainty about my future. Although this thesis was probably the most challenging academic task I have undertaken so far, it was an extremely rewarding experience during which I was able to solidify my passion for computational science and space systems engineering. For this reason, I would like to first thank Prof. Kleomenis Tsiganis, who presented to me a very simple research question about Hera's capabilities to perform orbit determination for Dimorphos, which would form the basis for this thesis. In addition, Prof. Tsiganis provided continuous guidance and support throughout the course of this thesis, for which I am extremely grateful. I would also like to thank Dr. Sebastien Charnoz of the Institut de Physique du Globe de Paris, for kindly providing me with a similar implementation he worked on, which was used to get a better understanding of the problem early on and to get some ideas for my own implementation. Next, I would also like to thank Thanasis Theocharis, with whom I discussed some early ideas for the optimization part of this thesis. I would also like to thank all the members of the AcubeSAT project for working hard on an extremely ambitious and inspiring project with me for the past 5 years, in which we have always been fighting against the odds, working hard to make things right but most importantly ensuring that we were having fun. My friends and family undoubtedly played an important role, listening and supporting me in their own ways for the past year. Last but definitely not least, I have to thank Georgia Pantelidou, for proofreading parts of this thesis, for always believing in me even when I doubted myself and for being my biggest supporter.

I would like to dedicate this thesis to the people working for the humble and peaceful cause of planetary defense, who work in the shadows to slowly but surely ensure that humanity will not be eradicated by a celestial body crashing into our planet. I am sure the dinosaurs would have been pleased to have you as well!

"*The plan is nothing; the planning is everything.*"

- Dwight D. Eisenhower

Contents

List of Figures

List of Tables

List of Abbreviations

AFC	**A**steroid **F**raming **C**amera
AIDA	**A**steroid **I**mpact and **D**eflection **A**ssessment
CK	**C**amera **K**ernel
DART	**D**ouble **A**steroid **R**edirection **T**est
DCP	**D**etailed **C**haracterization **P**hase
ECA	**E**volutionary **C**enters **A**lgorithm
ECP	**E**arly **C**haracterization **P**hase
ELP	**E**nd-of-**L**ife **P**hase
ESA	**E**ropean **S**pace **A**gency
FK	**F**rame **K**ernel
IK	**I**nstrument **K**ernel
JPL	**J**et **P**ropulsion **L**aboratory
LSK	**L**eap **S**econds **K**ernel
NASA	**N**ational **A**eronautics and **S**pace **A**dministration
NEA	**N**ear-**E**arth **A**steroid
NEC	**N**ear-**E**arth **C**omet
NEO	**N**ear-**E**arth **O**bject
PHA	**P**otentially **H**azardous **A**steroid
SPICE	**S**pacecraft **P**lanet **I**nstrument **C**-matrix **E**vents
SPK	**S**pacecraft **P**lanet **K**ernel
SSR	**S**um of **S**quared **R**esiduals

List of Symbols

a	Semi-major axis	km
e	Eccentircity	-
i	Inclination	$^\circ$
M	Mean anomaly	$^\circ$
E	Eccentric anomaly	$^\circ$
T	Orbital period	s
t	Time	s
u	Argument of latitude	$^\circ$
m	Mass	kg
I	Moment of inertia	$\mathrm{kg\,km^2}$
r	Position	km
v	Velocity	$\mathrm{km\,s^{-1}}$
J_2	Second zonal harmonic	-
f	Focal length	mm
ν	True anomaly	$^\circ$
Ω	Right ascension of the ascending node	$^\circ$
ω	Argument of periapsis	$^\circ$
$\tilde{\omega}_{true}$	True longitude of periapsis	$^\circ$
λ_{true}	True longitude	$^\circ$

Chapter 1

Introduction

1.1 Planetary defense

The final law provided in Akin's Laws of Spacecraft Design states that *"Space is a completely unforgiving environment. If you screw up the engineering, somebody dies."* [1]. This law was written for engineers designing and building crewed space systems and it highlights the facts that space is an extremely hostile environment towards humans. However, the issue at hand is that space is also hostile to humans even when they are on their own planet. Of significant risk to planet Earth is a potential collision with a large asteroid or near-Earth object (NEO), which would cause extreme environmental phenomena such as massive tsunamis, multiple firestorms and potentially an impact winter caused by dust particles and debris rising to the stratosphere. All of this would ultimately lead to a mass extinction event, in which several species and life forms could not survive on the planet anymore. In fact, this has already happened once in the history of the Earth around 66 million years ago, when an object about 10 kilometers wide hit our planet in North America and triggered the *Cretaceous-Paleogene* extinction event, which is the event that caused the extinction of most dinosaurs [2].

1.1.1 Near-Earth Objects

Fortunately, unlike dinosaurs, humanity has created and developed resources dedicated to cataloging near-Earth objects, estimating the likelihood of Earth impacts and developing methods to avoid a potential collision. These resources are usually realized as a dedicated office within a larger government mandated space agency. Following the conventions used by the European Space Agency's Near-Earth Objects Coordination Centre (NEO), we can define **near-Earth objects** as asteroids or comets with a perihelion distance of $\leq$1.3 au, with comets having the additional requirement of having a period shorter than 200 years [3]. Furthermore, we can define **Potentially Hazardous Asteroids (PHAs)** as asteroids that have a Earth Minimum Orbit Intersection Distance (MOID) of 0.05 au or less, combined with an absolute magnitude H of 22 or brighter [3].

NASA's Jet Propulsion Laboratory has a dedicated Center for Near Earth Object Studies (CNEOS). CNEOS publishes rough statistics to monitor the progress of annual NEA discoveries, which can be seen in Figure 1.1. By comparing JPL's data with data from IAU's Minor Planet Center, it can be determined that from 28 500 objects discovered as of March 2022, 2271 of them are characterized as Potentially Hazardous Asteroids (PHAs) [4].

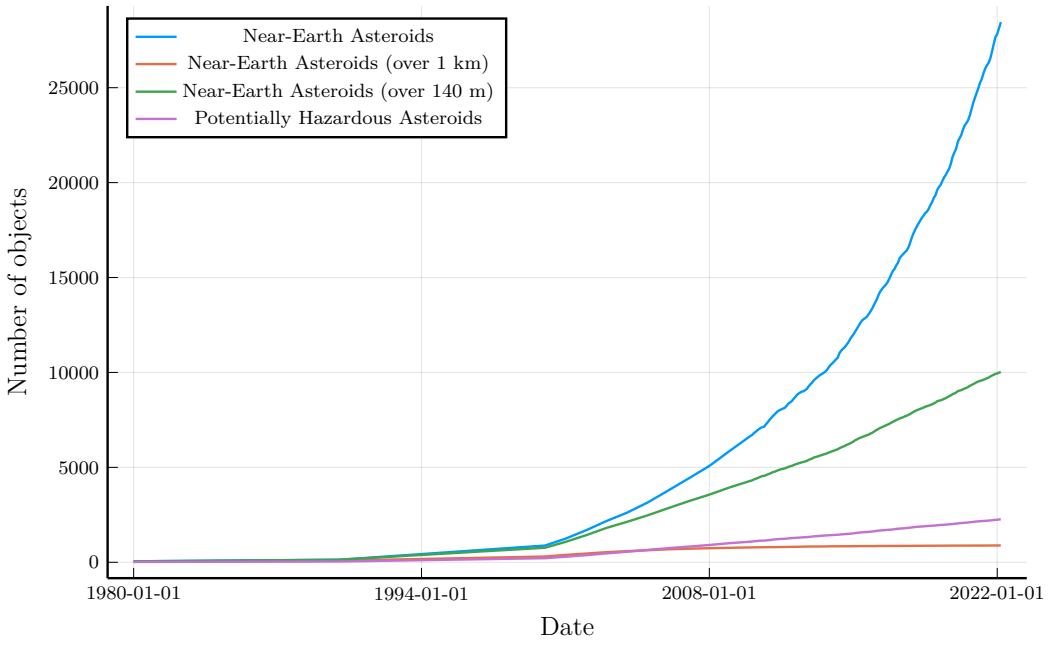


FIGURE 1.1: Near-Earth Asteroids cataloged over the years. Today, about 8% of them are classified as Potentially Hazardous Asteroids (data from [5])

1.1.2 Techniques for collision avoidance

Different techniques have been proposed over the years to avoid a collision between the Earth and a Potentially Hazardous Asteroid [7]. All of these mitigation techniques achieve one of the following final outcomes:

1. **Deflection of the asteroid**, meaning that the asteroid's orbit is changed to the point that the collision probability with the earth is eliminated so that the asteroid no longer poses a threat to the planet.

2. **Fragmentation of the asteroid**, meaning that the asteroid is being reduced into smaller pieces which can then be disintegrated during their entry to the atmosphere due to drag forces developed.

Deflection techniques are preferred, due to the fact that they can be planned in advance several months or even years prior to the time of closest approach and that they completely eliminate the probability of collision. Fragmentation techniques have to be carefully planned: if the asteroid is not reduced to sufficiently small pieces then there is a chance that it won't completely burn up upon entry to the atmosphere,

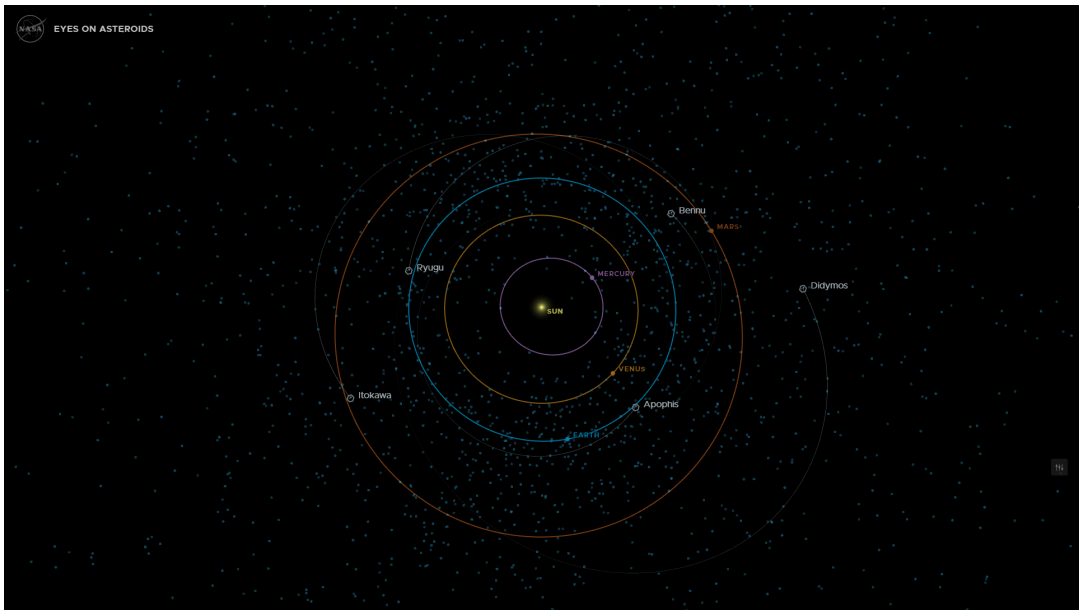


FIGURE 1.2: Mapping of Potentially Hazardous Asteroids (as light blue points) for March 8th, 2022 (extracted from [6])

leading to catastrophic results for Earth and its inhabitants. Let's briefly explore some of the techniques that have been proposed:

1. **Use of thermonuclear weapons**. The idea behind this strategy is to eliminate any major collision by splitting the asteroid into smaller pieces using a thermonuclear device (fragmentation technique) [8, 9] or to detonate the weapon on or close to the surface of the asteroid (stand-off technique), which would be sufficient to change its orbit path [10]. This technique is considered to be 10 to 100 times more effective than non-nuclear alternatives [11]. However, apart from high development and operational risks, this technique entails serious international concerns and public policy issues which relate to the use of nuclear weapons in outer space [12]. Nevertheless, the possibility of using thermonuclear devices for the peaceful purpose of defending the planet can be definitely explored further.

2. **Use of slow push techniques**, such as the gravity tractor concept [13]. The idea here is to use massive unmanned spacecraft to change the asteroid's orbit using the forces of gravity, leading to a deflection of the asteroid. These type of technique is found to be very expensive and requires years or even decades to complete the deflection process [11].

3. **Use of kinetic impact techniques**. Probably the simplest method to implement, kinetic impact uses a large spacecraft and places is into a collision course with the asteroid, resulting into a transfer of speed from the spacecraft to the asteroid following the law of conservation of momentum [14]. These methods are considered to be the most mature approach that can be investigated using today's technology [11].

1.2 The Asteroid Impact and Deflection Assessment (AIDA)

The Asteroid Impact and Deflection Assessment (AIDA) mission is an international collaboration between NASA and ESA. AIDA will make the first demonstration of asteroid deflection by kinetic impact [15]. AIDA is also the successor to the Don Quijote spacecraft mission studied by ESA [16]. Two independent missions are combined to form the AIDA mission. The **Double Asteroid Redirection Test** (DART) by NASA, is the kinetic impact part of the mission, while ESA's **Hera** mission is the characterization spacecraft which will survey the double asteroid system after DART's impact to the smaller asteroid body [15].

1.2.1 Didymos & Dimorphos

Didymos and Dimorphos is a binary asteroid system in our solar system. It is in a elliptical heliocentric orbit around the sun, with ar orbital period of about 2 years, which can be seen in Figure 1.3. Based on its properties, the Didymos-Dimorphos binary asteroid system is classified as a Potential Hazardous Asteroid. It has been chosen for the kinetic impact experiment due to the fact that it is a binary asteroid system: a kinetic impact mission to a binary asteroid system can allow for the photometric light-curve measurements of the target system from Earth-based facilities and comparing them before and after the experiment to determine the change in the orbit of the system which would enable validation of the mission from ground based observations solely.

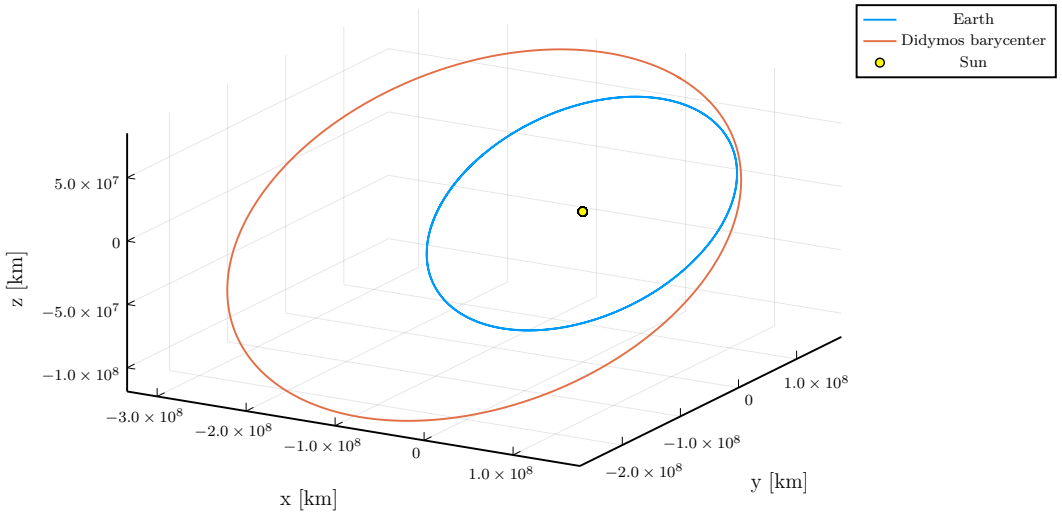


FIGURE 1.3: The orbit of the Earth and the orbit of the Didymos-Dimorphos system around the Sun. The close proximity of the orbits in combination with some properties of the asteroids leads to the binary system being characterized as a Potentially Hazardous Asteroid.

Didymos is the larger body in the system and is considered to be the primary body. Dimorphos on the other hand is the secondary body which is in an almost circular orbit around Didymos [17]. Light-curve measurements mentioned before would

explore the change of orbital period for the orbit of Dimorphos around Didymos, which means that a kinetic impact experiment will have to target the secondary body to enable ground based observations, making this system a very good candidate to explore the kinetic impact concept. Furthermore, analysis of the reflectance spectrum of the system indicates that its composition is consistent with the L/LL chondrites, which is the composition of the most common meteorite falls leading to the potential use of experimental results to a large number of planetary defense scenarios [17].

1.2.2 The DART mission

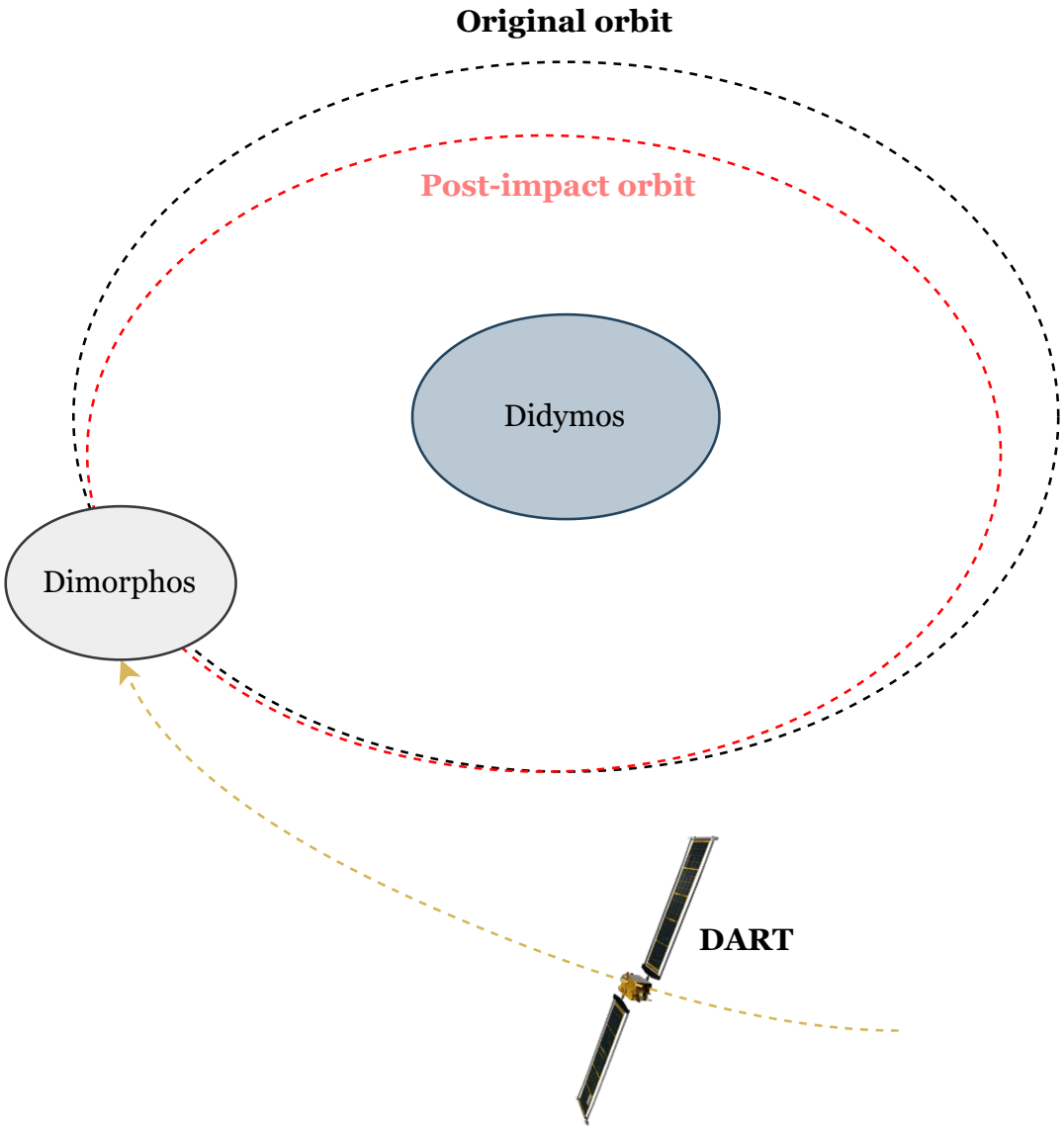


FIGURE 1.4: Mission overview for the DART spacecraft. DART will crash into Dimorphos, altering the latter's orbit around Didymos

The Double Asteroid Redirection Test (DART) is a planetary defense demonstration which was launched in November 2021 and is managed by NASA's Planetary Defense Coordination Office [17]. DART will attempt to crash into the secondary

body (Dimorphos) of the binary asteroid system mentioned before, with the goal of demonstrating the kinetic impact concept for the purposes of planetary defense. The impact is expected to take place in September 2022. DART will alter Dimorphos' orbit around a little bit, a concept which is illustrated in Figure 1.4. This change in orbital period should be enough to be observed even by ground measurements from Earth. DART's main mission requirements are [17]:

1. DART shall intercept the secondary member of the binary asteroid Didymos as a kinetic impactor during its 2022 close approach to Earth.
2. The DART impact on the secondary member of the Didymos system shall cause at least a 73 s change in the binary orbital period.
3. The DART project shall characterize the binary orbit with sufficient accuracy by obtaining ground-based observations of the Didymos system before and after spacecraft impact to measure the change in the binary orbital period to within 7.3 s (1σ confidence).
4. The DART project shall use the velocity change imparted to the target to obtain a measure of the momentum transfer enhancement parameter referred to as "Beta" (β) using the best available estimate of the mass of Didymos B.
5. The DART project shall obtain data, in collaboration with ground-based observations and data from another spacecraft (if available), to constrain the location and surface characteristics of the spacecraft impact site and to allow the estimation of the dynamical changes in the Didymos system resulting from the DART impact and the coupling between the body rotation and the orbit.

1.2.3 The Hera mission

Given the fact that Hera is the main subject of this thesis, let's spend a bit of time to understand its objectives and the different mission phases. Hera is the European component of the AIDA mission and its main objective is to investigate the changes in geophysical and dynamical properties of the target binary asteroid after the DART impact [18]. Hera will thus be the first spacecraft to observe at a close range the outcome of a kinetic impact deflection test.

Hera hosts two main payloads, namely:

1. The **Asteroid Framing Camera (AFC)**, which will be used for Guidance, Navigation & Control (GNC) and scientific measurements [18]. Several scientific measurements are expected to be obtained using the AFC, including [18]:
   (a) Measurement of Dimorphos' mass, by measuring the wobble of Didymos with an accuracy equal to or less than 1 m.
   (b) Measurement of the binary system orbital period, spin parameters and semi-major axis of Dimorphos with an accuracy of 1% or better.

(c) Measurement of irregular rotations (if any), by observing several orbits.

(d) Study of chemical and mineralogical properties of the surface material.

(e) Study the geomorphology and crater density for both asteroids.

2. The **LIDAR** instrument, which will be used to measure the shape of both objects in the Didymos-Dimorphos systems [18]. Measurements using the LIDAR include [18]:

   (a) Obtaining an accurate three-dimensional shape for Dimorphos.

   (b) Measurement of Dimorphos' mass, by measuring the wobble of Didymos with an accuracy equal to or less than 1 m.

   (c) Mapping of Dimorphos' surface topography.

   (d) Support the study of surface chemical and mineralogical properties by measuring intensity or albedo returned.

Hera is currently scheduled to launch in October 2024 and will arrive to the Didymos-Dimorphos system by late 2026 using the approach phase. After arriving to the binary asteroid system, Hera will begin with the proximity operations phases, which are split into four different phases according to [19], namely:

1. **Early Characterization Phase (ECP)**, in which Hera remains at a distance beyond the gravitational sphere of influence (about 30 km), with the objective of conducting a physical and dynamical characterization of the binary asteroid system via imaging. The scheduled orbit for Hera for the Early Characterization Phase can be seen in Figure 1.5.

2. **Detailed Characterization Phase 1 (DCP1)**, in which Hera will approach at a minimum distance of 10 km from the binary asteroid's system barycenter, enabling the accurate characterization of the mass of Dimorphos by measuring the wobble period of Didymos. A part of the orbit planned for DCP1 can be seen in Figure 1.6.

3. **Payload Deployment Phase (PDP)**, in which Hera the two CubeSats which Hera hosts will be deployed and commissioned until they reach full operational capabilities.

4. **Detailed Characterization Phase 2 (DCP2)**, which is a phase equivalent to the DCP1 but also combines the operation of the two CubeSats.

5. **Detailed Characterization Phase 3 (DCP3)**, in which Hera will progressively approach Dimorphos at a distance which is expected to reach as low as 2 km from the surface of Dimorphos, which will enable the accurate characterization of the impact crater. In addition, high-risk autonomy demonstration experiments can take place at this stage. Part of the orbit planned for DCP3 can be seen in Figure 1.7.

6. **End of life Phase (ELP)**, in which the spacecraft will be disposed by landing on the surface of Dimorphos. Spacecraft operations can continue to gain experience in proximity operations near asteroids in support of future resource utilization missions.

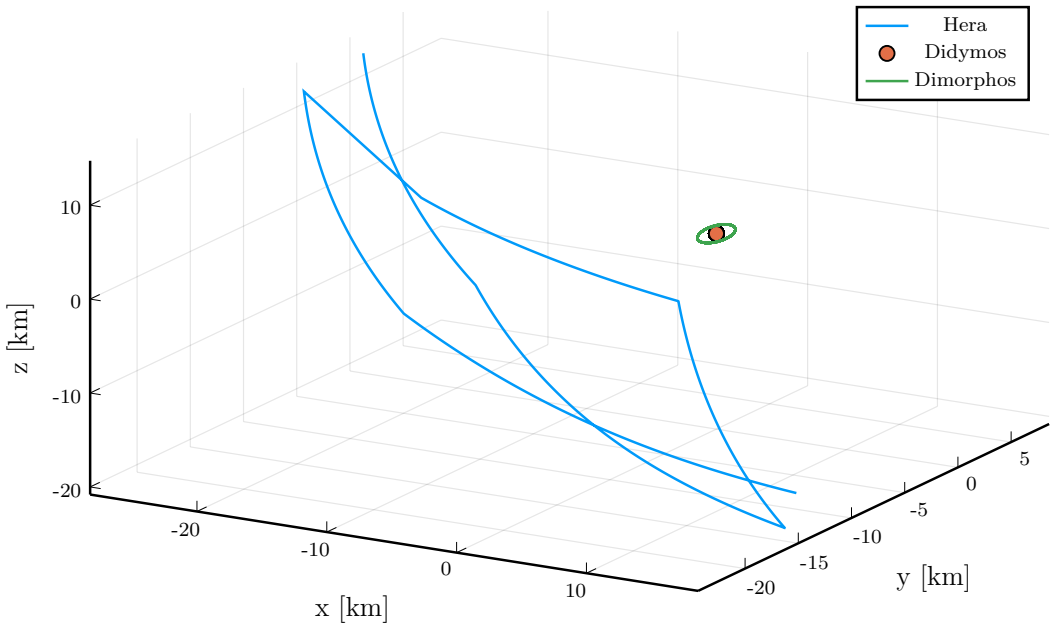


FIGURE 1.5: The orbit of Hera with respect to the barycenter of the Didymos-Dimorphos system during the Early Characterization Phase

1.3 Scope & quick overview of this thesis

As mentioned before, the dynamical characterization of the system and the orbit determination part will take place in the Early Characterization Phase (ECP), using the Asteroid Framing Camera (AFC). The main requirement to be verified during this phase is the value of the semi-major axis for the orbit of Dimorphos. However, the images generated by Hera could theoretically be used to perform a full orbit determination for Dimorphos. The main goal for this thesis is to develop an algorithm which is able to meet Hera's requirement for measuring the semi-major axis of dimorphos with an accuracy of 1% or better. This algorithm can then be tested to see if it is possible to perform full orbit determination for Dimorphos.

Since Hera has not launched into space just yet, a simulated set of images has to be generated to be used by the orbit determination algorithm. In Chapter 2, the orbital dynamics of the Didymos-Dimorphos system will be explored and the orbit of Hera during the Early Characterization Phase (ECP) will be introduced. The process used to generate the simulated images will then be described. To replicate the expected observed images by Hera as accurately as possible, a series of errors will be introduced and included in the generation process for the simulated images.

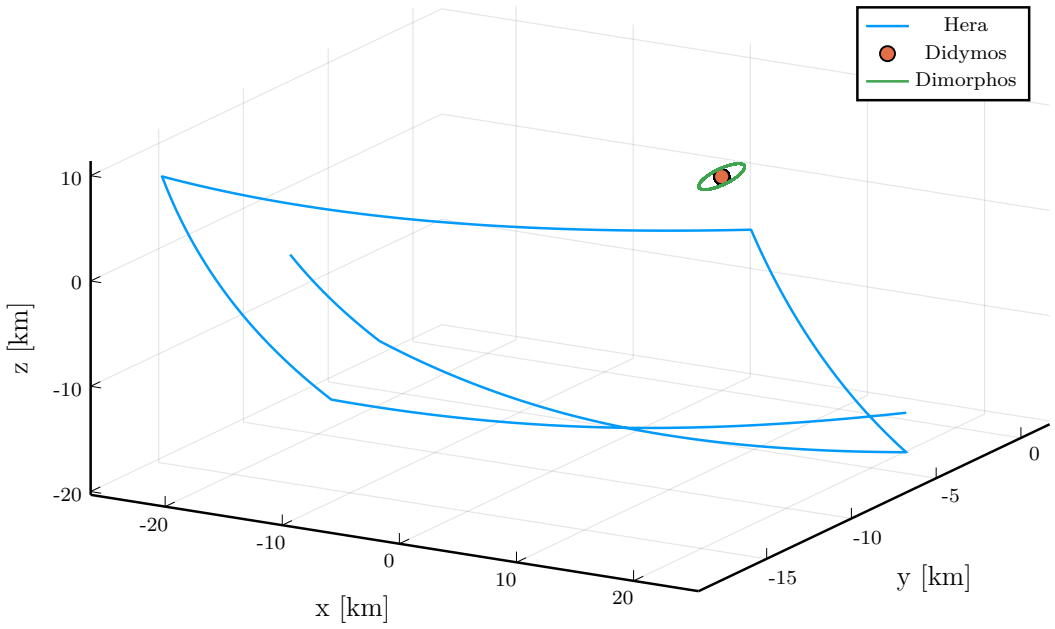


FIGURE 1.6: The orbit of Hera with respect to the barycenter of the Didymos-Dimorphos system during the Detailed Characterization Phase 1

In Chapter 3, the orbit determination algorithm developed for this thesis will be explained in detail. The algorithm transforms the orbit determination problem to an optimization problem, where the minimum of a cost function is searched with the use of a metaheuristic algorithm. The performance metrics used to evaluate the results will then be explained and some preliminary tests for different types of orbits will be performed to determine the performance of the orbit determination procedure.

Finally, the algorithm will be used for the expected orbit of Dimorphos around Didymos in Chapter 4. The orbit determination procedure will be tested using a repeatability analysis for its ability to reproduce the results for one orbit and for its ability to determine a wide range of initial state vectors. Furthermore, the possibility of using fewer images for the orbit determination and the possibility of performing orbit determination using unperturbed keplerian orbital elements will be examined. A discussion of results and future steps will then be provided in Chapter 5.

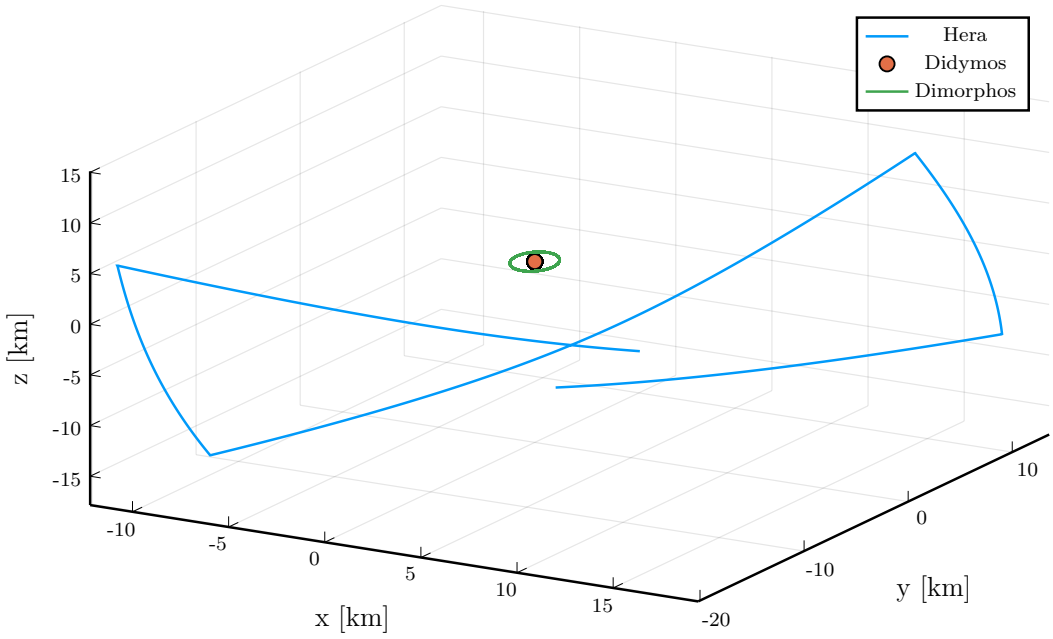


FIGURE 1.7: The orbit of Hera with respect to the barycenter of the Didymos-Dimorphos system during the Detailed Characterization Phase 3

Chapter 2

Simulating photographs captured by Hera

2.1 Programming Language

Given the computational nature of the orbit determination problem introduced in Chapter 1, a decision for the appropriate programming language in which the model will be developed has to be made. In today's academic environment, several scientists chose to develop their code in dynamic languages such as MATLAB or Python, which enhance productivity by giving a variety of tools to developers. However, these dynamic programming languages suffer during problems which can be classified as computationally intensive, leading to the use of languages such as C or Fortran when dealing computationally heavy problems [20].

The problem with the latter languages is that they do not offer the productivity provided by the dynamic languages, which have arguably made the development of scientific code fairly easier, leading to developers having to perform a trade-off for each problem to choose the appropriate language. This problem is frequently described as the two language problem in literature. A solution which combines both the productivity and performance and thus solves the two language problem is Julia [20]. From Figure 2.1, it can be concluded that code written in Julia will achieve similar benchmark times to C [21], while also providing the user with high level functions and options to write code in a more productive manner.

In the context of this thesis, it can be expected that the code developed will be computationally intensive, given the fact that numerical integrators used to perform the propagation of the asteroids for the fitting procedure (which will undoubtedly run several times until a solution is identified), leading to a need for performance. In addition, coordinate transformations, missing objects from the images generated and other pitfalls require a productive way to deal with these problems. Since the two language problem is being faced, the decision has been made early to use Julia as the programming language for the development of this thesis, including the generation of the simulated photographs taken by Hera.

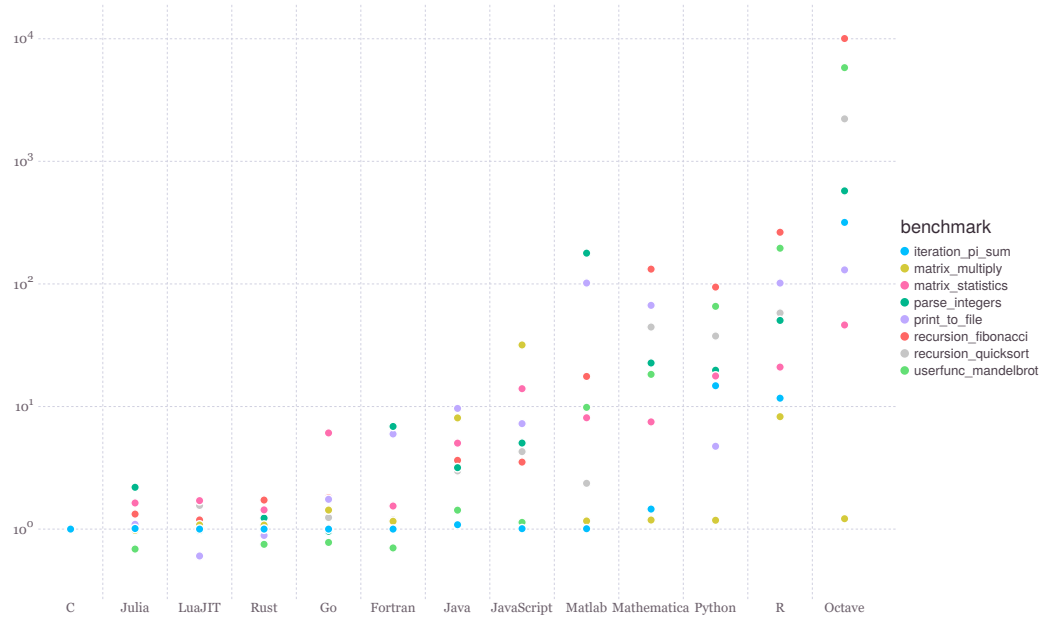


FIGURE 2.1: Normalized benchmark time of some problems against the C implementation for different programming languages [21]

2.2 Orbital Dynamics

For the description of the orbit determination problem being faced from the perspective of astrodynamics, the problem can be simplified into two separate problems, namely:

1. The orbit followed by the Hera spacecraft during the observation period.
2. The behavior of the two asteroids Didymos & Dimorphos.

Once an adequate description for both of these sub-problems has been given, they can then combine them into a single dynamics model using the appropriate reference frames and transformations between them. After this combination is complete, the image generation procedure can be built and different types of errors can be added.

2.2.1 Hera Spacecraft

As briefly discussed in , the Hera spacecraft will follow hyperbolic trajectories with respect to the Didymos-Dimorphos barycenter. This is based on the legacy from a previous mission to a small body. The Rosetta mission visited the comet 67P/Churyumov-Gerasimenko in 2014. The Rosetta spacecraft implemented hyperbolic trajectories for a variety of reasons described by Herfort and Casas [22], which include:

1. Poor gravity potential estimates to plan an accurate circular or elliptic orbit.
2. Large approach velocities for circular orbits when compared to orbital velocities.
3. Lower sensitivity to insertion maneuver errors for hyperbolic trajectories.

4. The hyperbolic trajectory arcs can be designed in a manner which ensures that the sun remains behind the spacecraft at all times, providing good observation conditions.

5. The cost in terms of Δv for the maintenance of the hyperbolic arcs remains fairly small, usually at the order of $\frac{\text{cm}}{\text{s}}$.

These problems remain applicable to the Didymos-Dimorphos asteroid system. Based on the expertise from Rosetta, ESA will command Hera to fly trajectories with peri-centre velocity larger than the escape velocity, with a safety margin C, as defined in Equation (2.1) [19].

$$v_{peri} = (1 + C)\sqrt{\frac{2\mu}{r_{peri}}} \quad , \quad C > 0 \tag{2.1}$$

The safety margin ensures that no collision can occur during an arc due to the presence of uncertainties in the gravity field model and is assigned a value of $C = 0.4$ for Hera proximity operations [19].

2.2.2 SPICE and SPICE.jl

Given all of this information, hyperbolic arcs can be defined at will. However, to ensure consistency with the actual implementation of Hera's orbit, the hyperbolic arcs of Hera will be extracted from SPICE. SPICE provides a portable, multi-discipline mechanism useful in both planning space science observations and defining the position of objects with respect to pre-defined reference frames [23]. SPICE is comprised of different logical elements which also form the acronym SPICE (Spacecraft, Planet, Instrument, Camera matrix and Events) and correspond to different SPICE data products which can then be used by end users for different applications[23]. An overview of the SPICE architecture can be found in Figure 2.2. Most notably, the following SPICE kernels are of interest:

- **SPK**, which provide position and velocity of spacecraft and different bodies as a function of time.
- **LSK**, which house the leapseconds kernel required for time computations.
- **IK**, which house information about specific instruments on-board a spacecraft
- **CK**, which provide orientation of a camera instrument ("camera matrix") for specific time intervals.
- **FK**, which offer access to a number of reference frames which can then be used to transform coordinates between reference frames.

The European Space Agency has a dedicated department called the ESA SPICE Service, which is located at the European Space Astronomy Center. ESA SPICE

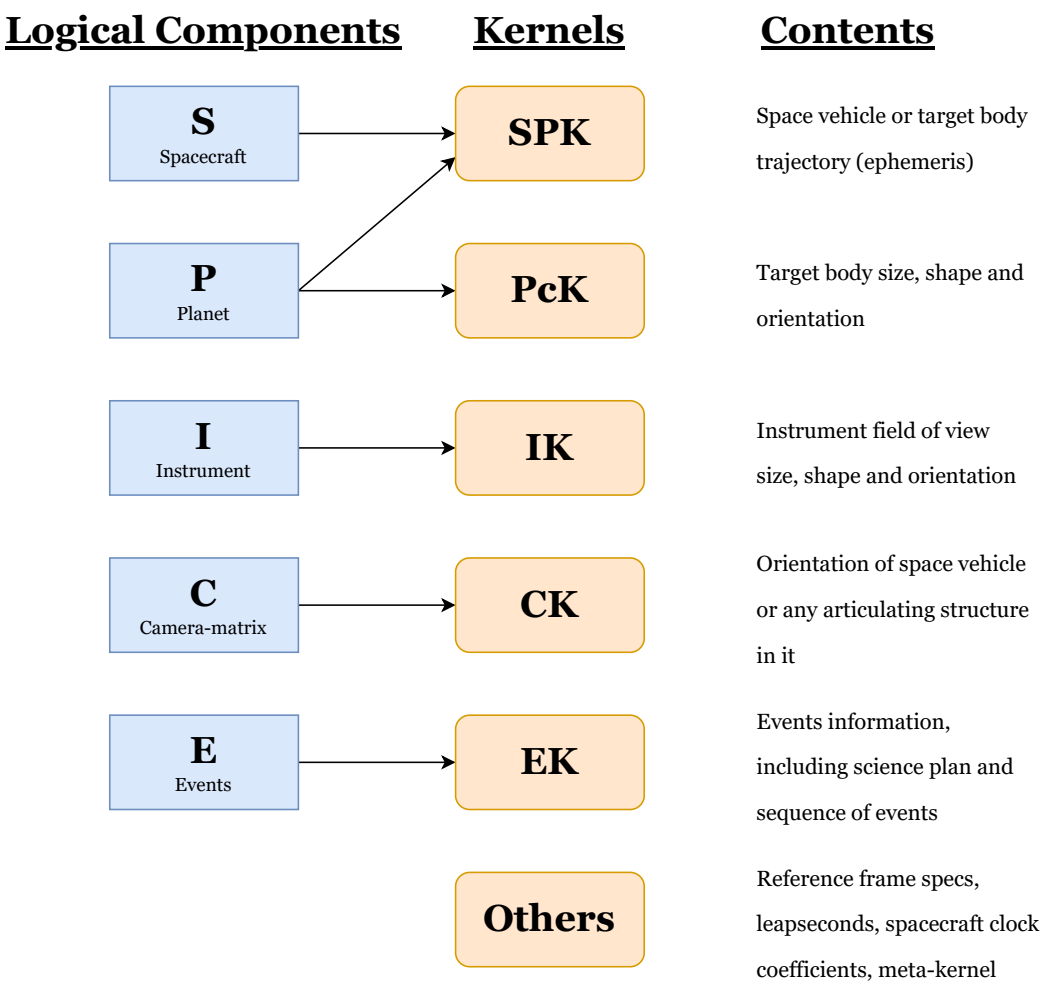


FIGURE 2.2: Explanation of the SPICE architecture [23]

Service is responsible for the creation and distribution of SPICE data for a number of ESA missions, including the Hera mission which is of interest to us [24]. The repository is updated every few months with the most up-to-date SPICE data based on revised estimates and/or new observational data. For this thesis, the Hera SPICE Kernel Dataset version 1.1 dated 09/11/2021 is used. To access the SPICE data provided, a dedicated Julia wrapper for the SPICE toolkit has to be used to convert the data to a usable format. This wrapper is SPICE.jl.

As previously discussed in Chapter 1, the dynamical characterization of the Didymos-Dimorphos binary asteroid system takes place during the first of the six proximity operation phases, namely during the Early Characterization Phase [19]. During this phase, Hera performs hyperbolic arcs at a distance of about 30 km from the barycenter of the two binary asteroids, with the main objective of capturing images to perform a physical and dynamical characterization of the system [19]. Typical orbits followed by Hera during this phase can be seen in Figure 2.3, which has been extracted directly from the SPICE data discussed before.

For the context of this thesis, a time interval corresponding to 5 hyperbolic arcs has

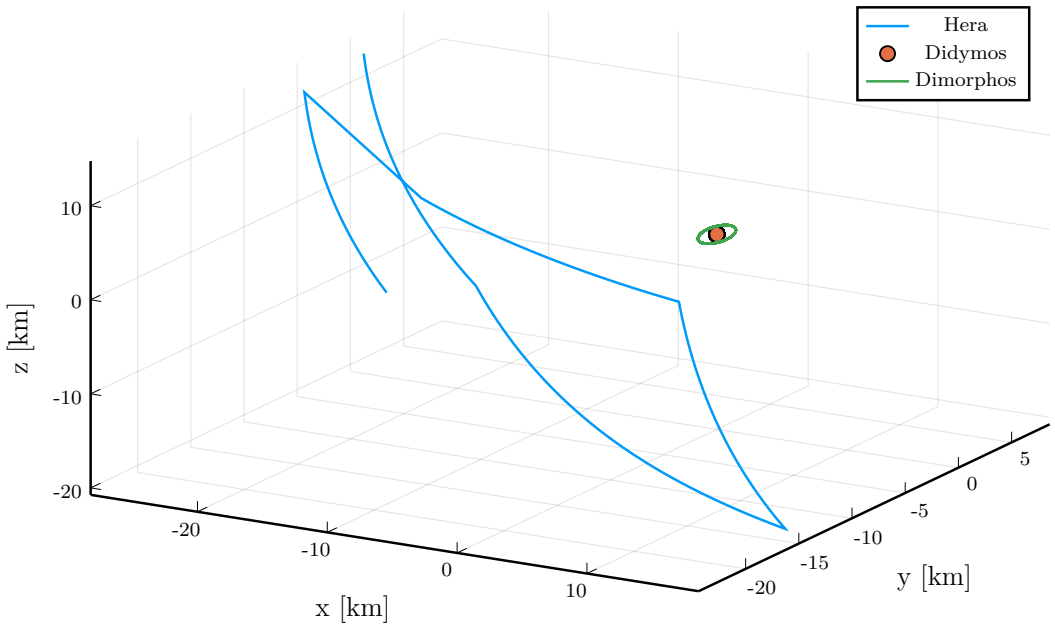


FIGURE 2.3: Five hyperbolic arcs performed by Hera with respect to the system's barycenter over a period of 15 days.

been chosen. The observation time used has been set to achieve this is set to 300h, which is approximately equivalent to 25 orbits of Dimorphos around Didymos. This observation period is equal to 12.5 days, providing a reasonable time frame for the acquisition of pictures. At the worst case scenario, assuming that Hera has to acquire 1000 photos of the two asteroids, approximately 4 photos per hour have to be taken, providing enough time per hour for other operations including orbital maintenance. One also has to note that due to high pointing errors especially towards the end of a hyperbolic arc [25], pointing errors would fluctuate even more for longer observation periods.

Finally, the observation period selected includes four transitions between hyperbolic arcs, which correspond to the worst observational conditions, where the pointing error gets maximized [25]. The corresponding orbit of Hera for the selected time interval during the Early Characterization Phase with respect to the system's barycenter can be seen in Figure 2.4.

2.2.3 Didymos-Dimorphos dynamics

In Section 2.2.2, the orbits of all bodies were considered with respect to the barycenter of the Didymos-Dimorphos system. A local inertial frame is defined (similar to the earth centered inertial frame J2000), with its origin being the Didymos-Dimorphos barycenter, with the latter being obtained from SPICE in the ICRF. It should be noted that all coordinates extracted from SPICE will be reported in this inertial reference frame with the binary asteroid's barycenter as its origin. While the orbit of the Hera spacecraft will be extracted and used directly from SPICE, the orbits of the two

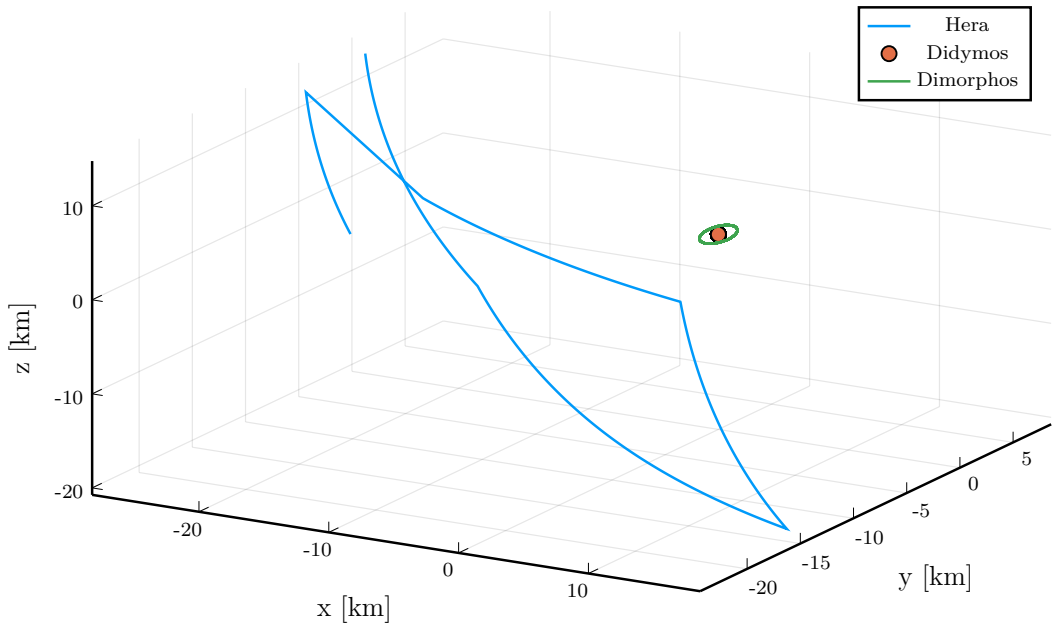


FIGURE 2.4: The hyperbolic arcs which Hera orbits on during the 300h observation period chosen for this thesis.

asteroids will be extracted from a self-developed Runge-Kutta 4 integrator. This will enable testing of several types of orbits and orbital parameters, which are not limited by the one reference case included in the data provided by the ESA SPICE Service. The properties of the system and the integrator will be discussed now.

Since Hera's orbit is predefined from SPICE and does not have to be calculated, the two body problem can be used for the two asteroids. The gravitational force between two bodies is given by:

$$\vec{F}_i = G\frac{m_i m_j}{r_{ij}^3}\vec{r}_{ij} \tag{2.2}$$

While the equation of motion for each body is given by:

$$\ddot{\vec{R}}_i = G\frac{m_j}{r_{ij}^3}\vec{r}_{ij} \tag{2.3}$$

The equation of relative motion will thus be given by:

$$\ddot{\vec{r}} = -G\left(m_1 + m_2\right)\frac{\vec{r}}{r^3} \tag{2.4}$$

Being a small asteroid body, Didymos does not have atmosphere which can cause a perturbation to the motion of Dimorphos due to atmospheric drag. However, the non-spherical shape of the asteroid will cause a perturbation in the gravitational potential. While one of the main goals of the DART mission is to obtain a better understanding of

the gravity environment of the binary asteroid system [26], only perturbations caused by J_2 will be considered in this thesis. This perturbation will cause the variation of orbital elements throughout the period of the orbit and will allow us to model complex situations by working with osculating elements. Considering the effect of J_2, the equation of relative motion which will be used will be:

$$\ddot{\vec{r}} = -G\left(m_{Didymos} + m_{Dimorphos}\right)\frac{\vec{r}}{r^3} + \vec{a_{J_2}} \tag{2.5}$$

The acceleration caused by J_2 can be written in cartesian form as [27]:

$$a_i = -\frac{3J_2\mu R^2_{didymos} r_i}{2r^5}\left(1 - \frac{5r_k^2}{r^2}\right) \tag{2.6}$$

$$a_j = -\frac{3J_2\mu R^2_{didymos} r_j}{2r^5}\left(1 - \frac{5r_k^2}{r^2}\right) \tag{2.7}$$

$$a_k = -\frac{3J_2\mu R^2_{didymos} r_k}{2r^5}\left(3 - \frac{5r_k^2}{r^2}\right) \tag{2.8}$$

where $\mu = G\left(M_{didymos} + M_{dimorphos}\right)$ and i, j and k correspond to x, y and z coordinates. The final form of the equations of motion for each cartesian coordinate to be integrated are:

$$\ddot{x} = -\frac{\mu x}{r^3} - \frac{3J_2\mu R^2_{didymos} r_i}{2r^5}\left(1 - \frac{5r_k^2}{r^2}\right) \tag{2.9}$$

$$\ddot{y} = -\frac{\mu y}{r^3} - \frac{3J_2\mu R^2_{didymos} r_j}{2r^5}\left(1 - \frac{5r_k^2}{r^2}\right) \tag{2.10}$$

$$\ddot{z} = -\frac{\mu z}{r^3} - \frac{3J_2\mu R^2_{didymos} r_k}{2r^5}\left(3 - \frac{5r_k^2}{r^2}\right) \tag{2.11}$$

Equations (2.9) to (2.11) form a system of differential equations which have to be integrated. To integrate the system, a Runge-Kutta method will be used, which achieves the accuracy of a Taylor series approach without requiring the calculation of higher derivatives [28]. A fourth order Runge-Kutta method will be used here (RK4), where the solution is given by the iterative form of:

$$y_{i+1} = yi + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{2.12}$$

The k constants are computed using:

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + \frac{h}{2}, y_i + \frac{k_1 h}{2}) \\
k_3 &= f\left(x_i + \frac{h}{2}, y_i + \frac{k_2 h}{2}\right) \\
k_4 &= f(x_i + h, y_i + k_3 h)
\end{aligned}
\tag{2.13}
$$

In Equation (2.13), h refers to the step size used for the integration, which in this case is the time step used. A time step of $dt = 60$ s will be used. To ensure that the integrator works properly, it has been validated against the PKEPLER routine [27]. The validation procedure has showed that the cartesian positions and velocities obtained using the RK4 integrator used here fully coincides with the cartesian positions and velocities obtained using the PKEPLER routine. The full validation procedure and the results obtained can be found in Appendix A.

2.2.4 Didymos J_2

To compute the J_2 constant of Didymos, the moments of inertia of the asteroid are used. Let's derive the J_2 constant now. The expressions of inertial integrals for a rigid body will be used [29], which provide a relationship for the moment of inertia and the zonal gravitational coefficients, namely:

$$I_{xx} - I_{yy} = -4mr_o^2 C_{22} \tag{2.14}$$

and:

$$I_{yy} - I_{zz} = mr_o^2 (C_{20} + 2C_{22}) \tag{2.15}$$

Combining Eq. (2.14) and Eq. (2.15) yields:

$$-C_{20} = \frac{2I_{zz} - I_{yy} - I_{xx}}{2mr_o^2} \tag{2.16}$$

taking into consideration the fact that $J_2 = -C_{20}$, the final expression for J_2 is obtained to be:

$$J_2 = \frac{2I_{zz} - I_{yy} - I_{xx}}{2mr_o^2} \tag{2.17}$$

Let's assume Didymos to be a triaxial ellipsoid and obtain a mean radius for the body. Assuming $a = 0.416194$ km and $b = 0.418765$ km, a radius value is obtained to be:

$$R = \frac{a+b}{2} = 0.4174795\ km \tag{2.18}$$

For Didymos, a mass of $m_{Didymos} = 5.32 \times 10^{11}$ kg will be used. The moments of inertia are computed to be such that there is an alignment of results with the results computed by the General Use Binary Asteroid Simulator (GUBAS) [30], and are presented in Table 2.1. Combining all of these values together in Equation (2.17), a final value of J_2 is obtained to be:

$$J_2 = 0.012503167534491537 \tag{2.19}$$

The J_2 value from Equation (2.19) will be used in the RK4 propagator to introduce the perturbation in all instances from now on.

TABLE 2.1: Moments of Inertia for Didymos

Moments of Inertia	**Value** [kg km^2]
I_{xx}	31436.196699045453
I_{yy}	32009.61802392827
I_{zz}	32882.223794965415

2.2.5 Orbital Elements & Types of Orbits

To plot results and access the performance of the fitting and optimization procedure which is used to perform orbit determination of Didymos and Dimorphos, orbital elements will be used instead of cartesian positions and velocities. These elements will be **osculating orbital elements**, due to the J_2 perturbation which is included and is discussed in Section 2.2.4. The definitions used by Vallado will be presented here [27]. The first element is the **eccentricity** e, which defines the shape of the orbit. In this case, the eccentricity will be considered to be $e < 1$, since Dimorphos is in either a circular $(e = 0)$ or elliptical orbit $(0 < e < 1)$ around Didymos. It is worth noting that Hera is in *hyperbolic orbit* around the binary asteroid system which means that $e > 1$, but since its orbit is extracted directly from SPICE, the constraints discussed here only apply to the Didymos-Dimorphos system. The **semi-major axis** a is the sum of periapsis and apoapsis distances divided by two. These two elements form the shape and size of the ellipse.

Next, there is the **inclination** i, which refers to the tilt of the orbit plane and is an angle measured between the plane of the orbit and the equator of the main body. Orbits with inclination of 0 $^\circ$ are called equatorial orbits, orbits with inclination of 90° are called polar orbits and orbits with any other value of inclination is called an inclined orbit. The **right ascension of the ascending node** Ω is the angle in the equatorial plane measured positive eastward from the i unit vector to the location of the ascending node, with the ascending node referring to the point on the equatorial

plane at which the orbiting body crosses the equator from south to north (opposite to the descending node).

Finally, there is the **argument of periapsis** ω, which is the angle measured from the ascending node in the direction of satellite motion to the closest point of the orbit (periapsis). The final element is the **true anomaly** ν, which is an angle that determines the orbiting body's current position relative to the location of the periapsis. True anomaly can be replaced by the **mean anomaly** M, which is defined by Kepler's equation, which can be seen in Equation (2.20). In Eq. (2.20), E refers to the eccentric anomaly.

$$M = E - esin(E) = \sqrt{\frac{\mu}{a^3}}(t - T) \tag{2.20}$$

Equation (2.20) is Kepler's equation, with E being the eccentric anomaly. These six orbital elements are enough to describe the orbit of a Dimorphos around Didymos. Measurements or assumptions exist for all orbital elements of Dimorphos' orbit, which are used to formulate the requirements for the DART and HERA missions [26]. These are presented in Table 2.2 and are the main sources of constraints used later on.

TABLE 2.2: Measured or assumed orbital elements for Dimorphos from literature

Orbital Element	Value	Source
a	1.19 ± 0.03 km	Derived from system mass & period [31]
e	< 0.03	Measured [32]
i	$0°$	Assumed [33]
Ω	$40°$	Assumed [17]
ω	Unknown	-
M	$78.9 \pm 1.9°$	Specific epoch fit [17]

The six orbital elements that have been presented so far can define the orbit of an orbiting body and its location except under certain geometric conditions. There is a need to define alternative orbital elements for perfectly circular and equatorial orbits. Although these perfect orbits do not exist, orbits close to these limits ($e \approx 0$ and $i \approx 0°$) will cause problems with computer generated solutions [27]. This is also the case for Dimorphos, since from Table 2.2 it can be determined that the orbit of Dimorphos is very close to being circular ($e < 0.03$) and it is assumed to be equatorial ($i = 0$). Let's explore what has to be done in these special cases, by following the notation used by Vallado [27].

Elliptical Equatorial Orbits

For a non-circular equatorial orbit, no ascending node can be defined. This causes a problem, since the lack of the ascending node means that the argument of periapsis can not be defined. The solution is to define the **true longitude of periapsis** $\tilde{\omega}_{true}$,

which combines the right ascension of the ascending node and the argument of periapsis to remove the ambiguity. $\tilde{\omega}_{true}$ is defined as the angle measured eastward from the vernal equinox to the eccentricity vector and is given by the equation:

$$cos(\tilde{\omega}_{true}) = \frac{\hat{i} \cdot \vec{e}}{|\hat{i}||\vec{e}|} \tag{2.21}$$

Circular Inclined Orbits

In the case of a circular inclined orbit, there is no periapsis which can be used to define the argument of periapsis ω and the mean anomaly M. The solution is to introduce the **argument of latitude** u, which is the angle measured between the ascending node and the orbiting body's position vector in the direction of satellite motion. The argument of latitude is computed using:

$$cos(u) = \frac{\vec{n} \cdot \vec{r}}{|\vec{n}||\vec{r}|} \tag{2.22}$$

Circular Equatorial Orbits

Finally, when dealing with circular equatorial orbits, both the ascending node and the periapsis do not exist, meaning that in this case the right ascension of the ascending node, the argument of periapsis and the mean anomaly cannot be defined. The solution here is to define the **true longitude** λ_{true}, which is the angle measured eastward from the i axis to the position of the satellite and is the same as the right ascension of the satellite that's sometimes used. The definition for λ_{true} is:

$$cos(\lambda_{true}) = \frac{\hat{i} \cdot \vec{r}}{|\hat{i}||\vec{r}|} \tag{2.23}$$

The orbit of Dimorphos

All types of orbits will be examined in the context of this thesis to ensure that the orbit determination procedure developed works and can be implemented for a variety of orbits, but given all of the above and the information for the orbit of Dimorphos obtained from Table 2.2, the most representative orbit for Dimorphos is considered to be a *circular equatorial orbit*.

2.2.6 Transformations between cartesian positions/velocities & orbital elements

Since the propagator integrates an initial state vector composed of cartesian positions and velocities, it also outputs the final orbit in cartesian positions and velocities. But, as discussed earlier, orbital elements will be required to interpret all the results. It would therefore be necessary to have routines which quickly transform between the

two representations. The **RV2COE** and **RV2COE** will be used [27], which have been written in Julia for the purposes of this thesis and are provided in Appendix B.2.

2.3 Image generation process

So far, the orbital motion of Didymos, Dimorphos and the Hera spacecraft has been discussed. The photographing procedure which takes place on-board the Hera spacecraft can now be discussed. In an ideal scenario, the orbit determination procedure which is used for this thesis would be validated with the use of the actual photographs taken by Hera, however since the mission is set to launch in 2024 and reach the binary asteroid system in the spring of 2027, the only way to validate the orbit determination procedure is by simulating the photographs which Hera will take. Let's discuss step by step how the images will be simulated.

2.3.1 Reference frames

In SPICE, all orbital data is stored in kernels. Usually, there are generic kernels like the ones containing information about the sun and its planets which are very common and even come built-in with the SPICE installation and there are mission specific kernels developed for sharing orbital information on a particular mission. ESA SPICE Service reports the coordinates of Hera with respect to the barycenter of the Didymos-Dimorphos binary asteroid system. But it it possible to know where that is? The solution to that is to also include in our analysis the kernels which contains the coordinates of the Didymos-Dimorphos barycenter with respect to the solar system barycenter. Coordinates are reported in the ICRF reference frame. Having loaded the barycenter of Didymos-Dimorphos, Hera's planned orbits can be immediately obtained in cartesian ICRF coordinates and transformations between coordinate systems can also be performed.

The Hera SPICE kernels dataset also includes the positions of Didymos and Dimorphos with respect to the barycenter of the system. However, the data used here are only placeholder data of a circular orbit and cannot be used for the purposes of this analysis. To overcome this, the equations of motion provided in Section 2.2.3 that are integrated using the RK4 integrator developed for this thesis can be very easily reported with respect to the systems barycenter. If $\vec{r}$ is the integrated form of the equation of motion, the orbit of Dimorphos with respect to barycenter will be given by:

$$\vec{R}_{Dimorphos} = \frac{m_{Didymos}}{m_{Didymos} + m_{Dimorphos}} \vec{r} \tag{2.24}$$

while the orbit of Didymos with respect to the system's barycenter will be:

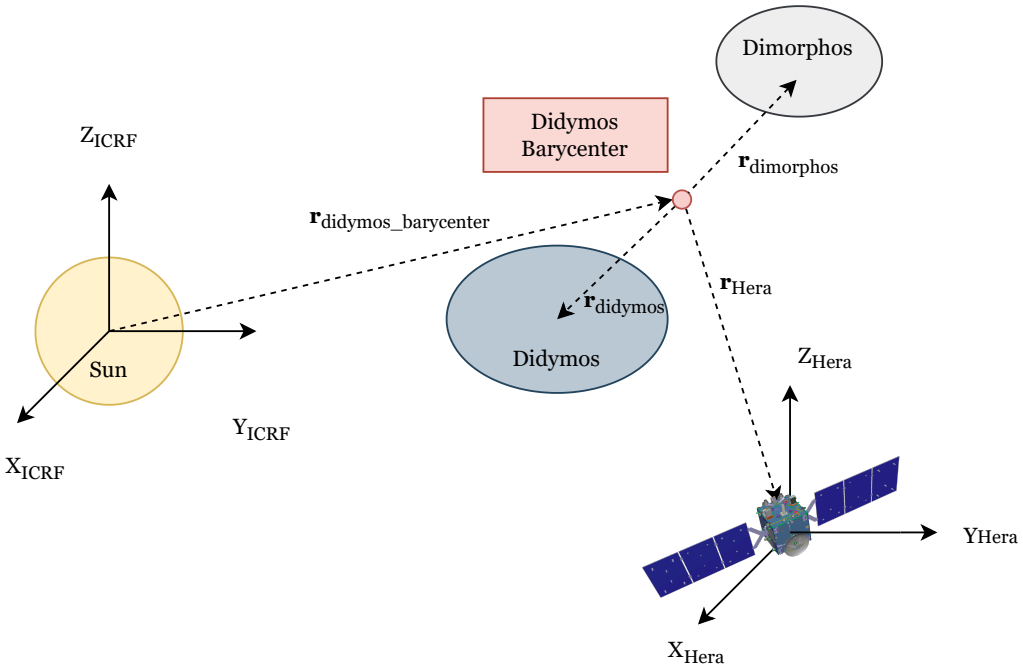


FIGURE 2.5: Getting from the ICRF to the Hera reference frame (definitely not to scale)

$$\vec{R}_{Didymos} = -\frac{m_{Dimorphos}}{m_{Didymos} + m_{Dimorphos}}\vec{r} \tag{2.25}$$

Evidently, there is now a way to directly plug in the output cartesian positions of the integrator to the ICRF frame and all the data of Didymos and Dimorphos generated can now be simply placed around the barycenter of the system. The final result here is that the coordinates of all three bodies (Didymos, Dimorphos and Hera) can be reported in the same reference frame, as seen in Figure 2.5. Now it is only a matter of time of getting to the camera frame of Hera and simulating the photographs. Unfortunately, it is not that simple, due to the lack of camera frame kernels in the dataset used which are described below.

2.3.2 Adding Hera's attitude by rotating the camera frame

The science deck of Hera containing all the scientific instruments, including the Asteroid Framing Camera (AFC), is found on the +Z face of Hera [34]. During the early characterization phase, Hera would be on hyperbolic arcs with an average distance from the system's barycenter of about 30 km and it will point its science deck towards the system to conduct the preliminary physical and dynamical characterization of Didymos [25]. However, this is not reflected in the Hera SPICE kernels dataset used here, since the attitude of the spacecraft has not been yet modeled in the dataset and the spacecraft axes coincide with the ICRF axes. Therefore, manual rotations have to be made to correctly reflect the attitude of the spacecraft. The existing definition of Hera's body frame can be used and rotated in such a manner so that the +Z axis (or to

be clearer the $\vec{e_z}$ unit vector of Hera's body frame) is pointing towards the barycenter of the Didymos-Dimorphos system. Do note that it is assumed that Hera is constantly tracking the barycenter of the system meaning that the $\vec{e_z}$ unit vector constantly points towards the barycenter of Didymos-Dimorphos at all times. Pointing performance errors will be discussed later on, for now let's see how the rotation which is pictured in Figure 2.6 will be achieved.

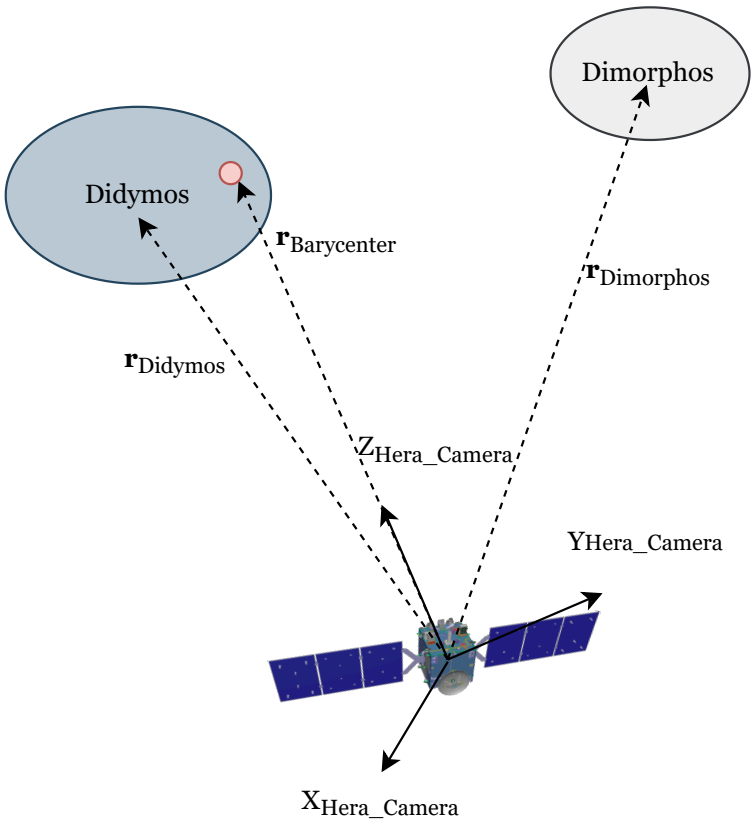


FIGURE 2.6: Definition of Hera's camera frame so that the $\vec{e_z}$ unit vector of the frame is pointing towards the Didymos-Dimorphos system barycenter (image not to scale)

The rotation matrix R has to be computed, which rotates unit vector $\vec{a}$ to $\vec{b}$ (in this case $\vec{a}$ would correspond to $\vec{e_z}$ of Hera's camera frame and $\vec{b}$ would correspond to the position vector $\vec{r}_{barycenter}$ of the binary asteroid's barycenter in the camera reference frame). After first converting vectors $\vec{a}$ and $\vec{b}$, let:

$$d = a \times b \tag{2.26}$$

Furthermore, let:

$$s = ||d|| \tag{2.27}$$

and:

$$c = a \cdot b \tag{2.28}$$

The rotation matrix R will be given by:

$$R = I + [d]_\times + [d]_\times^2 + frac1 - cs^2 \tag{2.29}$$

where $[d]_\times$ is the skew-symmetric cross-product matrix of c:

$$[d]\times \overset{\text{def}}{=} \begin{bmatrix} 0 & -d_3 & d_2 \\ d_3 & 0 & -d_1 \\ -d_2 & d_1 & 0 \end{bmatrix} \tag{2.30}$$

The final fraction in Equation (2.29) can be further simplified as by using the property $sin^2(x) + cos^2(x) = 1$ to get:

$$\frac{1-c}{s^2} = \frac{1-c}{1-c^2} = \frac{1}{1+c} \tag{2.31}$$

so the final equation for the desired rotation matrix will be:

$$R = I + [d]_\times + [d]_\times^2 + \frac{1}{1+c} \tag{2.32}$$

This formula cannot be used for vectors pointing in exactly opposite directions, because in that case $c = -1$ and the last fraction would be undefined. A new orthogonal basis can be formed afterwards for the camera frame. By performing this procedure at every time step t_i, the coordinates of Didymos, Dimorphos and their barycenter are expressed in the Hera camera reference frame which includes the attitude of the spacecraft, meaning that the Hera camera frame is constantly tracking the barycenter of the system (do note that pointing errors will be discussed later on).

2.3.3 Getting from world coordinates to pixel coordinates

At this point, all coordinates of Didymos and Dimorphos are expressed in Hera's camera reference frame. If these coordinates can be successfully transformed into pixel coordinates, the desired goal will be achieved, since pixel coordinates will correspond to the actual photographs captured by Hera. Let us now explain how the pixel coordinates for the two bodies will be obtained.

Firstly, a **basic perspective projection** will be used to project the asteroid points (world coordinates) into the image plane of the camera (camera coordinates). The process is highlighted in Figure 2.7. Let P be a point of Didymos to be mapped on the image plane, with world coordinates (camera frame) $P = (X, Y, Z)$. By defining

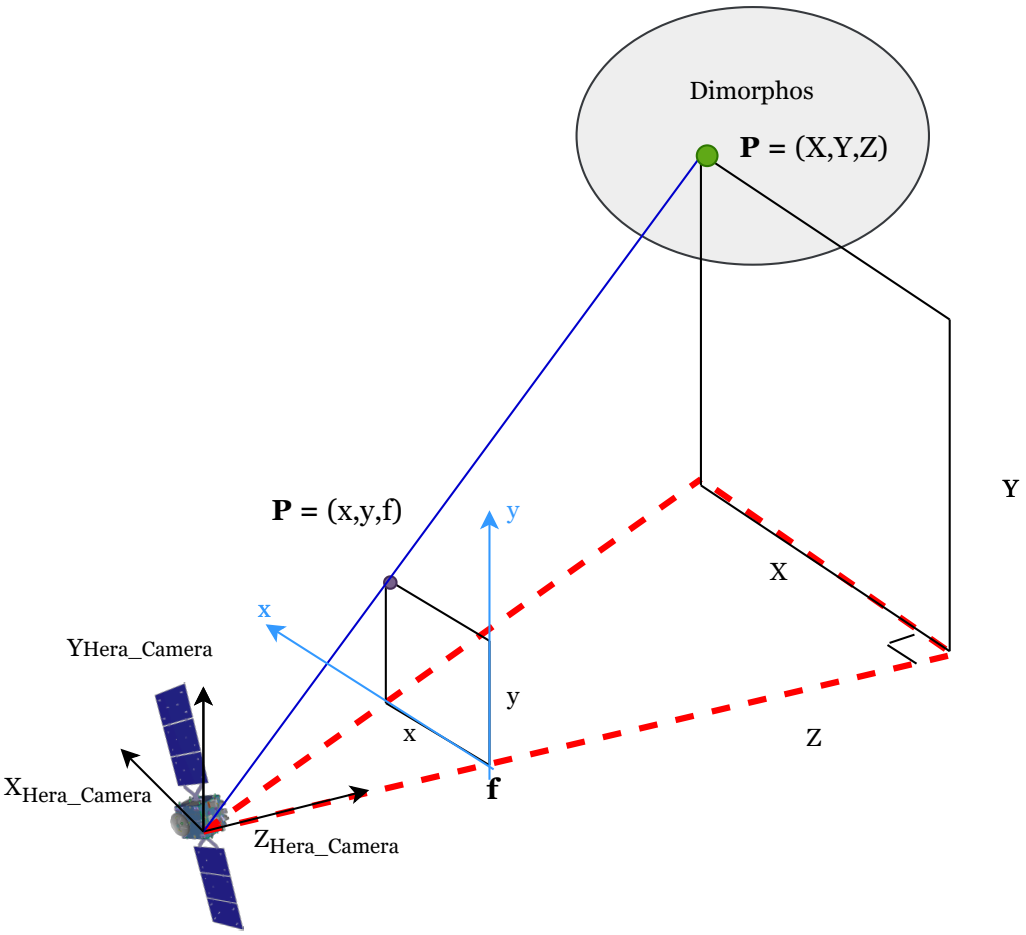


FIGURE 2.7: Basic perspective projection used to simulate the photographs taken by Hera on the image plane

the image plane to be at the focal distance f of the camera (meaning that the image plane would be at a distance f and parallel to the XY plane of the camera frame), the similar triangles rule can be used to project the point P on the image plane, following the trigonometry presented in Figure 2.8. The projection of the point P would be the image point p which, according to the similar triangles rule, will have coordinates $p = (x, y, f)$ with:

$$x = f\frac{X}{Z} \tag{2.33}$$

and

$$y = f\frac{Y}{Z} \tag{2.34}$$

It is apparent that to complete this projection, the properties of Hera's imaging system are required. Hera uses an instrument called the Asteroid Framing Camera (AFC), developed by JenaOptronik [35]. The properties of Hera's asteroid framing camera can be found in Table 2.3.

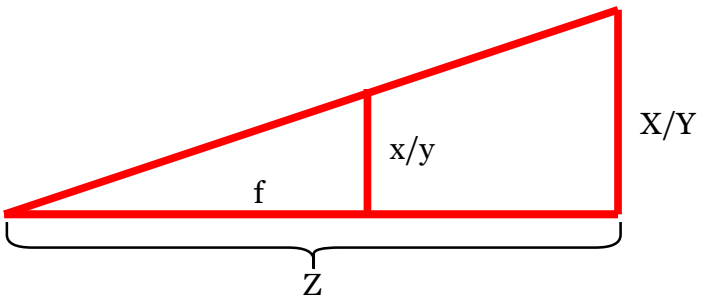


FIGURE 2.8: Similar triangles to obtain the coordinates on the image plane

TABLE 2.3: Hera's Asteroid Framing Camera (AFC) properties [35]

Parameter	Value
Field of View	5.5 °
Focal Length	10.6 mm
Image size	1020 × 1020 pixels
Angular Resolution	94.1 μrad/pixel
Spectral Range	420-850 nm

Using the focal length value from Table 2.3, the projections of the objects can be expressed on the image plane. Using the Field of View, SPICE provides a built-in tool to calculate the four boundary vectors of the camera, which in practice represent the four corners of the photograph. This concept is illustrated in Figure 2.9. Expressing the boundary vectors in the updated Hera camera frame which includes the correct attitude, the boundary vectors can be used to get the four boundary points on the image plane. Using these four points, which define the limits of the photograph, it can be immediately determine whether a point projected on the image plane belongs inside the photograph or not.

Furthermore, from Table 2.3, it can be observed that an image captured using the Asteroid Framing Camera will be 1020 by 1020 pixels wide. Thus, the boundary points on the image plane can be assigned to represent the corner pixels, corresponding to pixel coordinates (1, 1), (1, 1020), (1020, 1) and (1020, 1020) respectively. After that, 1020 bins of equal width can be assigned for each pixel. Each bin contains a subset of the image plane coordinates range between the two boundary points in the x and y dimension. This means that by having the coordinates of a point on the image plane, it can be immediately assigned to its corresponding pixel coordinates using this procedure or reject it if it lies outside the boundaries of the photograph. This process is summarized in Figure 2.10. In this thesis, only the geometric center of each asteroid will be used (with the introduction of a centroiding error which will be discussed later on), which means that at each photograph taken by Hera there is one point on the pixel grid for Didymos and one point for Dimorphos.

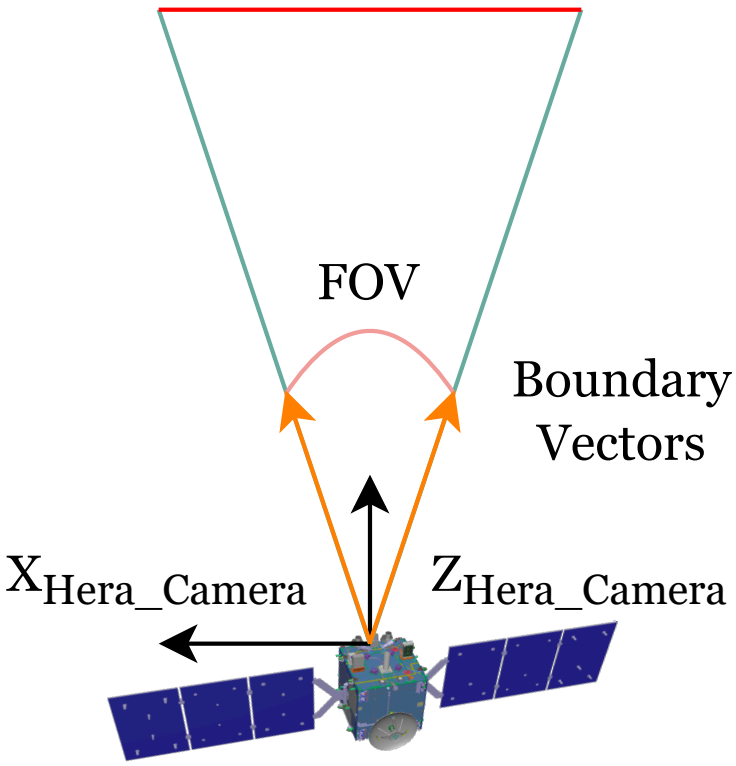


FIGURE 2.9: Field of View and boundary vectors definition

2.3.4 Types of photographs and working with them

As hinted above, due to several errors introduced in the photographs which will be explained in Section 2.4, not all objects might be visible in the photographs. Thus, it is important to distinguish between the different cases which are encountered when generating the data and how they can be dealt with in the simulations. There are four different cases which have to be considered.

1. Photographs containing both Didymos and Dimorphos
2. Photographs containing only Didymos
3. Photographs containing only Dimorphos
4. Photographs containing neither Didymos nor Dimorphos

Photographs containing both Didymos and Dimorphos

Under normal conditions, both Didymos and Dimorphos are to be observed in a photograph, since the Hera mission requirements use good estimates for the geometrical shape of the orbit and included the appropriate imaging equipment to capture both asteroids [17]. A typical photo in this case can be found in Figure 2.11. As

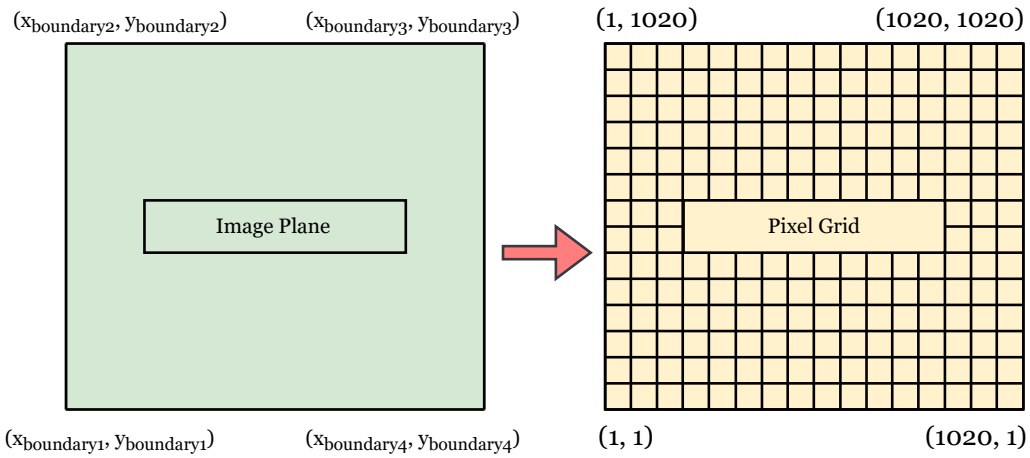


FIGURE 2.10: Assignment of coordinates from the image plane to the pixel grid

previously discussed, only the geometric center of each asteroid is to be observed in the simulated photographs.

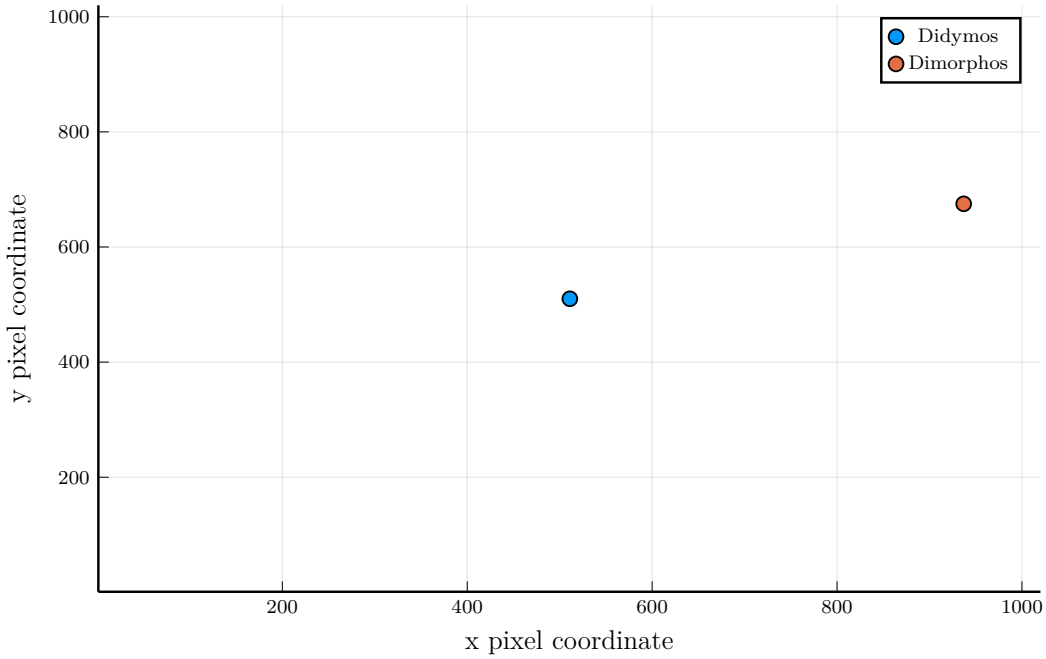


FIGURE 2.11: A photograph containing both Didymos and Dimorphos

Photographs containing only Didymos

Depending on the distance of Hera from the two asteroids or if the values for the orbital elements of Dimorphos were wrongly measured, Dimorphos might not be visible in a photograph (or better yet: its geometric center might not be visible). This can also happen due to a higher pointing performance error of Hera's attitude control system. This case is displayed in Figure 2.12.

Photographs containing only Dimorphos

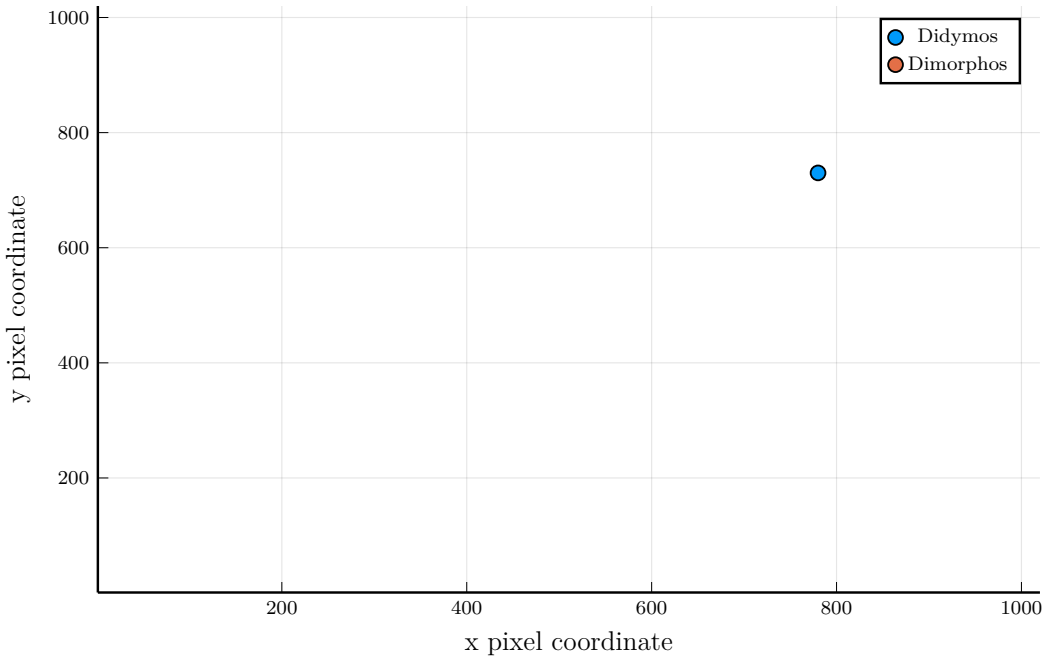


FIGURE 2.12: A photograph containing only Didymos

This type of photograph is considered extremely rare to obtain, given the fact that Hera constantly points its camera towards the binary asteroid's barycenter, close to where Didymos is located. Nevertheless, due to a higher pointing performance error towards the end of a hyperbolic arc [25], it is possible that a photograph is obtained which only contains Dimorphos, as illustrated in Figure 2.13. This higher pointing performance error is attributed to the fact that Hera has to point its thrusters to the correct direction to perform the necessary orbital maneuver required to switch to the next hyperbolic arc [25].

Photographs containing neither Didymos nor Dimorphos

Similar to the previous case, this type of photograph is considered extremely rare and can only happen near the end of a hyperbolic arc where Hera has to change its attitude to perform the required maneuver to move to the next hyperbolic arc, which can generate a photograph with neither Didymos nor Dimorphos visible, as seen in Figure 2.14.

For the orbit determination procedure implemented in this thesis (which aims to determine the orbit of Dimorphos around Didymos), photographs containing **both** Didymos and Dimorphos are used. This is because the relative motion of the objects is important: photographs containing only Didymos lack the object of which the orbit has to be determined, while photographs containing only Dimorphos fail to provide information about the position of the body in its orbit. Guessing the orbit is out of the question, so photographs containing neither asteroid are immediately discarded. In Julia, this is implemented using the `missing` type of data values.

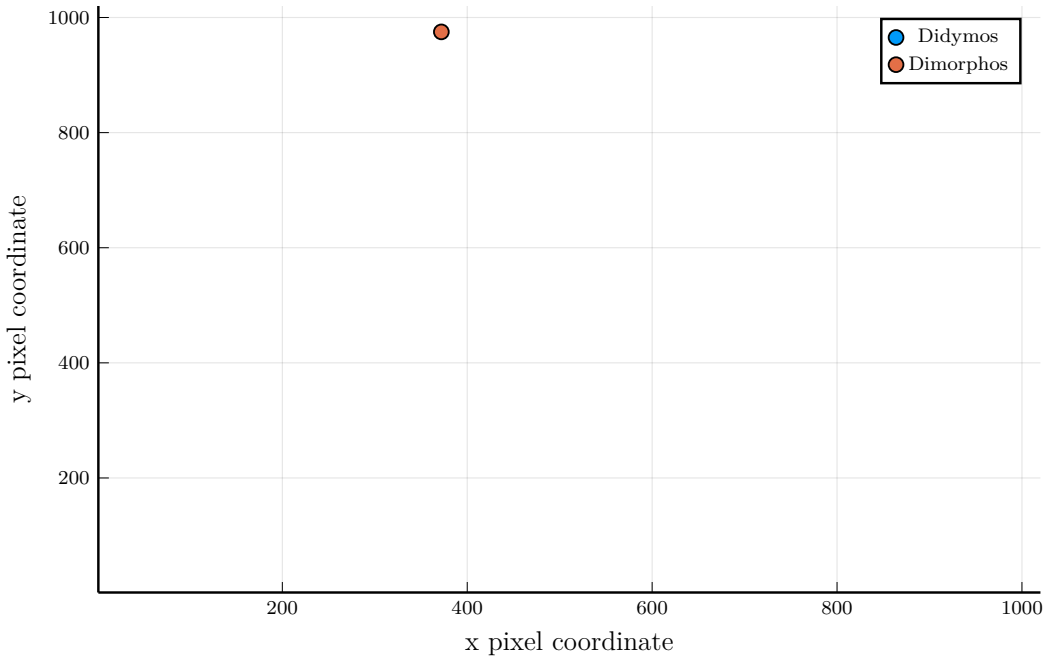


FIGURE 2.13: A photograph containing only Dimorphos

If Didymos or Dimorphos, based on their position on the image plane, lie outside the pixel grid of the photograph, they are immediately assigned `missing` values as their x and y pixel coordinates. Furthermore, if only one asteroid is assigned pixel coordinates inside the photograph and the other one is not, then the asteroid inside the photograph also has its pixel coordinates replaced with `missing` values, for the reasons discussed above. This process is performed automatically during the generation of the photographs and the ones who contain only `missing` values can be immediately disregarded.

2.4 Errors considered in the images

The image generation process so far has been pretty straight forward: Extract Hera's orbit from SPICE, propagate the orbit of Dimorphos around Didymos, express the relevant coordinates in the camera frame and use a projection method to obtain the final photograph. However, reality is a bit more complicated. Different sources of error impact the quality of the images in different ways. Optical effects like blurry images which can definitely occur are not considered here. The errors introduced in this analysis have the end effect of changing the x and y coordinate of Didymos and/or Dimorphos in a given image. This effect was previously identified and it can lead to the extreme scenario of neither asteroid been visible in the image generated. The point here is fairly simple: Since these sources of error will inevitably exist in the real photos captured by Hera, it is important to include them in our analysis as well to see if the orbit determination procedure developed can still work with these errors. Let's briefly introduce the different types of errors before going into greater detail:

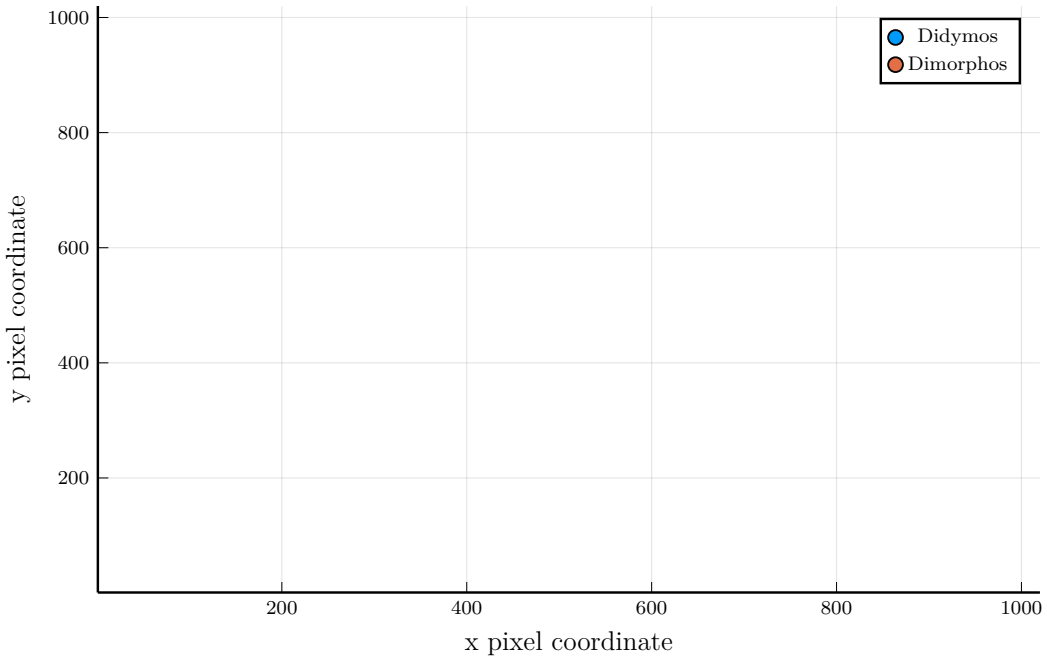


FIGURE 2.14: A photograph containing neither Didymos nor Dimorphos

1. **Position errors**, which include:
   (a) Error on the position of Hera.
   (b) Error on the position of the barycenter of Didymos-Dimorphos.
2. **Pointing errors**, introduced due to the attitude control system of Hera.
3. **Centroid pixel error**, caused by attempting to locate the geometric center of an object.
4. **Randomly dropped images**, images which lack some information and can't be used for the orbit determination process.
5. **Error on the mass** of Didymos and Dimorphos.

Each error mentioned above will be examined in great detail.

Position errors

When talking about the errors that exist for a satellite's orbit, there are two main ones to be considered:

1. Along track error
2. Cross track error

Both of these errors are illustrated in Figure 2.15. The *along track error* refers to the fact that the actual position of the satellite might be a little bit ahead or behind of the predicted solution generated by a computer. This is a particular problem when

dealing with ground station passes of satellites in Low Earth Orbit, since it is required to minimize the along track error to establish a good connection with the satellite [27]. On the other hand, the cross track error is a deviation from the predicted position, which can happen due to the imperfect knowledge of one of the satellite's orbital elements for example. Later on, during the orbit determination procedure, the algorithm will attempt to fit different orbits which have high values of both along track and cross track errors: these errors are gradually reduced until the computer finds the best possible fit.

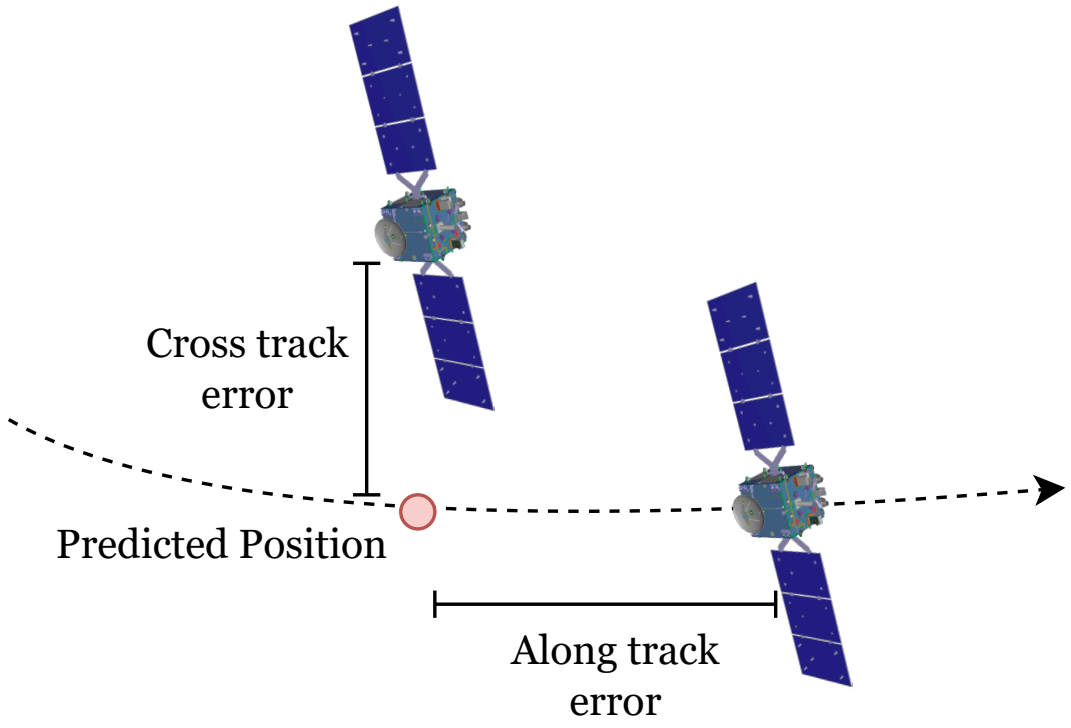


FIGURE 2.15: Definition of along and cross track errors for Hera

Hera uses a X-band communication system for uplink and downlink with ground stations on Earth and a non-stop communication link can lead to very good position estimates. For this reason, the error considered here will be in the order of 10 m. A normal distribution is created with mean $\mu = 0$ m and standard deviation of $\sigma = 10$ m. At every time step in the image generation process, Hera's position is logged using SPICE and three random values are generated from the normal distribution and applied to each cartesian coordinate x, y and z of Hera.

For the barycenter of the Didymos-Dimorphos system the tracking performed by Hera is the main concern. It is assumed that the Asteroid Framing Camera of Hera is always tracking the barycenter of the system. However, the barycenter is not a physical point which can be tracked so in reality Hera will probably be tracking Didymos instead. Even if there was a way to accurately track a landmark on Didymos which would correspond to the position of the barycenter, an error would definitely be introduced as seen in Figure 2.16. To accomodate for this in our analysis, a normal

distribution is created with mean $\mu = 0$ m and standard deviation of $\sigma = 30$ m. At every time step in the image generation process, prior to the transformations required to express the positions of objects in Hera's camera frame, three random values are generated from the normal distribution and are applied to each cartesian coordinate x, y and z of the system's barycenter.

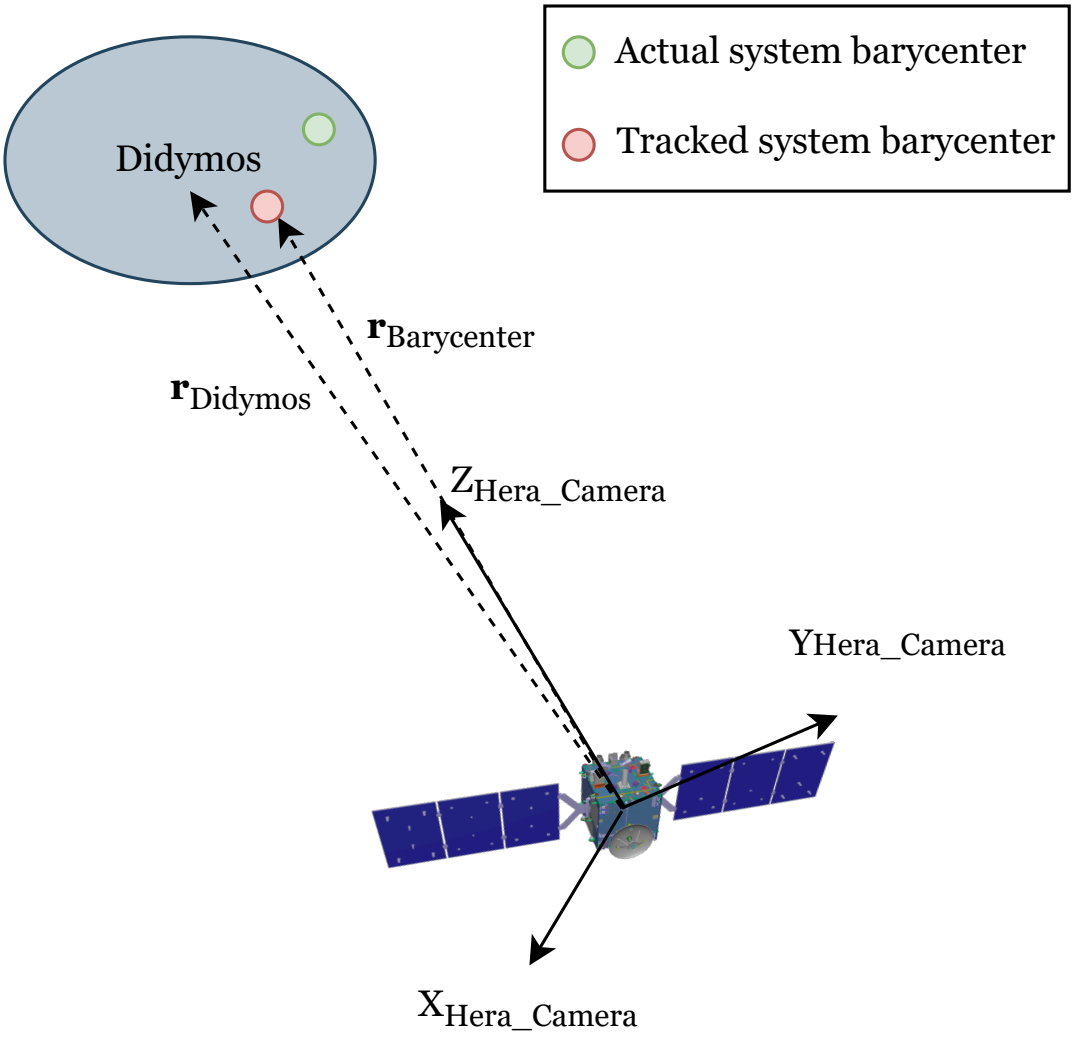


FIGURE 2.16: Definition of the barycenter position error

2.4.1 Pointing error

To point its camera towards the two asteroids, Hera uses an attitude control system. This system is capable of autonomous operations in-orbit [25], but since no system is perfect, a small error in pointing performance is to be expected, a fact which is illustrated in Figure 2.17. In this analysis a conservative pointing performance of $\pm 1^\circ$ from the nominal pointing target of Hera is considered (which is the Didymos-Dimorphos barycenter). To simulate this in our code, a normal distribution is created with mean $\mu = 0$ m and standard deviation of $\sigma = 1\ ^\circ$. One of the three axis is then randomly selected, in which the pointing error will be applied to. Using a randomly

generated value θ from the normal distribution, a rotation matrix R is generated. Depending on the axis the rotation matrix will be:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2.35}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.36}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.37}$$

The generated rotation matrix is used to define the coordinates of all objects accounting for the pointing error of Hera's attitude control system.

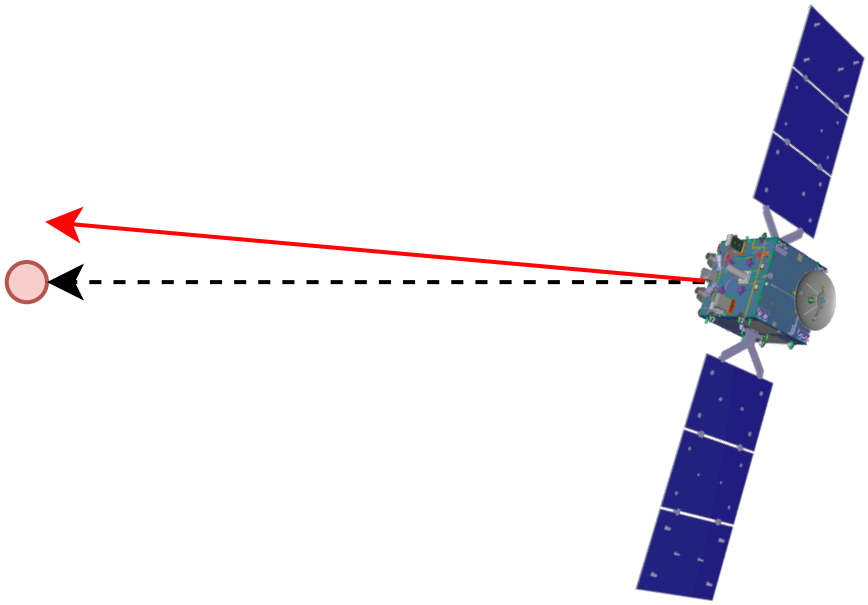


FIGURE 2.17: Predicted pointing vs actual pointing (red) caused by the attitude control system

2.4.2 Centroid pixel error

To better understand the centroid pixel error, an illustration has to be used. In Figure 2.18, Didymos is being illuminated by the sun. The photons from the sun however are only able to illuminate the area of the asteroid which face the sun and have direct line of sight with it, while the remaining areas remain dark. This causes a

problem for small bodies in the solar system which, unlike larger bodies like planets which are almost spherical, have non-canonical shapes. It is therefore challenging to identify the geometric center of the object being observed using a photograph, since it is only partially illuminated.

An example of this can be seen in Figure 2.19, which shows a photo of the comet 67P/Churyumov-Gerasimenko taken by the Rosetta spacecraft's navigation camera on 2014 at a distance of about 30 km [36]. From this picture, it can be determined that it is hard to identify the geometric center of the comet. Hera also uses a thermal camera to correctly identify the shape of the asteroids and locate their geometric centers [25], but in this analysis only the images captured in the visible part of the spectrum will be considred. Therefore, a centroid pixel error will be introduced now.

It is assumed for this analysis that the centroid pixel error will have a maximum value of 4 pixels. At each time step, four random numbers are generated uniformly in the interval of $[-4,\ 4]$. After the coordinates of Didymos and Dimorphos are calculated in the pixel grid, one of the four random numbers is added to the x and y pixel coordinates of Didymos and Dimorphos respectively to account for the centroid pixel error.

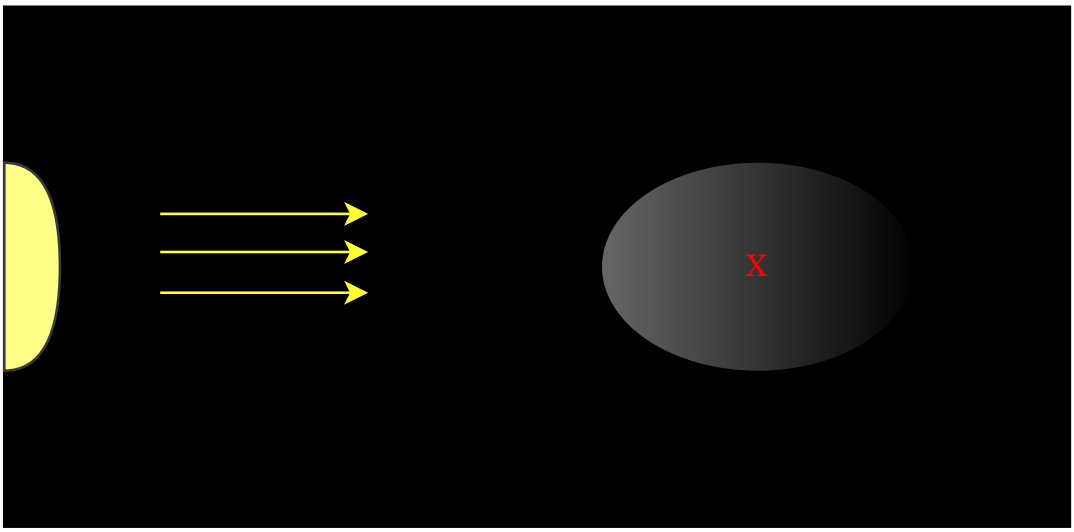


FIGURE 2.18: Illustration of the centroid pixel error (not to scale)

2.4.3 Randomly dropped images

In Figure 2.7 an important assumption is made when generating the photographs using the basic perspective projection: that the distance between the camera and the object is known. However, the blue line in Figure 2.7 shows that all points which belong on the blue line will be projected at the same point on the image plane. To determine the distance from the asteroids, Hera uses a Planetary ALTimeter (PALT) which works as a LIDAR system [25]. However, if for whatever reason the LIDAR does not take a measurement while taking a photograph or the measurement is

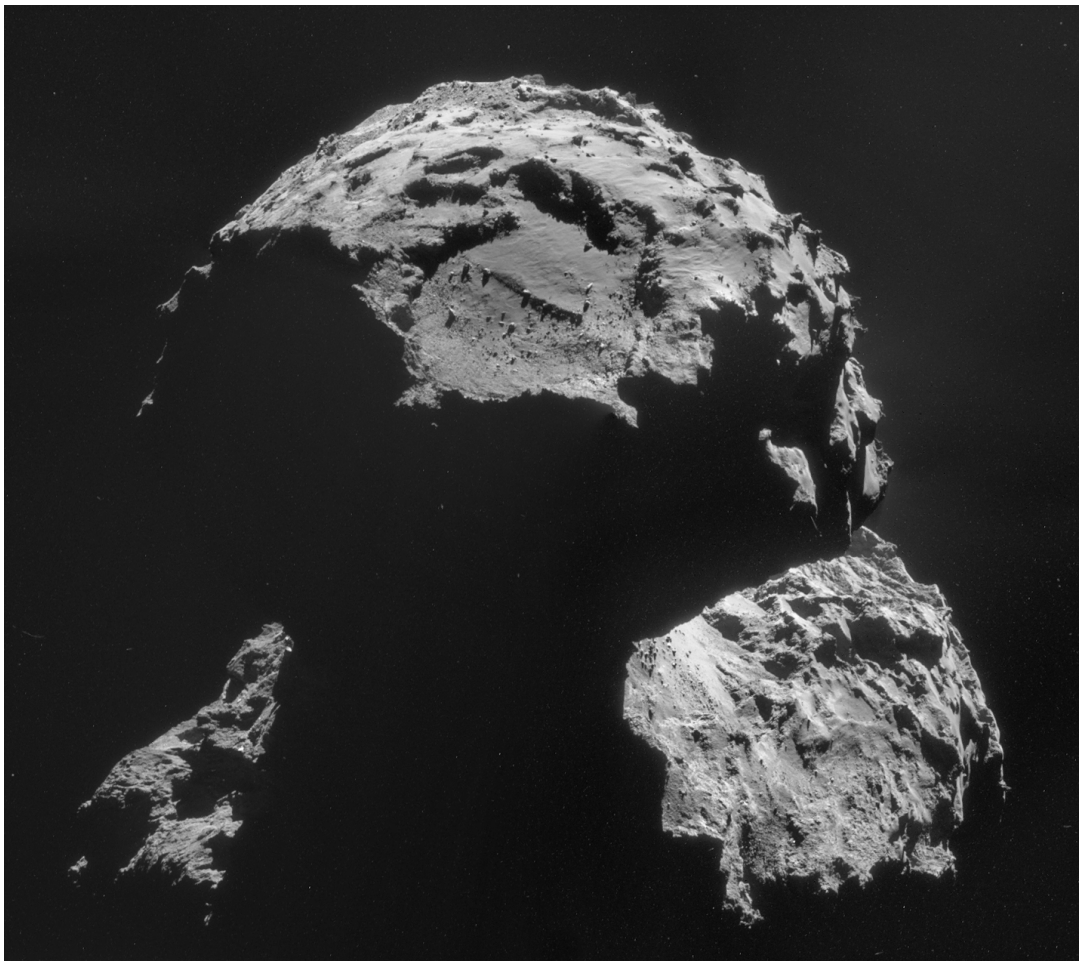


FIGURE 2.19: Image of the 67P/Churyumov-Gerasimenko comet captured by Rosetta [36]

inaccurate, the image cannot be used afterwards for the orbit determination procedure. Thus, we speak of randomly dropped images. To simulate these types of images, 4% of the total images are randomly selected and dropped using the `missing` data type described earlier.

2.4.4 Mass error

Existing measurements for Didymos and Dimorphos report their masses as the mass of the binary asteroid system and include an error of $5.55 \times 10^{11} \pm 0.42 \times 10^{11}$ kg [17]. The mass of each object is then derived by using density and shape assumptions. Hera will be able to determine the mass of each body independently [17], which means that more accurate more measurements will exist to be used in conjuction with the captured photographs of the asteroids. To simulate this in our analysis, a randomly generated percentage error uniformly created from [-2%, 2%] will be added at the start of the analysis, so that the integration sequence includes the mass values with the added mass error for Didymos and Dimorphos.

2.4.5 Summary of all errors

Table 2.4 provides a summary of all errors considered in the image generation process for quick reference.

TABLE 2.4: Summary of errors included in the image generation process

Error	Value
Hera position error	Randomly generated for each axis from Normal distribution ($\mu = 0.0$, $\sigma = 10$) [m]
Barycenter position error	Randomly generated for each axis from Normal distribution ($\mu = 0.0$, $\sigma = 30$) [m]
Pointing error	Randomly generated for one axis from Normal distribution ($\mu = 0.0$, $\sigma = 1$) [°]
Centroid error	Randomly generated for each pixel coordinate from Uniform distribution $[-4, 4]$ [pixels]
Mass error	Randomly generated for each asteroid from Uniform distribution $[-2, 2]$ [%]
Randomly dropped	Randomly removed 4% of total images

Chapter 3

Orbit determination procedure

In Chapter 2, the image generation process used to simulate the images captured by Hera was examined. By completing this process, a set of images containing Didymos and Dimorphos has been obtained (or neither object, as explained in Section 2.4). These images contain the geometric centers of both objects stored as pixel coordinates. In addition, the timestamp at which a image was generated is known. The question now is the following: Using the aforementioned set of images and the information contained therein, can we estimate the initial state vector used to generate the original data set (containing the orbit of Dimorphos are Didymos) and how accurate will this estimate be? In other words, an orbit determination problem is now being faced: details on the approach used to solve it will be provided in this chapter. In short, the process used to perform orbit determination is the following:

1. Reduce the problem to an optimization problem.
2. Decide on the optimization strategy to be used for locating the minimum.
3. Develop the code to generate images for a given initial state vector.
4. Using the optimization strategy locate the desired initial state vector which best corresponds to the observation data set.
5. Extract performance metrics to evaluate the solution given by the algorithm.

3.1 From orbit determination to optimization

As discussed previously, a set of images has been obtained which corresponds to the ones taken by Hera and includes the pixel coordinates of Didymos and Dimorphos. From now on, this set of images will be referred to as the **observed images**. Any additional set of images generated to serve the orbit determination procedure from now on will be referred to as the **predicted images**. If a set of predicted images is generated which would correspond to each image from the observed image (meaning that they would both refer to the same timestamp), an evaluation of the predicted set of images can then take place. Let us also remind readers once again that the orbit

determination problem discussed attempts to identify the orbit of Dimorphos around Didymos.

In Figure 3.1 a small set of images is to be observed, for which the observed pixel coordinates of Didymos and Dimorphos can be seen. Generating a prediction for the orbit of Dimorphos around Didymos, the predicted points can be plotted in the same figure. The process for generating the predicted images will be discussed later, for now let us only focus on how the predicted orbit in Figure 3.1 can be evaluated in terms of accuracy. Considering the fact that the information which can be immediately obtained from the image is the pixel coordinates of each object and the timestamp of each photograph, the decision was made to use the **sum of squared residuals (SSR)** to evaluate the fitted orbit. Assuming that n images are to be evaluated, the sum of squared residuals in this case would be defined as:

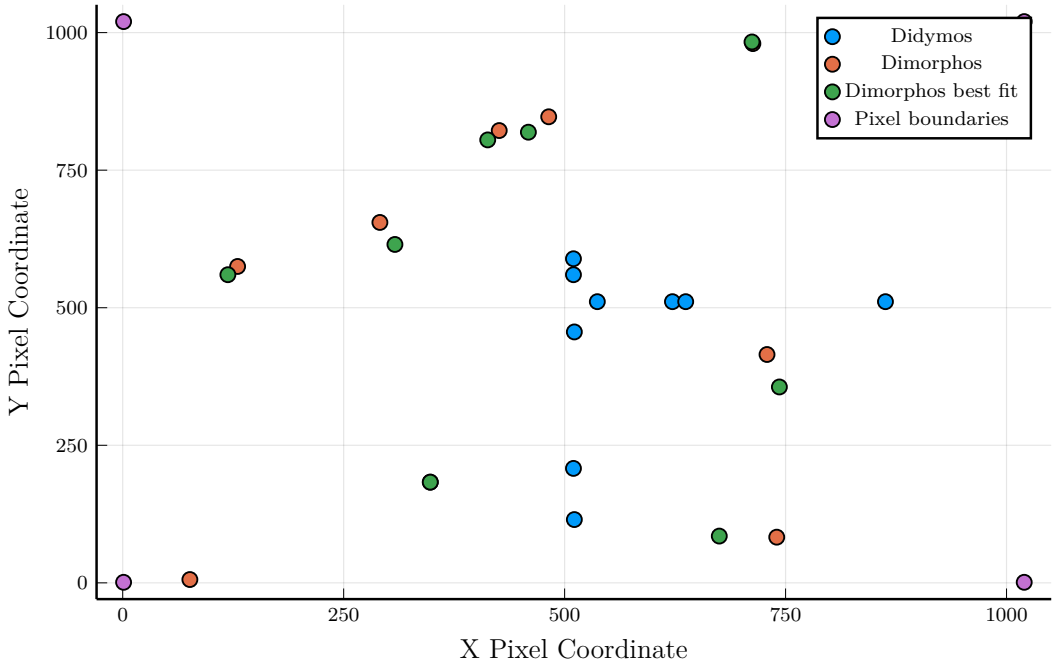


FIGURE 3.1: A set of 10 observed images and a corresponding set of 10 predicted images placed together in the pixel grid

$$SSR_x = \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{3.1}$$

and

$$SSR_y = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.2}$$

for each coordinate separately with x_i and y_i indicating the observed values in pixel coordinates while $\hat{x}_i$ and $\hat{y}_i$ refer to the predicted values in pixel coordinates. Since we are trying to determine the orbit of Dimorphos around Didymos, only the sum

of squared residuals for the x and y coordinates of Dimorphos will be considered. Finally, the two summations can be combined to obtain a final value of the sum of squared residuals to be:

$$SSR_{final} = SSR_x + SSR_y = \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 + \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.3}$$

From Equation (3.3), it can be immediately observed that if we were able to predict the observed pixel coordinates then the value of SSR_{final} would be zero, indicating that we have identified the observed orbit. For reasons that will become later in Section 3.3, $SSR_{final} = 0$ is impossible to be achieved due to the errors that were included in this analysis. However, the best prediction will minimize the value of SSR_{final} by definition. Let us now define a function f which would take as input the initial state vector of a set of predicted images and the predicted gravitational parameter value μ of the system, which in the general case would correspond to $f(a_o, e_o, i_o, \Omega_o, \omega_o, M_o, \mu_{predicted})$. Here, orbital elements are used for the initial state vector, but one may as well use the cartesian positions and velocities as well if desired. This function, which is provided in detail in Appendix B.3, will:

1. Generate images using the process described in Section 3.3 at the relevant timestamps by propagating the initial state vector.
2. Compute the sum of squared residuals between the observed and the predicted pixel coordinates for Dimorphos for each of the generated images.
3. Return the value of the sum of squared residuals SSR_{final}.

Or we can define f in simpler terms to be:

$$f(a_o, e_o, i_o, \Omega_o, \omega_o, M_o, \mu_{predicted}) = SSR_{final} \tag{3.4}$$

The benefit of this function is apparent. Effectively, using the function f, we have taken a predicted initial state vector and a gravitational parameter prediction and evaluated against the observed images to obtain its sum of squared residuals value. As previously discussed, the predicted orbit which best corresponds to the observed images has to have the minimum value of SSR_{final} between all orbits examined. A **cost function** has thus been defined for which the minimum has to be located. The orbit determination problem has now been transformed into an optimization problem: by identifying the initial state vector and the gravitational parameter which will minimize the cost function, the orbit which best fits the observed images will also be identified.

3.2 Metaheuristic used for optimization

Several techniques exist which attempt to locate the minimum of a function of several variables, which is the case of f which was defined above. These techniques are generally split into those which require the derivative of a function and are called gradient or descent methods and those which do not require the derivative of a function and are called nongradient or direct methods [28]. Due to the complexity of f and the numerical integrations involved when propagating to obtain the SSR_{final} value, a derivative is difficult to be obtained for our case. In addition, based on literature, constraints on the initial state vector values for the orbital elements can be placed, as it was previously shown in Table 2.2. These constraints either originate from observations of the Didymos-Dimorphos system or are logically imposed (for example the mean anomaly M can be constrained to be between 0° and 360°). All of these constraints are summarized in Table 3.1. Note that these constraints may be modified later on, mainly for the purpose of validating the optimization procedure for a variety of orbits.

TABLE 3.1: Imposed constraints for Dimorphos' orbit using data from Table 2.2

Orbital Element	Constraint Imposed
a	$[1160.0, 1220.0]$ [m]
e	$(0, 0.03]$ []
i	$[0, 360)$ [°]
Ω	$[0, 360)$ [°]
ω	$[0, 360)$ [°]
M	$[0, 360)$ [°]

We are thus limiting ourselves to nongradient optimization methods which support the addition of constraints. The most traditional approach to solve these types of problems, especially for one or two variables, would be a direct or brute force method. The random search method is a classical example of a brute force approach and will continuously evaluate the function at randomly selected values of independent variables [28]. If a sufficient number of iterations is completed, the minimum we are looking for will eventually be located. The problem with this approach is that it is very computationally expensive: we are basically randomly selecting values in hope of finding a minimum. By conducting a quick experiment, this approach would require a million iterations at a minimum to produce any decent result and would complete in a few days of computational time. Keep in mind that we will have to perform several runs to validate the orbit determination process, meaning that the computational time would quickly rise to unacceptable levels. Thus, brute force methods were quickly dismissed and metaheuristics were explored as an alternative.

A **Metaheuristic** refers to an algorithm which is often inspired by nature and is designed to solve complex optimization problems [37]. These types of algorithms belong

to the heuristics algorithms and form the basis for the field of artificial intelligence [38]. Notable examples of metaheuristic algorithms include:

1. **Ant Colony Optimization**, which links optimization to biological ants based on the observation of their behavior: when walking on routes from the next to a source of food, ants don't simply find a random route but attempt to locate quite a short route, thus enabling their behavior to be used for optimization problems [37, 39].
2. **Evolutionary Computation**, which is inspired by Darwinian evolution, with the algorithm generating a new generation at each iteration by using certain characteristics from the previous generation. Mutations at each generation are explored and characterized by their fitness, meaning that characteristics which improve the fitness are the ones selected for the creation of the next generation [37, 40].
3. **Particle Swarm Optimization**, which is based on the idea of simulating the flight of bird flocks. A set of particles is placed in the search space of an optimization function and computes the value at their assigned points. Then, each particle determines a move through the search space by combining the history of its own current and best locations with those of other particles. The particle swarm moves similarly to a flock of birds collectively foraging for food, having a high probability to locate a local optimum [37, 41].

For the purposes of this thesis, the **Evolutionary Centers Algorithm (ECA)** will be used, which is also classified as a metaheuristic. The principal behind the Evolutionary Centers Algorithm is utilizing the center of mass definition in order to create new directions for moving the worst elements of a population to better regions of the search space [42]. Let us briefly describe the algorithm used, using the original paper [42]. For each solution x_i in the population $P = \{x_1, x_2, ..., x_n\}$ of solutions, a subset U is selected so that $U \subset P$. From this subset, the center of mass c is obtained. Based on a randomly chosen solution $u_r \in U$ and the already generated center of mass c, a new direction is generated in order to create a new solution h_i, following the principle of:

$$h_i = x_i + \eta_i(c_i - u_r) \tag{3.5}$$

where η_i is a constant usually selected between (0, 2] and c_i is given by:

$$c_i = \frac{1}{W} \sum_{u \in U} f(u) \cdot u, \; W = \sum_{u \in U} f(u) \tag{3.6}$$

Where the function f calculates the equivalent mass and returns it as a constant for each solution. The implementation provided in the *Metaheuristics.jl* package [43] will be used for the optimization of our function. For setting the different parameters

required by the algorithm, the recommendations provided by the original authors of the Evolutionary Centers Algorithm will be used [42].

3.3 Generating predicted images for the cost function

A cost function of 7 variables has been defined so far and a way to optimize this cost function has been introduced. The question is now how exactly the set of predicted images to be evaluated against the observed ones will be generated. Although the answer to that question is fairly simple, since the image generation procedure described in Chapter 2 will be used, let us briefly explain the rationale behind this decision and some points of interest to be made.

An argument can be immediately presented stating that if the predicted images are generated using the same technique as the observed images, the initial state vector used for the observed images will be fairly easily to locate. This would happen since once the initial state vector is located, the cost function in this case will be $f(a_o, e_o, i_o, \Omega_o, \omega_o, M_o, \mu_{predicted}) = SSR_{final} = 0$. The algorithm would have then converged because, as pointed out before, the sum of squared error for the pixel coordinates would be zero.

The reason behind the fact that the value of f can never be zero lies on the errors included in the analysis. To generate the **observed** set of images, several sources of error were added in the analysis including position errors, pointing errors and centroid pixel errors among others. These types of errors attempt to simulate the real world environment in which positions and pointing performances are known with a certain degree of accuracy. If, for example, a spacecraft can achieve a pointing performance error of 1° there is no point investigating what happens below that 1° and there is certainly no way for us to know the exact pointing error of Hera just by looking at the observed images *a posteriori*. Given that, for the **predicted** set of images, the exact position and centroid pixel errors used in the observed set of images cannot be used. At the same time, the error on the mass of Didymos and Dimorphos is also not considered, instead we attempt to see if we can estimate the combined system mass using the optimization procedure. For randomly dropped images we can simply ignore the relevant pictures and not generated a predicted one, since we would have nothing to compare it with.

For the pointing error however, a different strategy can be used. It was previously explained that there are cases where Didymos is not centered at the picture taken by Hera due to a pointing performance error which is randomly generated from a normal distribution. In case both objects remain within the field of vision of the camera, the image is retained. We also note that there is no way of knowing the exact pointing error which caused the shift of both asteroids in pixel coordinates. The following strategy can be devised: if the position of Didymos is shifted on the pixel grid, this shift can be measured in pixel coordinates to be Δx and Δy in each

axis. This Δx and Δy can then be addded to the predicted images and the evaluation can occur afterwards. Since the perspective has potentially changed however, a new source of error is added, with this error being far greater than the ones caused by the lack of knowledge of the position of the bodies and most importantly the centroid pixel error.

Considering all of the above, it should be immediately obvious that the optimization algorithm will never be able to generate an initial state vector using the predicted images which will cause the sum of squared residuals to be zero. This can potentially translate into no or very slow convergence, which will be overcome later on by imposing a maximum number of iterations to be performed by the Evolutionary Centers Algorithm.

3.4 Performance metrics used to evaluate results

The cost function uses the sum of squared residuals to evaluate an initial state vector which is used to generate a set of predicted images. But as previously described, a value for the sum of squared residuals can never be zero, thus using the cost function value as a performance metric can only intuitively assess if we are "close" or "far" from the initial state vector used for the observed images. Different performance metrics have to be used to evaluate the performance of the optimization procedure. If we were dealing with classical keplerian elements, the percentage error could be directly computed and used to evaluate, which is defined as:

$$\%\ error = \frac{|observed\ value - predicted\ value|}{observed\ value} \tag{3.7}$$

Percentage error is used to evaluate the fitted value of the gravitational parameter. For the orbital elements however, the effects of J_2 are considered in this analysis, which means that rather than dealing with classical keplerian elements we are dealing with *osculating elements*. This means that the orbital elements vary over time and do not have a constant value. To evaluate how close a predicted time series is to the observed one, the mean absolute percentage error (MAPE) can be used, which is defined as:

$$\text{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right| \tag{3.8}$$

where A_t refers to the observed value and F_t to the predicted one, with n being the total number of observations. Assuming that the initial state vector of the observed images is known, strictly for the purposes of evaluating the initial state vector of the predicted images, an independent integration of the equations of motion can be performed to generate a time series of values where each element is described as a function of time. These two time series can then be used to compute the mean absolute

percentage error and evaluate the performance of the optimization procedure. The code used to compute the percentage errors and the mean absolute percentage errors can be found in Appendix B.5.

3.5 Testing the optimization procedure for different orbits

Let us first test the orbit determination procedure which uses the optimization of a cost function for a few different types of orbits. For the orbit of Dimorphos around Didymos, the following will be evaluated:

1. A **Normal Elliptical Orbit**
2. An **Elliptical Equatorial Orbit**
3. A **Circular Inclined Orbit**
4. A **Circular Equatorial Orbit**

Previously, we have seen that based on the constraints imposed for the orbital elements of Dimorphos' orbit in Table 2.2, the orbit of Dimorphos is considered to be a circular equatorial orbit. Nevertheless, it is worth testing if the orbit determination procedure works with different types of orbits. The time interval described in Section 2.2.2 was used for the generation of images. After conducting several runs, the maximum number of iterations was set to 100, since this number proved to be sufficient to produce adequate results and the algorithm appears to show convergence behavior.

3.5.1 Normal Elliptical Orbit

A normal elliptical orbit can have higher values of eccentricity with $0 < e < 1$. The constrains used by the algorithm have to be updated to reflect that and can be seen in Table 3.2.

TABLE 3.2: Constraints used by ECA for a normal elliptical orbit of Dimorphos around Didymos

Parameter	Constraint Imposed
a	$[1160.0, 1220.0]$ [m]
e	$(0, 0.99]$ [ ]
i	$[0, 360)$ [°]
Ω	$[0, 360)$ [°]
ω	$[0, 360)$ [°]
M	$[0, 360)$ [°]
μ	$[0.8\mu_{predicted},\ 1.2\mu_{predicted}]$ [$m^3\,s^{-2}$]

From the total of **1000 observed images** to be generated, only **821 were usable**. The algorithm run for **732.2890** s and performed **3466 function calls**. Convergence of the algorithm can be seen in Figure 3.2. Figure 3.3 to Figure 3.8 show the values of the

orbital elements as a function of time for both the observed and the predicted images, while Table 3.3 compares the orbital elements and the gravitational parameter using the mean absolute percentage error and the percentage error respectively.

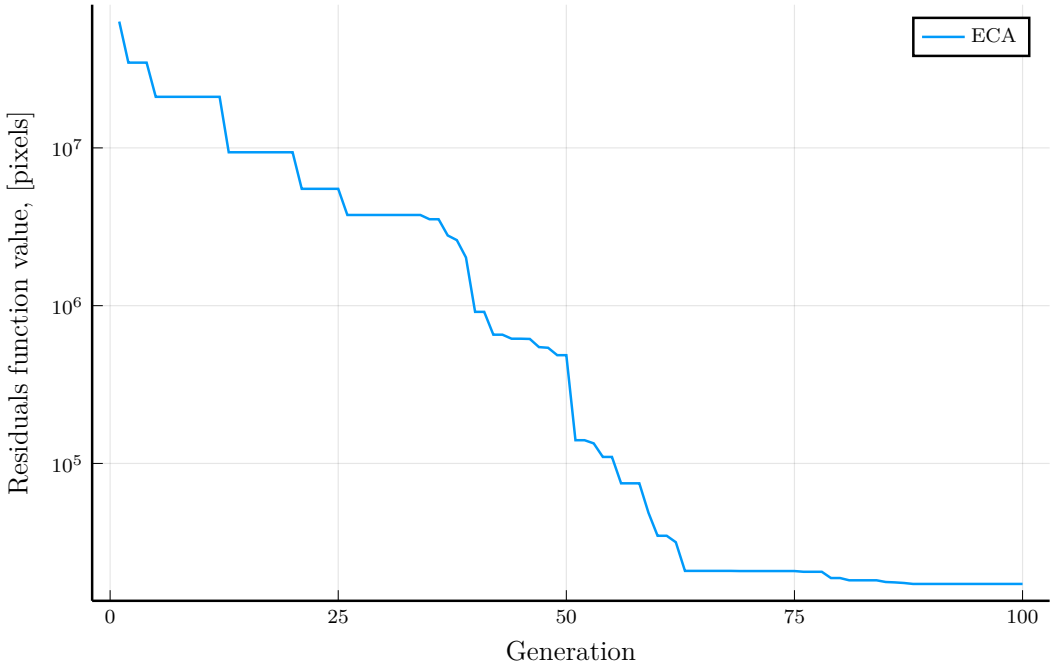


FIGURE 3.2: Error at each iteration of ECA for a normal elliptical orbit

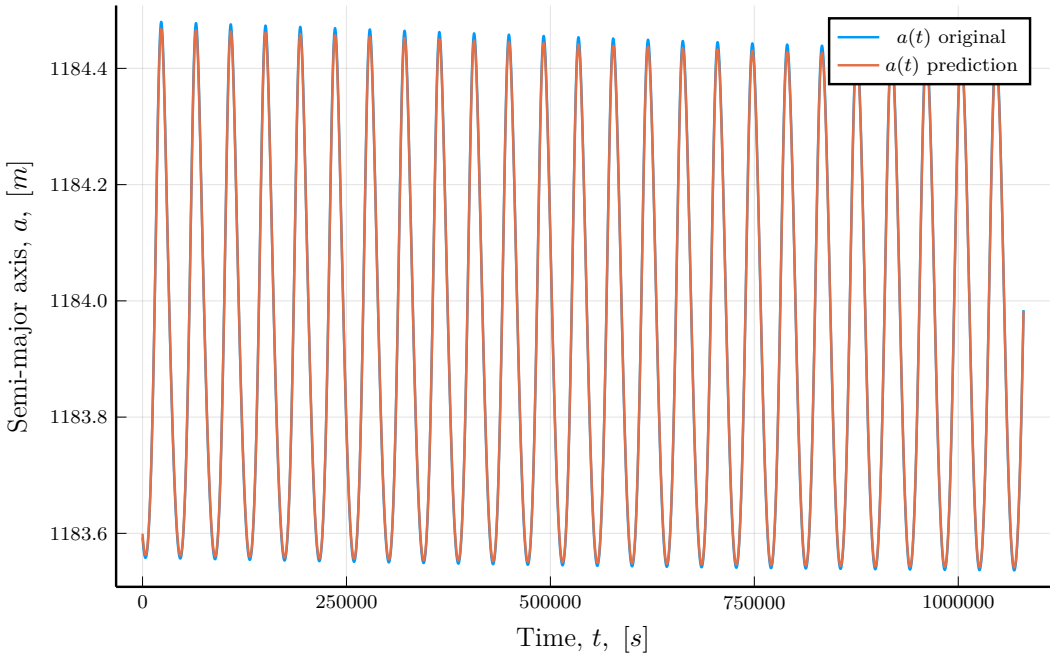


FIGURE 3.3: Original and predicted values of the semi-major axis for a normal elliptical orbit

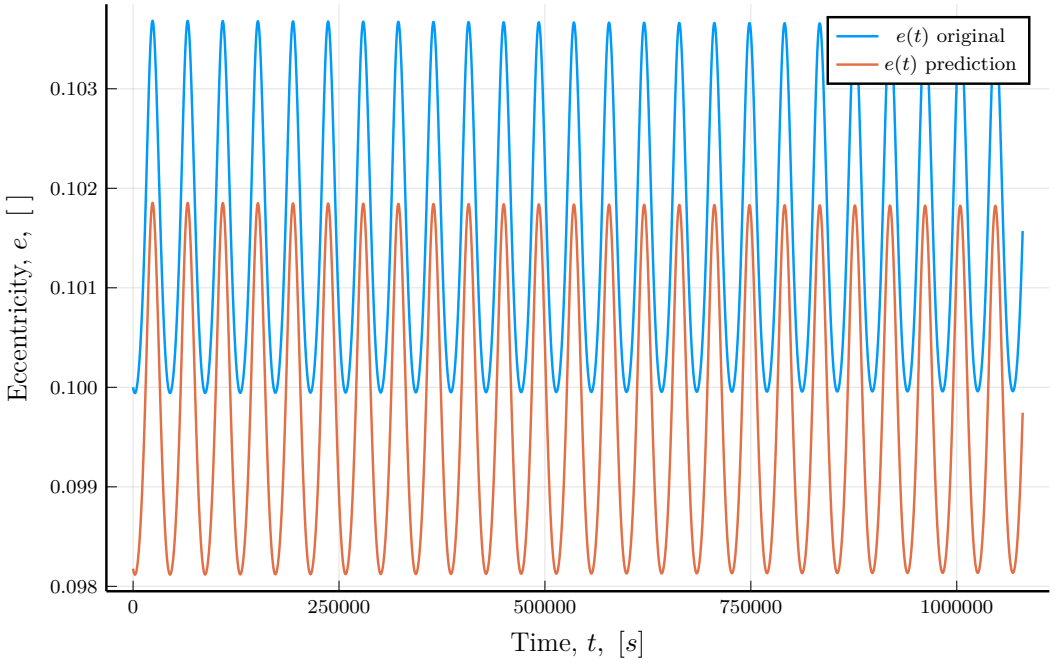


FIGURE 3.4: Original and predicted values of the eccentricity for a normal elliptical orbit

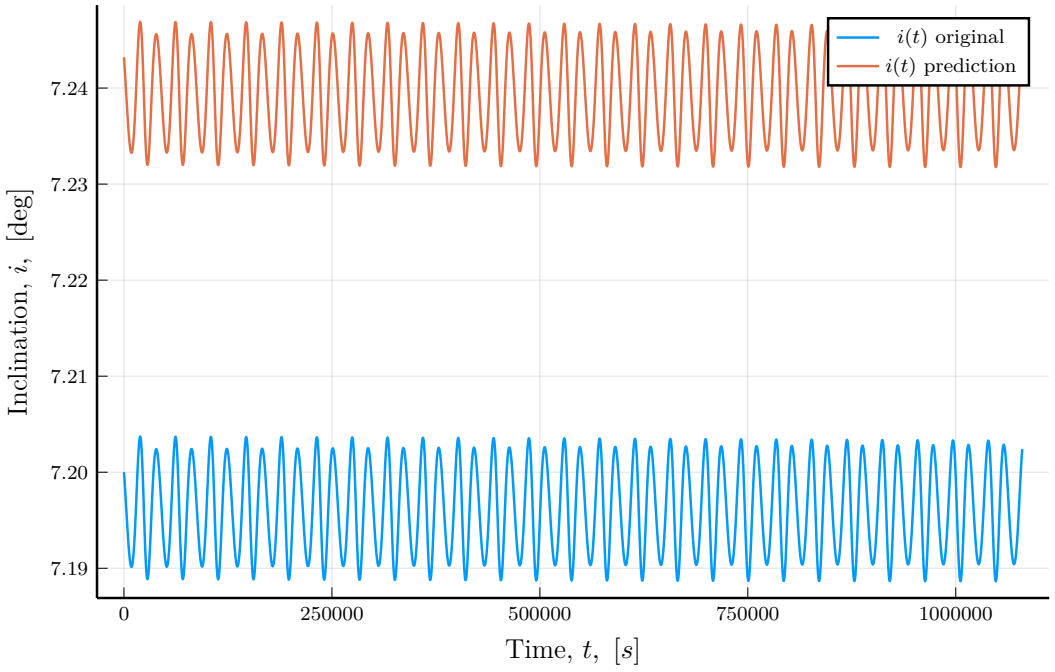


FIGURE 3.5: Original and predicted values of the inclination for a normal elliptical orbit

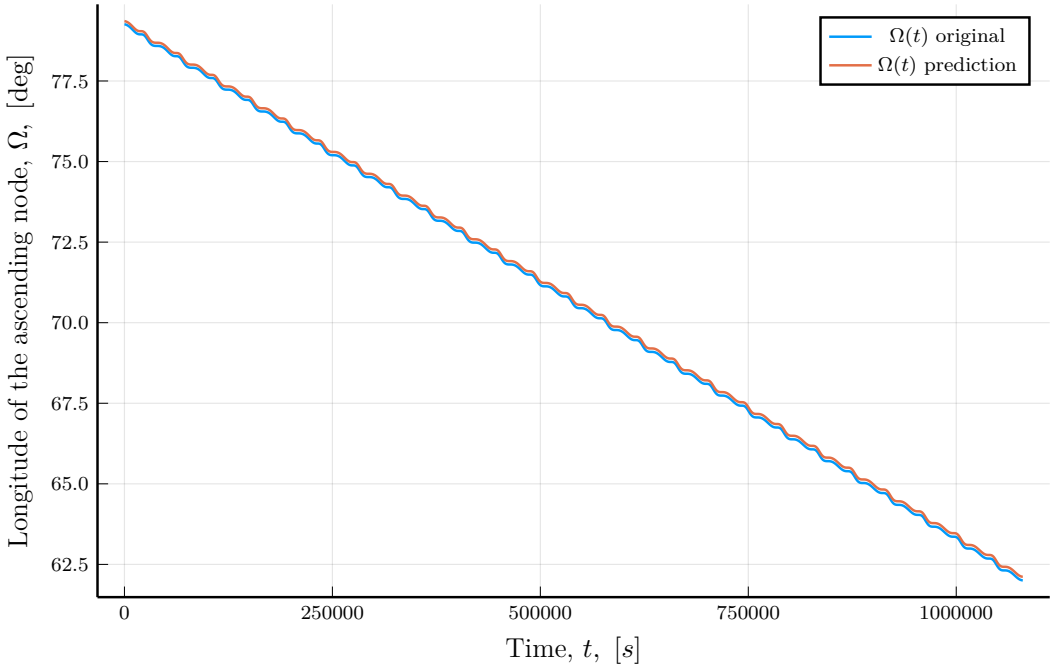


FIGURE 3.6: Original and predicted values of the longitude of the ascending node for a normal elliptical orbit

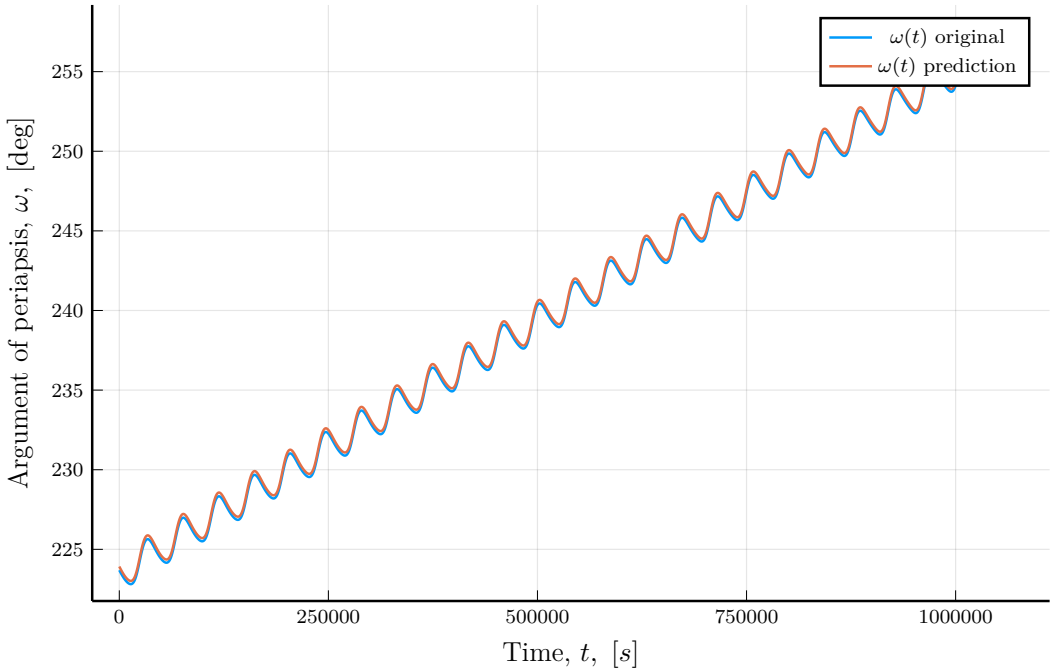


FIGURE 3.7: Original and predicted values of the argument of periapsis for a normal elliptical orbit

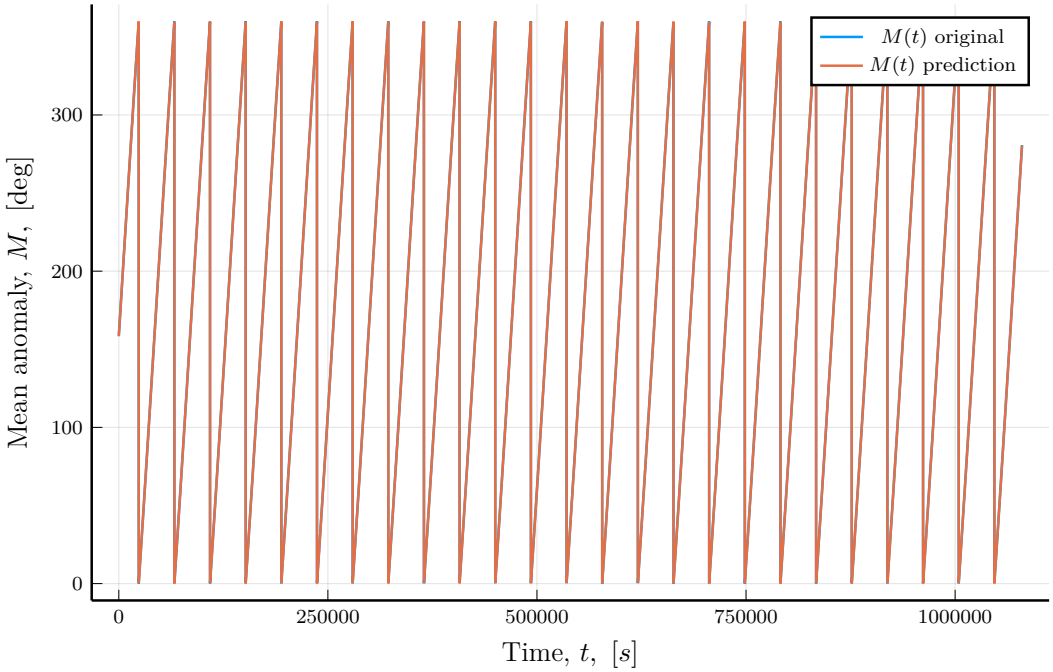


FIGURE 3.8: Original and predicted values of the mean anomaly for a normal elliptical orbit

TABLE 3.3: Original vs Predicted orbital elements and percentage errors for a normal elliptical orbit

Parameter	Original	Predicted	% error
a [m]	1183.593	1183.5992337593261	0.001483
e [ ]	0.1	0.09817795090994062	0.855586
i [°]	7.2	7.243202619949296	0.600338
Ω [°]	79.26	79.3582716351964	0.239957
ω [°]	223.69	223.92098406947352	0.093613
M [°]	162.39	162.06404802954347	0.182845
μ [$m^3\,s^{-2}$]	35.9607749040428	36.0835236863273	0.341341

3.5.2 Elliptical Equatorial Orbit

An elliptical equatorial orbit can have higher values of eccentricity with $0 < e < 1$ while the inclination is zero or very close to being zero $i \approx 0$. Remember that in this case the longitude of the ascending node cannot be defined, since no ascending node exists. The true longitude of periapsis $\tilde{\omega}_{true}$ is used instead, which combines Ω and ω. The constrains used by the algorithm have to be updated to reflect that and can be seen in Table 3.4.

TABLE 3.4: Constraints used by ECA for an elliptical equatorial orbit of Dimorphos around Didymos

Parameter	Constraint Imposed
a	$[1160.0, 1220.0]$ [m]
e	$(0, 0.99]$ [ ]
i	$[0, 0.1)$ [°]
$\tilde{\omega}_{true}$	$[0, 360)$ [°]
M	$[0, 360)$ [°]
μ	$[0.8\mu_{predicted}, 1.2\mu_{predicted}]$ [$m^3\,s^{-2}$]

From the total of **1000 observed images** to be generated, only **814 were usable**. The algorithm run for **709.5210** s and performed **3466 function calls**. Convergence of the algorithm can be seen in Figure 3.9. Figure 3.10 to Figure 3.14 show the values of the orbital elements as a function of time for both the observed and the predicted images, while Table 3.5 compares the orbital elements and the gravitational parameter using the mean absolute percentage error and the percentage error respectively.

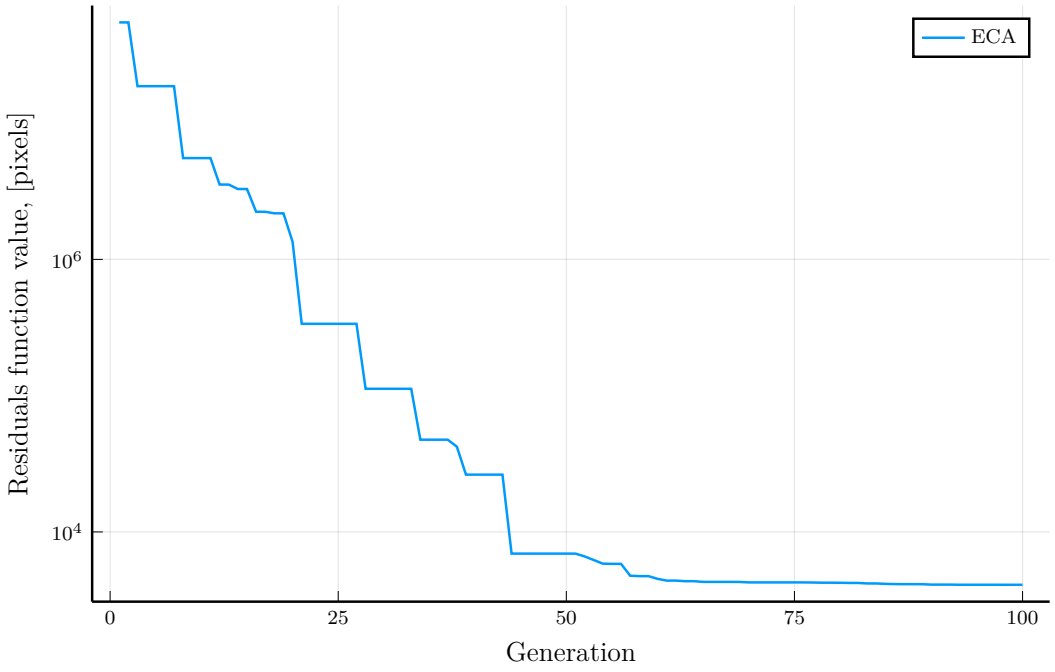


FIGURE 3.9: Error at each iteration of ECA for an elliptical equatorial orbit

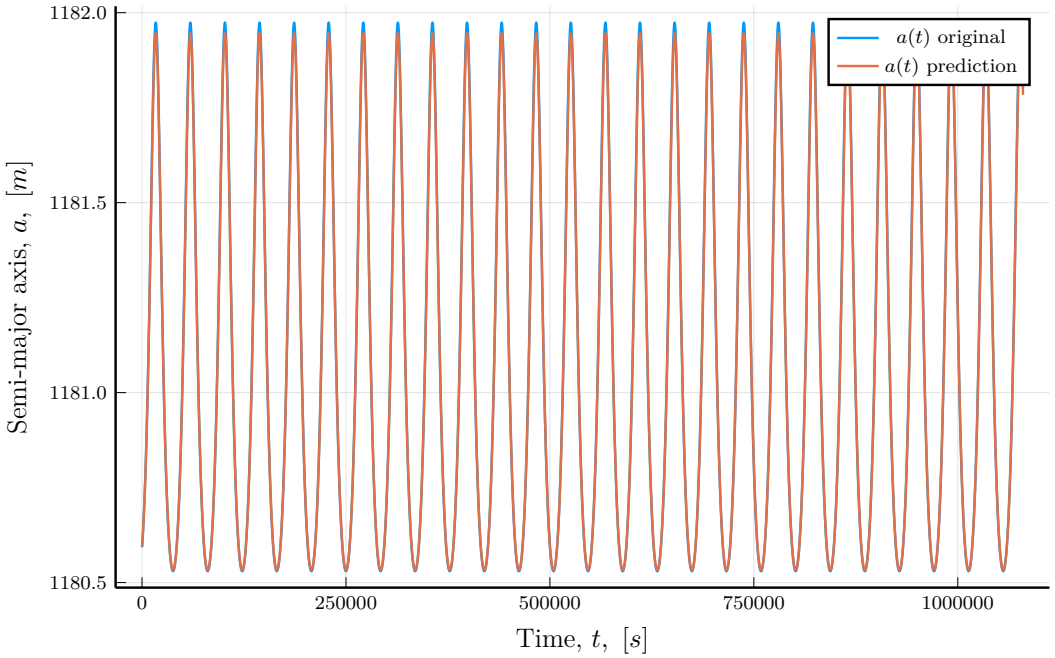


FIGURE 3.10: Original and predicted values of the semi-major axis for an elliptical equatorial orbit

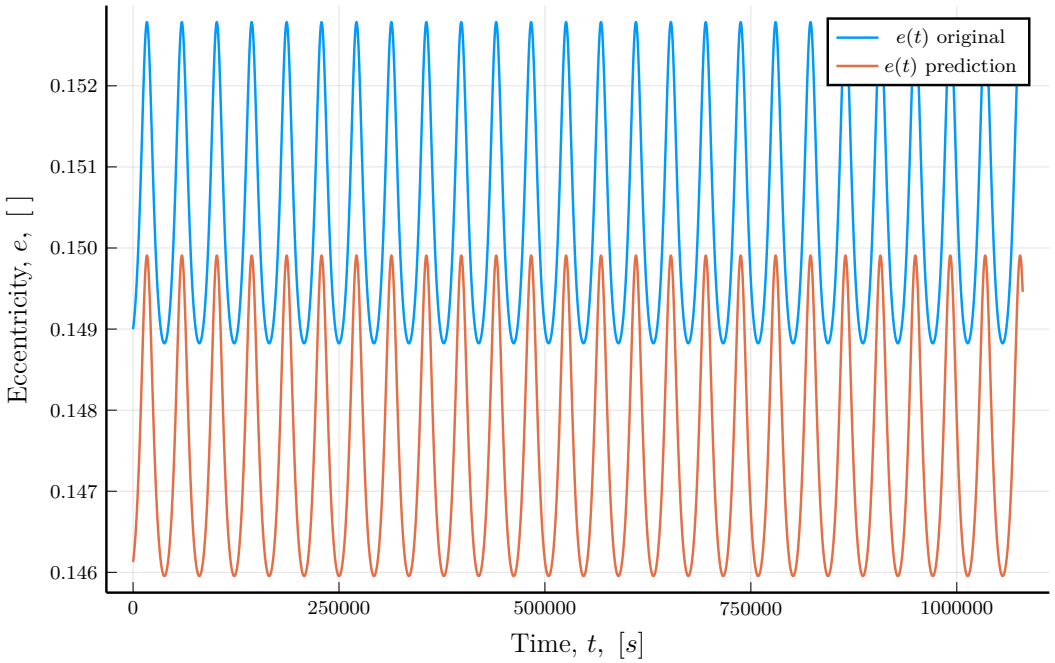


FIGURE 3.11: Original and predicted values of the eccentricity for an elliptical equatorial orbit

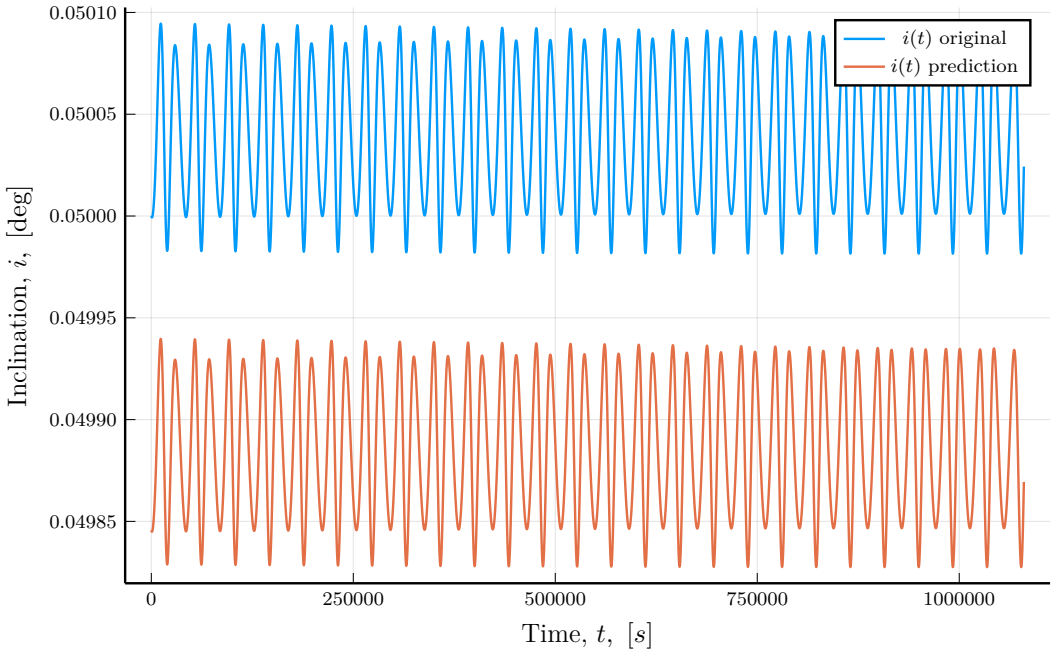


FIGURE 3.12: Original and predicted values of the inclination for an elliptical equatorial orbit

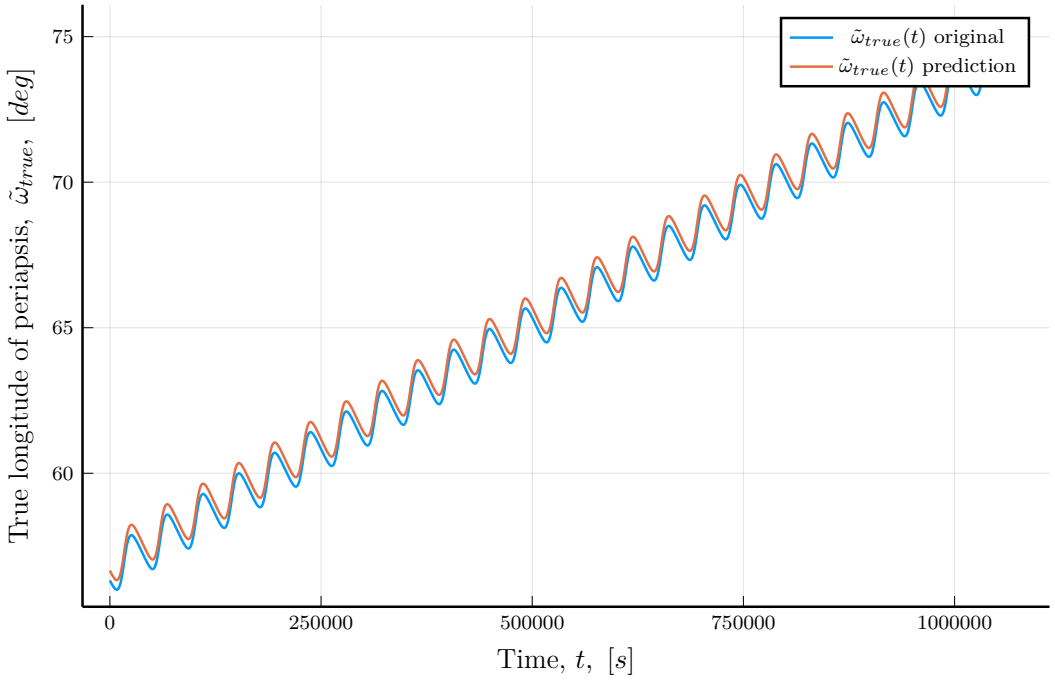


FIGURE 3.13: Original and predicted values of the true longitude of periapsis for an elliptical equatorial orbit

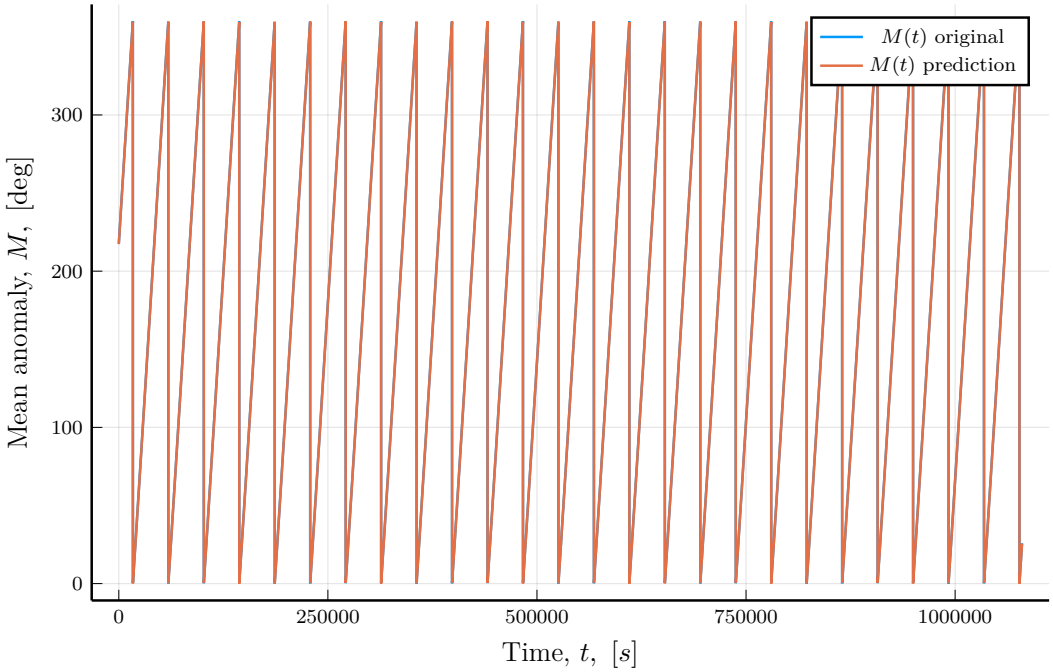


FIGURE 3.14: Original and predicted values of the mean anomaly for an elliptical equatorial orbit

TABLE 3.5: Original vs Predicted orbital elements and percentage errors for an elliptical equatorial orbit

Parameter	Original	Predicted	% error
a [m]	1180.593	1180.594912498158	0.000752
e [ ]	0.149	0.14612883844114952	0.905458
i [°]	0.05	0.04984557647488274	0.308660
Ω [°]	-	-	-
$\tilde{\omega}_{true}$ [°]	56.32	56.65855089419535	0.502331
M [°]	208.6	208.47245647042084	0.701786
μ [$m^3\,s^{-2}$]	36.328391472	36.288935865	0.108608

3.5.3 Circular Inclined Orbit

A circular inclined orbit has an eccentricity value very close to zero $e \approx 0$. In this case the argument of periapsis and the mean anomaly cannot be defined, since no periapsis exists. The argument of latitude u is used instead, which combines the argument of periapsis and mean anomaly. The constrains used by the algorithm have to be updated to reflect that and can be seen in Table 3.6.

From the total of **1000 observed images** to be generated, only **817 were usable**. The algorithm run for **628.0520** s and performed **3466 function calls**. Convergence of the algorithm can be seen in Figure 3.15. Figure 3.16 to Figure 3.20 show the values of the orbital elements as a function of time for both the observed and the predicted images,

TABLE 3.6: Constraints used by ECA for a circular inclined orbit of Dimorphos around Didymos

Parameter	Constraint Imposed
a	$[1160.0, 1220.0]$ [m]
e	$(0, 0.00000001]$ [ ]
i	$[0, 360)$ [°]
Ω	$[0, 360)$ [°]
u	$[0, 360)$ [°]
μ	$[0.8\mu_{predicted}, 1.2\mu_{predicted}]$ [$m^3\,s^{-2}$]

while Table 3.7 compares the orbital elements and the gravitational parameter using the mean absolute percentage error and the percentage error respectively.

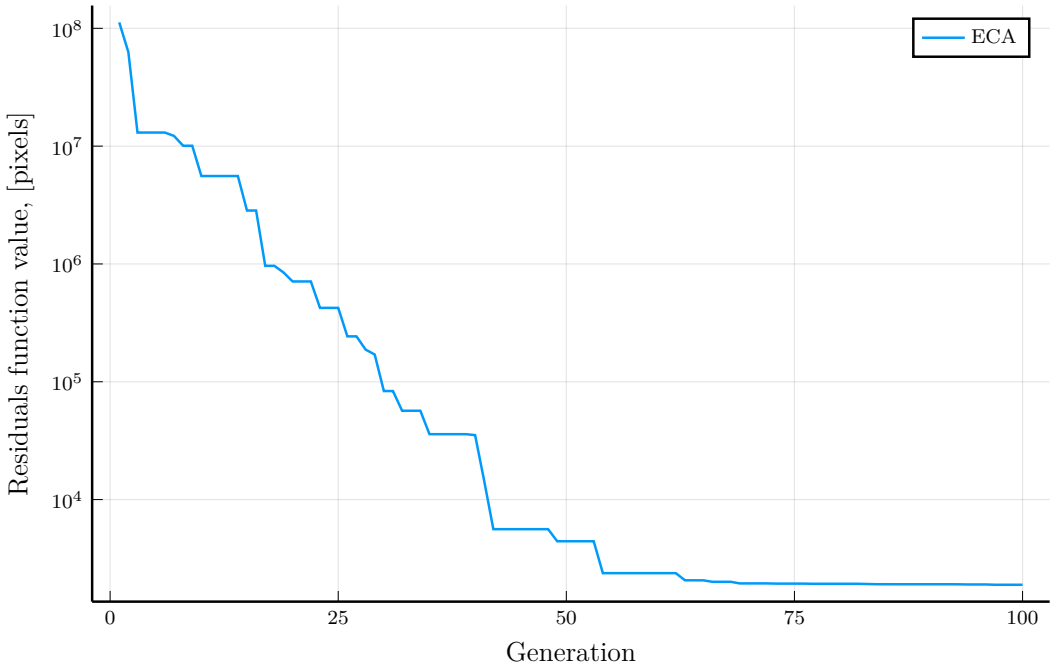


FIGURE 3.15: Error at each iteration of ECA for a circular inclined orbit

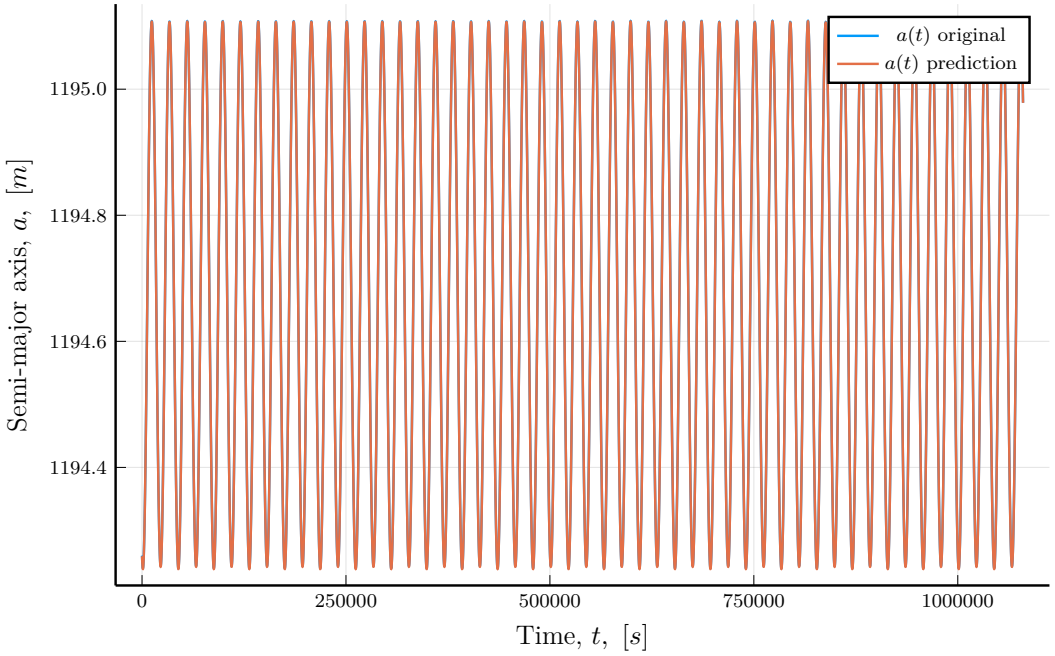


FIGURE 3.16: Original and predicted values of the semi-major axis for a circular inclined orbit

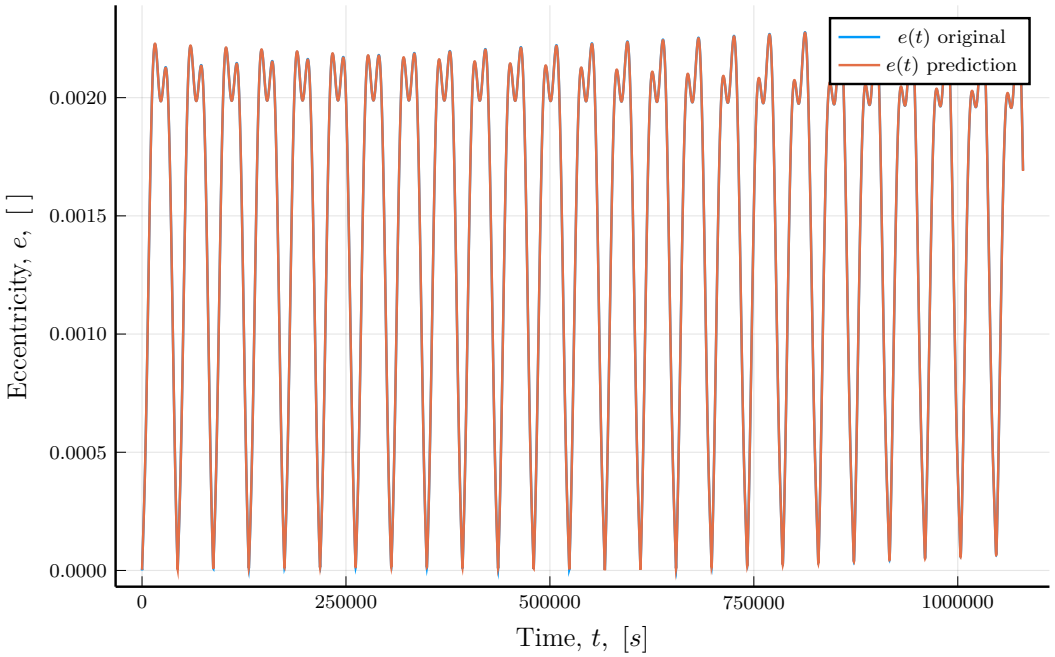


FIGURE 3.17: Original and predicted values of the eccentricity for a circular inclined orbit

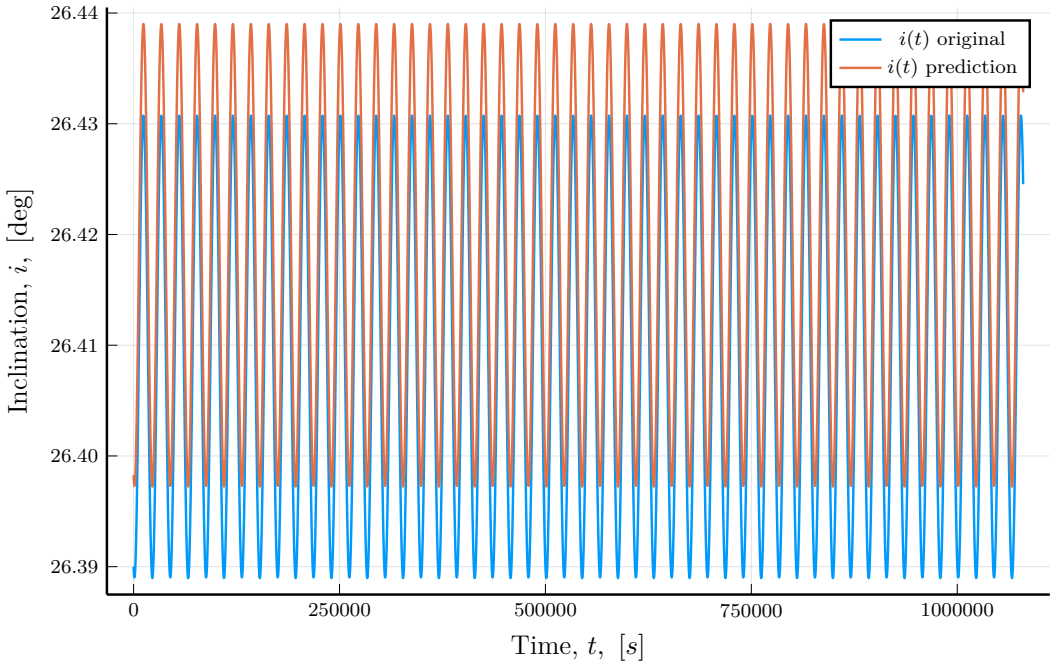


FIGURE 3.18: Original and predicted values of the inclination for a circular inclined orbit

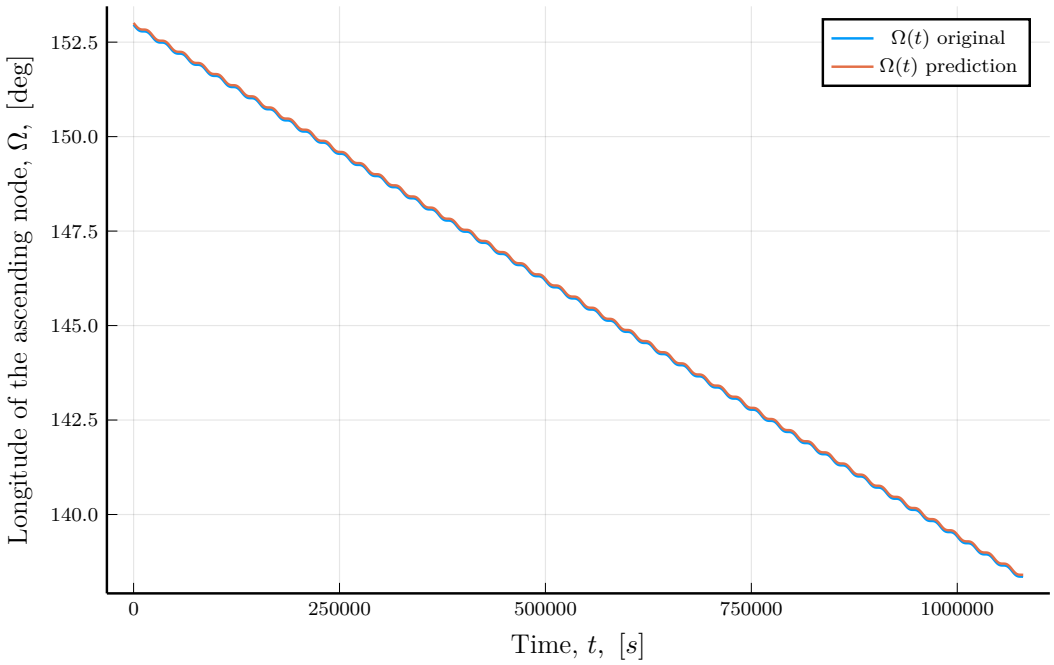


FIGURE 3.19: Original and predicted values of the longitude of the ascending node for a circular inclined orbit

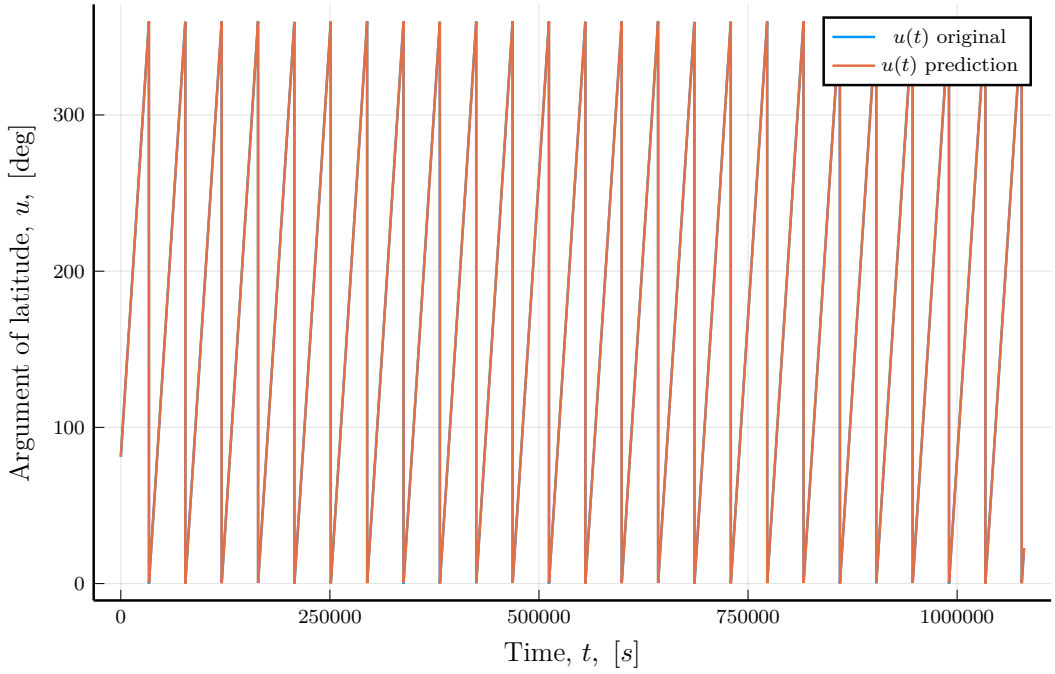


FIGURE 3.20: Original and predicted values of the argument of latitude for a circular inclined orbit

TABLE 3.7: Original vs Predicted orbital elements and percentage errors for a circular inclined orbit

Parameter	Original	Predicted	% error
a [m]	1194.26	1194.2589818467225	0.000083
e [ ]	0.0000073	2.9905597047728216e-6	0.275699
i [°]	26.39	26.39827739616834	0.031365
Ω [°]	152.96	153.00850518349245	1.054775
ω [°]	-	-	-
u [°]	81.26	81.20258444793858	0.007981
μ [$m^3\,s^{-2}$]	36.057895236	36.0896325478	0.088018

3.5.4 Circular Equatorial Orbit

A circular equatorial orbit has eccentricity and inclination value very close to zero $e \approx 0$, $i \approx 0$. In this case the longitude of the ascending node, the argument of periapsis and the mean anomaly cannot be defined, since no periapsis and no ascending node exist. The true longitude λ_{true} is used instead, which combines the longitude of the ascending node, the argument of periapsis and the mean anomaly. The constrains used by the algorithm have to be updated to reflect that and can be seen in Table 3.8.

From the total of **1000 observed images** to be generated, only **812 were usable**. The algorithm run for **542.0478** s and performed **3466 function calls**. Convergence of the algorithm can be seen in Figure 3.21. Figure 3.22 to Figure 3.25 show the values of the orbital elements as a function of time for both the observed and the predicted images,

TABLE 3.8: Constraints used by ECA for a circular equatorial orbit of Dimorphos around Didymos

Parameter	Constraint Imposed
a	$[1160.0, 1220.0]$ [m]
e	$(0, 0.00000001]$ [ ]
i	$[0, 0.1)$ [°]
λ_{true}	$[0, 360)$ [°]
μ	$[0.8\mu_{predicted},\ 1.2\mu_{predicted}]$ [$m^3\,s^{-2}$]

while Table 3.7 compares the orbital elements and the gravitational parameter using the mean absolute percentage error and the percentage error respectively.

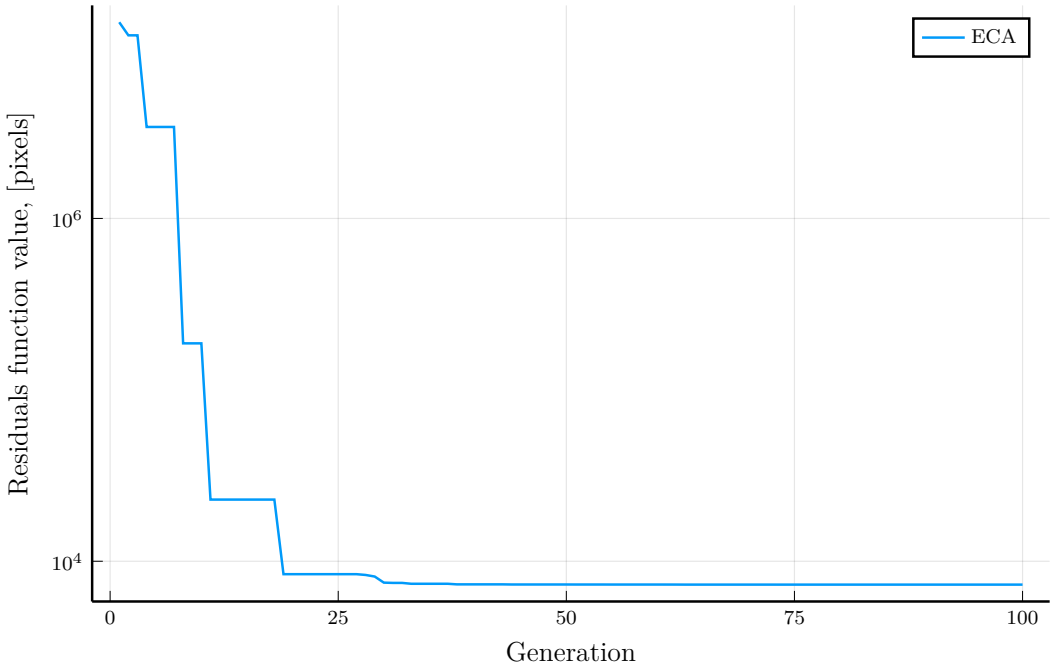


FIGURE 3.21: Error at each iteration of ECA for a circular equatorial orbit

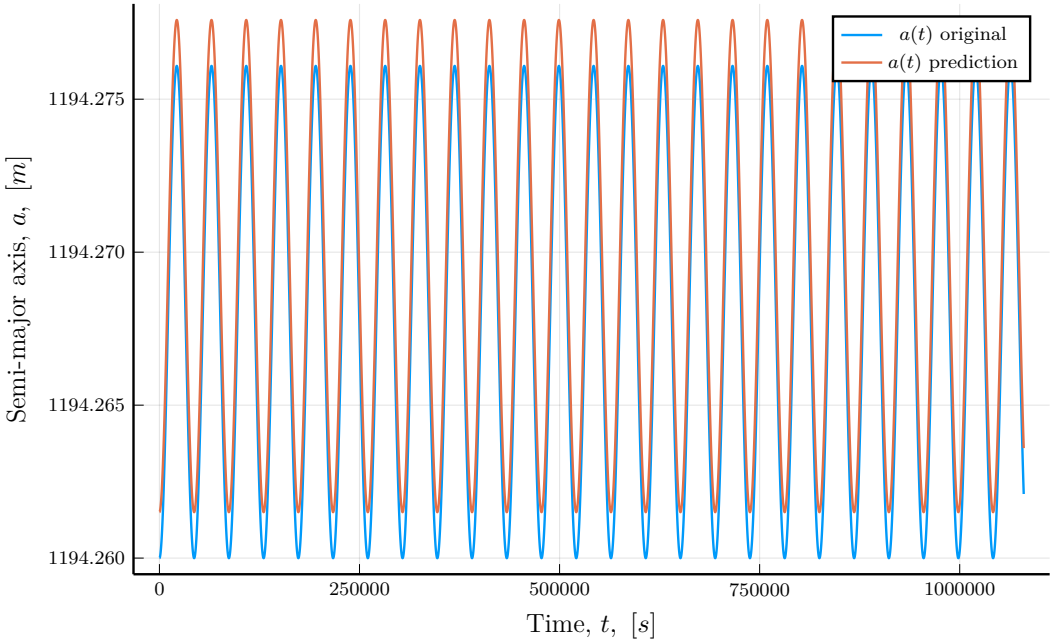


FIGURE 3.22: Original and predicted values of the semi-major axis for a circular equatorial orbit

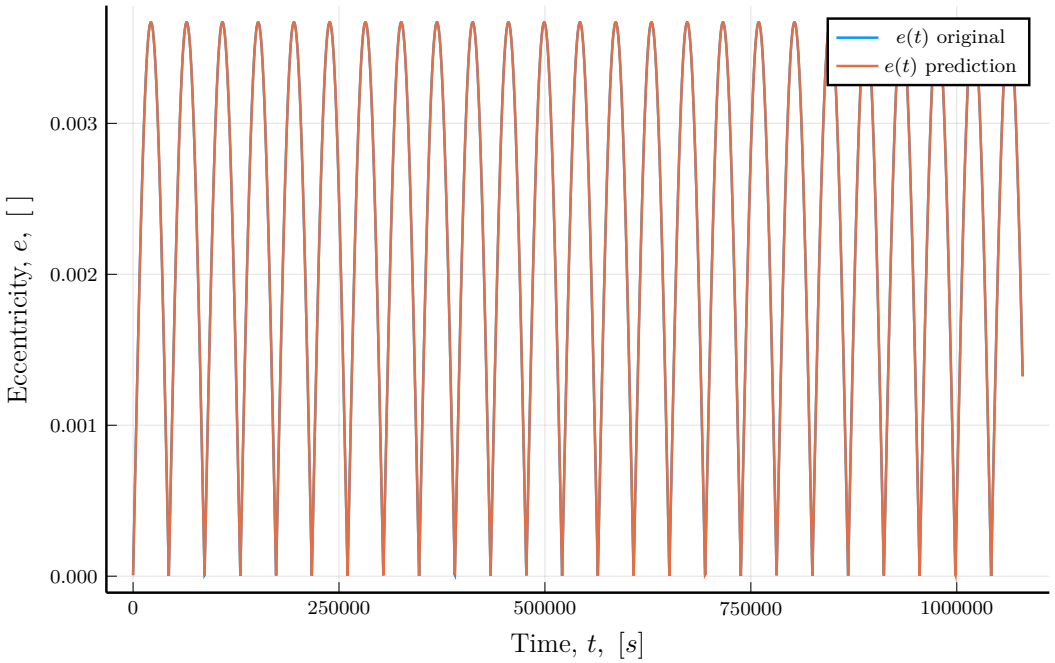


FIGURE 3.23: Original and predicted values of the eccentricity for a circular equatorial orbit

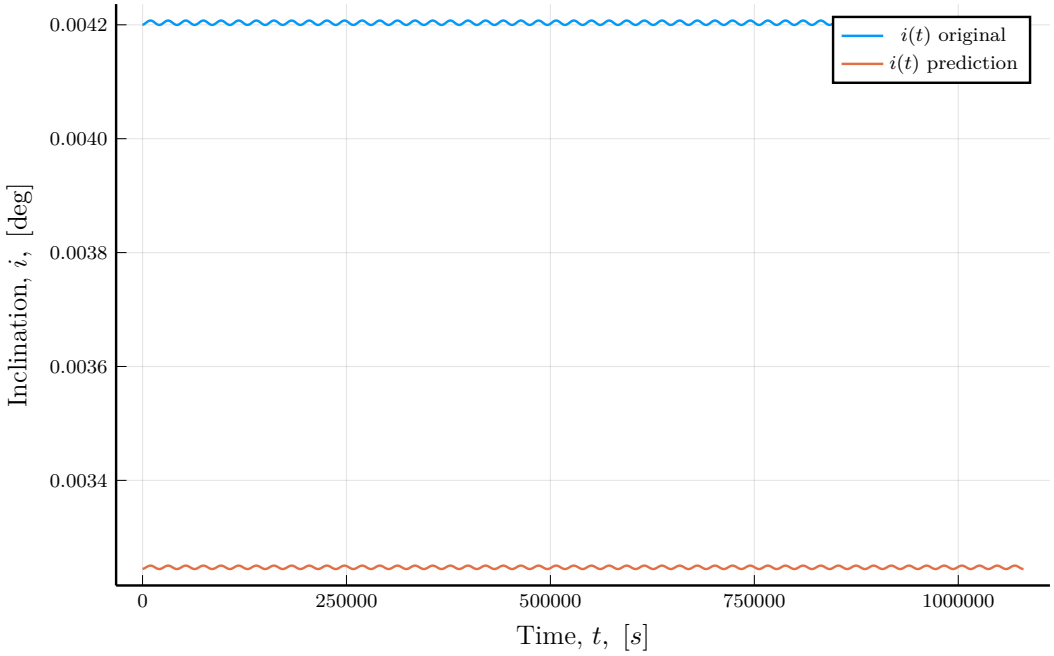


FIGURE 3.24: Original and predicted values of the inclination for a circular equatorial orbit

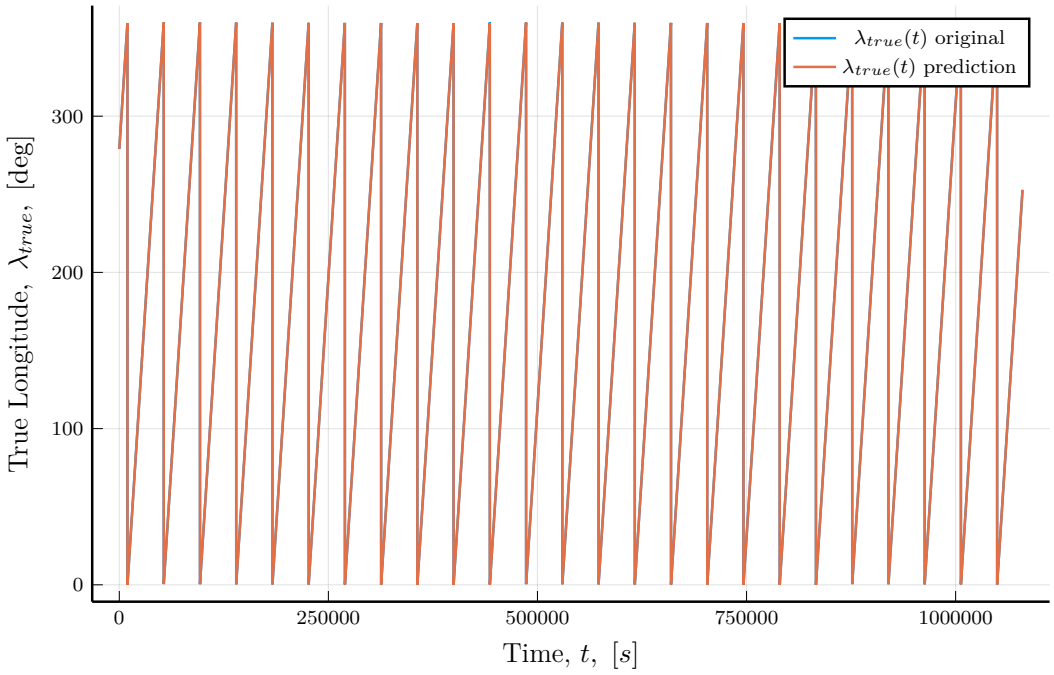


FIGURE 3.25: Original and predicted values of the true longitude for a circular equatorial orbit

TABLE 3.9: Original vs Predicted orbital elements and percentage errors for a circular equatorial orbit

Parameter	Original	Predicted	% error
a [m]	1190.63	1194.2589818467225	0.000125
e [ ]	0.0000062	7.813326627380822e-6	0.190320
i [°]	0.0042	0.003244515521395641	22.74961
Ω [°]	-	-	-
ω [°]	-	-	-
λ_{true} [°]	278.96	278.98685447683613	0.033267
μ [$m^3\,s^{-2}$]	35.7268699978	35.7471615962	0.056796

3.5.5 Preliminary findings and the case of inclination

The first results from running the orbit determination procedure for these reference cases seem to be very promising: most elements can be estimated with less than 1% mean absolute percentage error, while the gravitational parameter also appear to be estimated with less than 1% error. However, an anomaly is to be noted in Table 3.9, with the mean absolute percentage error for the inclination rising to 22.74961%. Repeating this case and the elliptical equatorial case for different parameters saw the errors fluctuate from below 1% to over 25% in some cases. Let us now attempt to explain this anomaly.

Firstly, from all the inclination diagrams above, it can be immediately observed that the value of inclination varies very little with the passage of time compared to the other orbital elements, to the point that it can be considered to be almost constant. Furthermore, in an attempt to identify the source of the anomaly, the spatial resolution was examined. The spatial resolution is a measure that refers to the smallest object that can be resolved by a sensor or, in this case, the linear dimension on the ground of the asteroid that is represented by each pixel [44]. The spatial resolution can be computed directly from the angular resolution by multiplying the latter with the distance of the object from the camera. Since the angular resolution of the asteroid framing camera is known from Table 2.3, let us quickly make a time plot of the spatial resolution over the entire observation window by computing the distance between Didymos and Hera to get a better understanding of spatial resolution. This figure can be seen in Figure 3.26.

From Figure 3.26, let's obtain a value of 2.6 m to be used as an average spatial resolution for investigation purposes. Furthermore, let's assume the orbit of Dimorphos to have a semi-major axis value of 1190 m, based on Table 2.2. If we now consider an equatorial orbit, let's attempt to find the minimum inclination value which would change the location of the geometric center of Dimorphos to a different pixel. This problem can be solved using trigonometry and is presented in Figure 3.27. Let's now attempt to find the minimum inclination for which Dimorphos should change to a

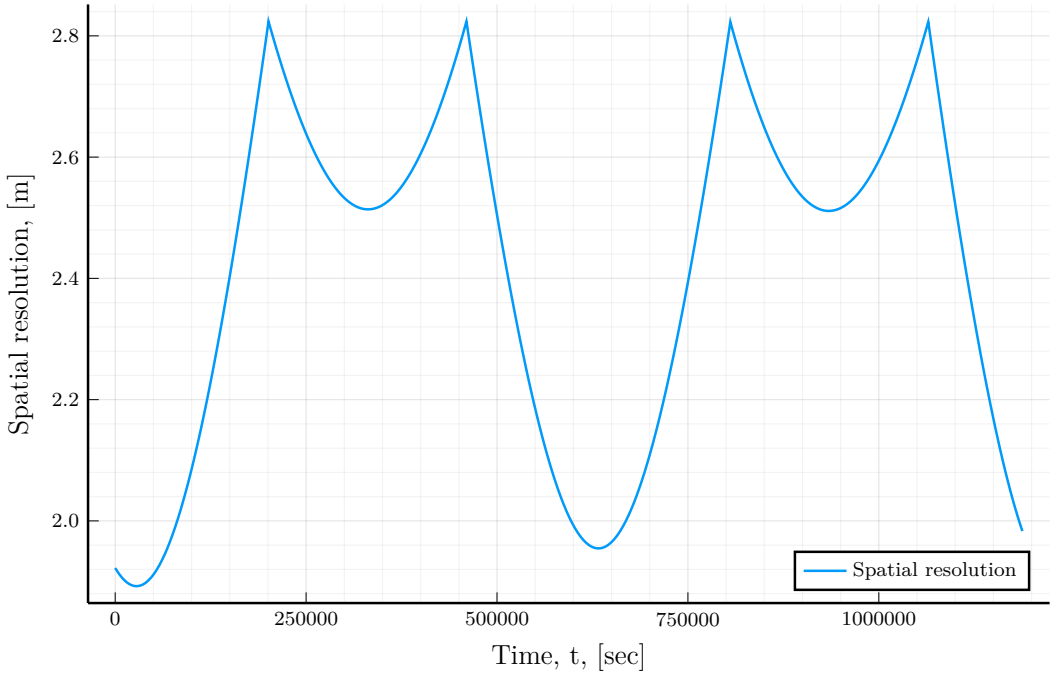


FIGURE 3.26: Spatial resolution as a function of time for the observation period of 300 hours

different pixel when Dimorphos is at the orbit's apoapsis (or the periapsis), meaning that the vertical distance x has to be greater than the spatial resolution.

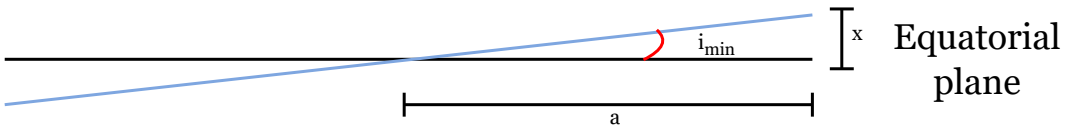


FIGURE 3.27: Geometry to find minimum inclination difference from an equatorial orbit to result in a different set of images

In this case x can be replaced with the average spatial resolution assumed above. We also have a value for the semi-major axis. Based on trigonometry, we have:

$$tan(i_{min}) = \frac{x}{a} = \frac{2.6}{1190} \tag{3.9}$$

solving for i_{min} we finally get:

$$i_{min} = 0.125183^{\circ} \tag{3.10}$$

The problem now becomes immediately obvious. The minimum inclination for which we can differentiate between two equatorial orbits is greater than the constraints we have imposed for the elliptical equatorial and the circular equatorial cases. In addition, the almost non-existent variation of the inclination throughout the observation period

means that for equatorial orbits we cannot try to fit the inclination using our algorithm, since the constraints are smaller than the minimum resolution which we can observe. The different mean absolute percentage errors ranging from below 1% to over 25% mentioned above are completely random: they correspond to the final guess used by the algorithm and have no importance since they are below the resolution. For this reason we won't attempt to fit the inclination in the case of equatorial orbits from now on, since it will have no impact to the orbit determination procedure.

Chapter 4

Orbit determination results & evaluation of performance

Having described the code used to generate the images captured by Hera and the algorithm used to perform orbit determination, it is now time to explore some more concrete results. First, a single initial state vector for the generation of the observed images will be used, followed by the interpretation of the result and the validation that the algorithm is able to determine the orbit for a larger number of runs. If the orbit determination is validated that way and produces adequate results, different initial state vectors will be examined to make sure that the algorithm works for many different cases. Finally, the possibility to use fewer images will be examined and an attempt will be made to fit keplerian orbital elements to the observed perturbed orbit of Dimorphos around Didymos.

4.1 Orbit determination for Dimorphos' orbit

4.1.1 A potential circular equatorial orbit for Dimorphos

The 300 h window discussed in Chapter 2, which occurs during the Early Characterization Phase of Hera's mission will be used again. Since the adequacy of the algorithm has already been proven, let's decide for an initial state vector which will be used to generate the observed set of images. The initial state vector selected can be found in Table 4.1. Do note the lack of longitude of the ascending node, inclination and argument of periapsis, which will not be used for the fitting process as decribed earlier in Section 3.5.5. The results from the algorithm, which took about 12 min to run, can be found in Table 4.2. A new way to visualize the result will also be provided, which shows the observed and predicted sets of coordinates in a single plot, which can be seen in Figure 4.5.

TABLE 4.1: Initial state vector used for the generation of the observed set of images

Parameter	Value
a	1180.329 [m]
e	0.0000096 [ ]
λ_{true}	147.326 [°]
μ	36.2112078095521 [$m^3\,s^{-2}$]

TABLE 4.2: Original vs Predicted orbital elements and percentage errors for the chosen orbit of Dimorphos

Parameter	Original	Predicted	% error
a [m]	1180.329	1180.33968186072	0.002727
e [ ]	0.0000096	8.13377155912324e-6	0.737925
i [°]	-	-	-
Ω [°]	-	-	-
ω [°]	-	-	-
λ_{true}	147.326	147.305858971508	0.247763
μ [$m^3\,s^{-2}$]	36.2112078095521	36.66749527	0.007328

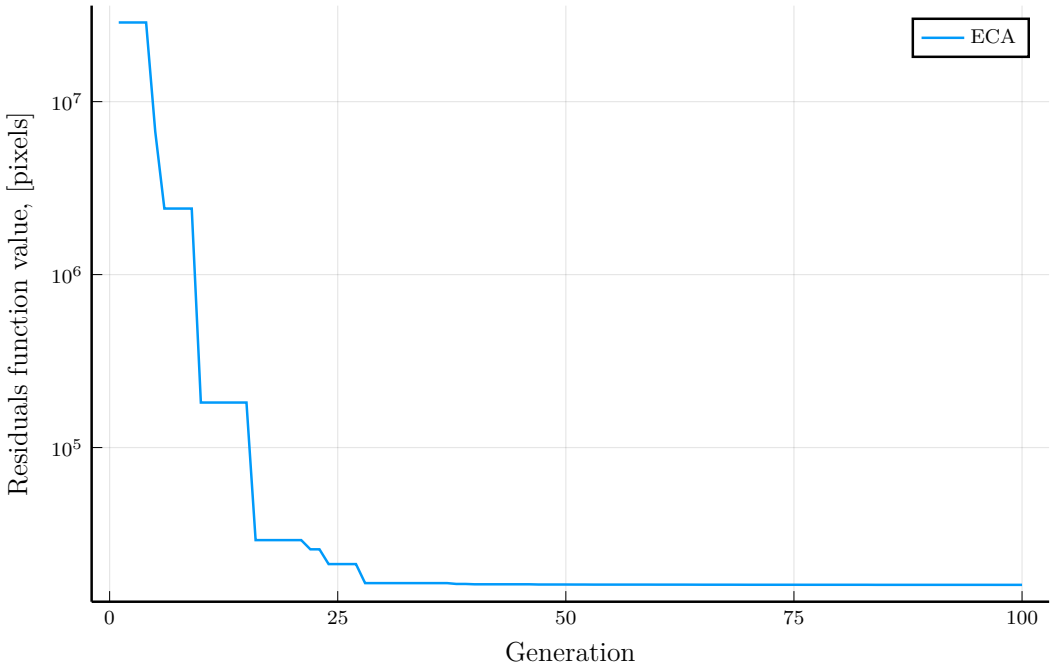


FIGURE 4.1: Error at each iteration of ECA for the orbit of Dimorphos

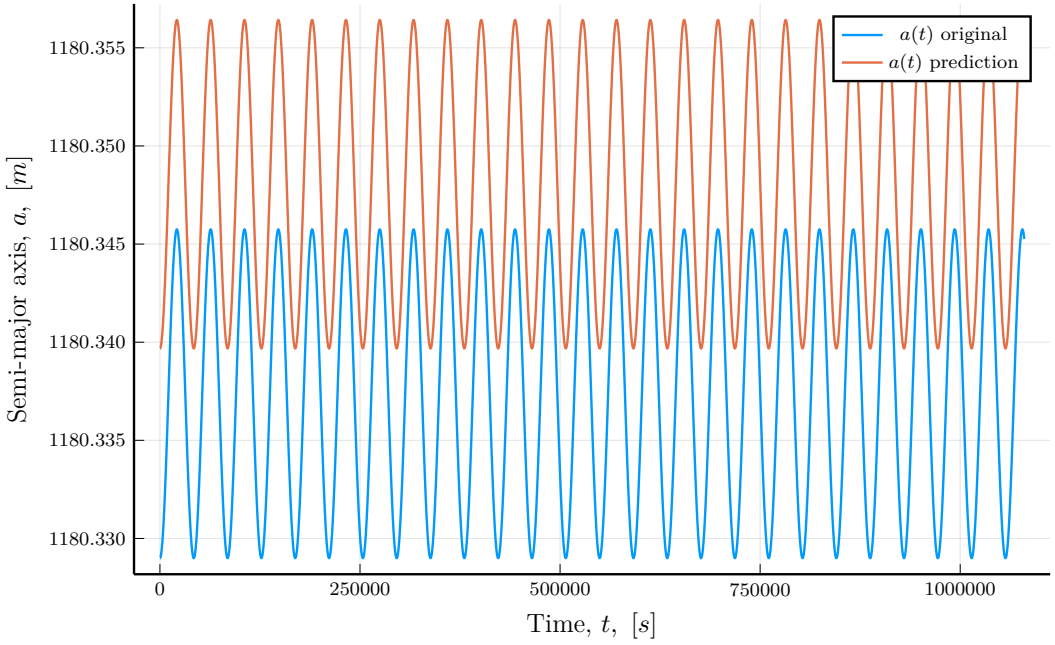


FIGURE 4.2: Original and predicted values of the semi-major axis for the orbit of Dimorphos

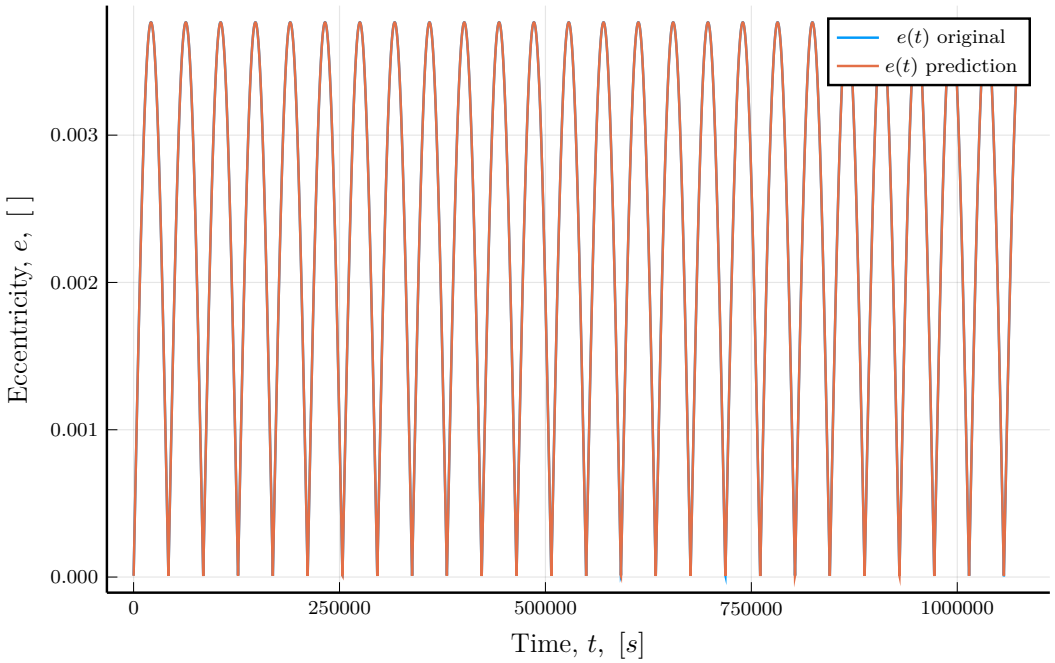


FIGURE 4.3: Original and predicted values of the eccentricity for the orbit of Dimorphos

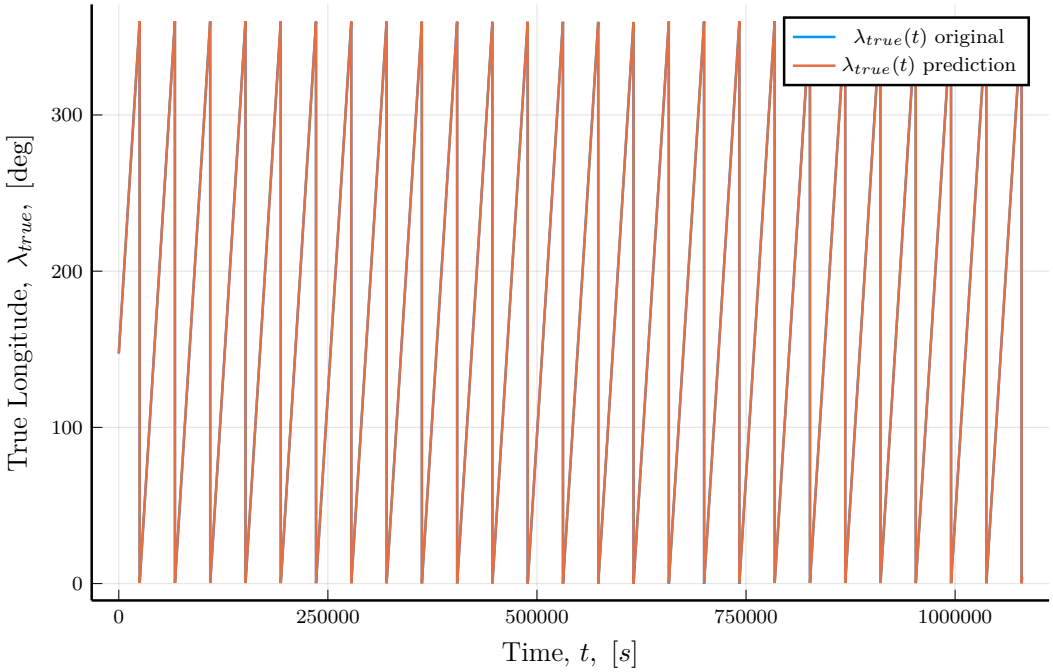


FIGURE 4.4: Original and predicted values of the true longitude for the orbit of Dimorphos

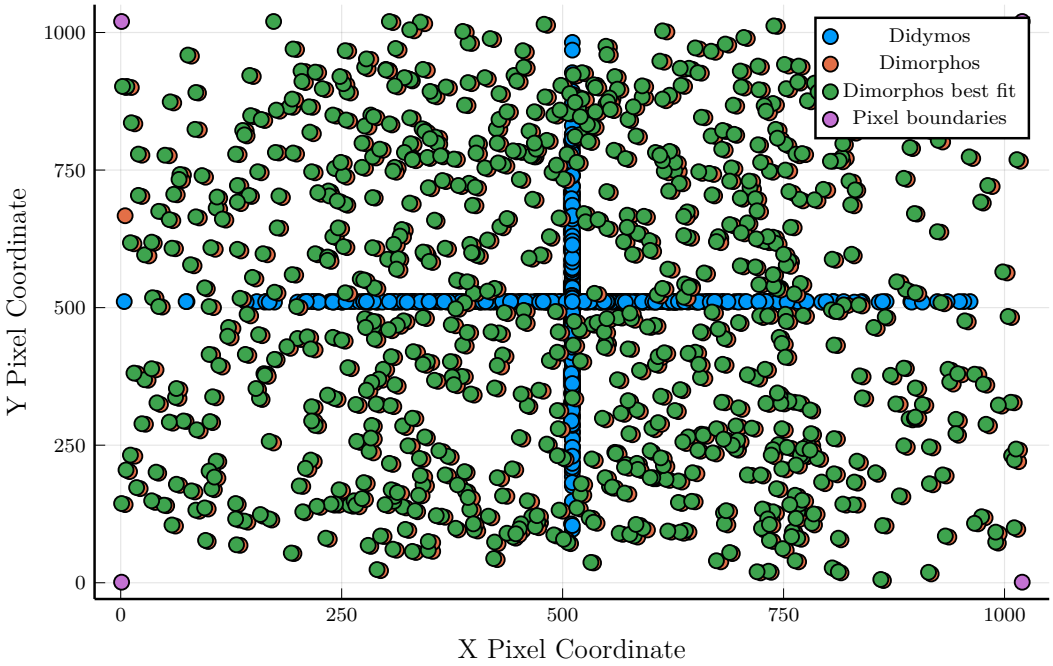


FIGURE 4.5: Original and predicted coordinates on the pixel grid for the orbit of Dimorphos

4.1.2 Repeatability analysis of the algorithm

The results provided above show that the algorithm can perform quite well, with the final results giving mean absolute percentage errors below 1% for each orbital element

and below 1% for the gravitational parameter. Considering the fact that convergence of the algorithm is not guaranteed for the reasons explained in Section 3.3, let's ensure now that the results described so far weren't "lucky guesses". The algorithm is then commanded to run continuously for the same original set of observed images to examine the output it produces at each run, with the predicted values generated by the algorithm in Table 4.3 and the % error results being presented in Table 4.4.

TABLE 4.3: Predicted initial state vector produced by the algorithm for 20 runs of the same set of observed images

Run ID	**Predicted** a [m]	**Predicted** e [ ]	**Predicted** λ_{true} [°]	**Predicted** μ [$m^3\,s^{-2}$]
#1	1180.339682	8.13E-06	147.305859	36.47655705
#2	1180.278227	1.04E-06	147.3281903	36.66749527
#3	1180.372943	5.43E-06	147.315742	36.61636663
#4	1180.182732	8.27E-06	147.3394979	36.01666168
#5	1180.368972	8.66E-06	147.35682	35.96385716
#6	1180.483087	8.64E-06	147.3136775	35.96385716
#7	1180.434957	9.62E-06	147.3167355	36.03227588
#8	1180.385632	1.95E-06	147.3407745	36.70674464
#9	1180.381408	5.77E-07	147.3149884	36.6960425
#10	1180.376835	2.72E-06	147.3142202	35.92487839
#11	1180.573045	9.38E-06	147.3313253	35.86587316
#12	1180.353934	3.28E-06	147.3386222	36.00496616
#13	1180.352732	8.27E-06	147.3394979	36.01666168
#14	1180.443805	8.71E-06	147.342159	36.1027755
#15	1180.447496	9.62E-06	147.3167355	36.03227588
#16	1180.299602	4.12E-06	147.3265379	36.4683235
#17	1180.327886	4.58E-06	147.3315979	36.21157991
#18	1180.299734	8.28E-06	147.3252727	36.46385387
#19	1180.355087	8.64E-06	147.3136775	35.96385716
#20	1180.356485	2.70E-06	147.3084908	35.92487839

TABLE 4.4: Mean absolute percentage error (for each orbital element) and percentage error (for gravitational parameter) for each of the 20 runs using the same set of observed images

Run ID	a % error	e % error	λ_{true} % error	μ % error
#1	0.002727	0.737925	0.247763	0.007328
#2	0.004676	0.990369	0.506663	0.012601
#3	0.003721	0.757856	0.388014	0.011189
#4	0.002321	0.419473	0.187688	0.00537
#5	0.002485	0.365895	0.156399	0.00683
#6	0.00279	0.563783	0.225893	0.00683
#7	0.001567	0.368042	0.170262	0.00494
#8	0.004795	0.946677	0.461658	0.013685
#9	0.004437	0.840472	0.460723	0.013389
#10	0.003857	0.468964	0.245128	0.00791
#11	0.003209	0.768048	0.315431	0.00954
#12	0.002113	0.414305	0.195094	0.0057
#13	0.002015	0.418818	0.184213	0.00537
#14	0.001254	0.251687	0.105544	0.00299
#15	0.001567	0.368042	0.170266	0.00494
#16	0.002492	0.957348	0.251665	0.007132
#17	9.60E-05	0.417246	0.017618	1.03E-05
#18	0.002482	0.665774	0.248433	0.006977
#19	0.002216	0.482187	0.239034	0.00683
#20	0.002326	0.449406	0.257523	0.00791

Let us now plot the results in histogram plots to observe if there is a trend. In Figure 4.7 to Figure 4.8 no distribution is immediately identifiable, however for the values of the semi-major axis in Figure 4.6 the data appear to be forming a normal distribution. This can be further investigated: let's make the null hypothesis H_0 that the data in Figure 4.6 come from a normal distribution. Using a normality test [45] a p-value of 0.23685 is obtained, indicating that the null hypothesis cannot be rejected. This is an important result, since the normal distribution formed by the predicted semi-major axis values ensures that the algorithm can guarantee the 1% accuracy requirement of Hera for 99.9% of the algorithm runs.

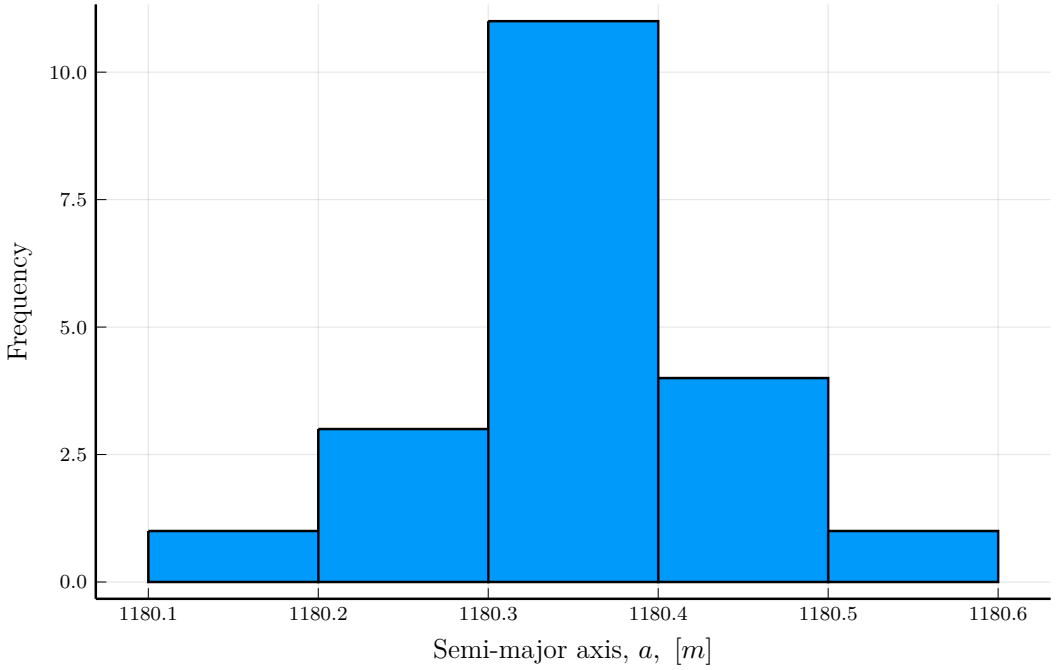


FIGURE 4.6: Histogram plot of the predicted semi-major axis values for the repeatability analysis

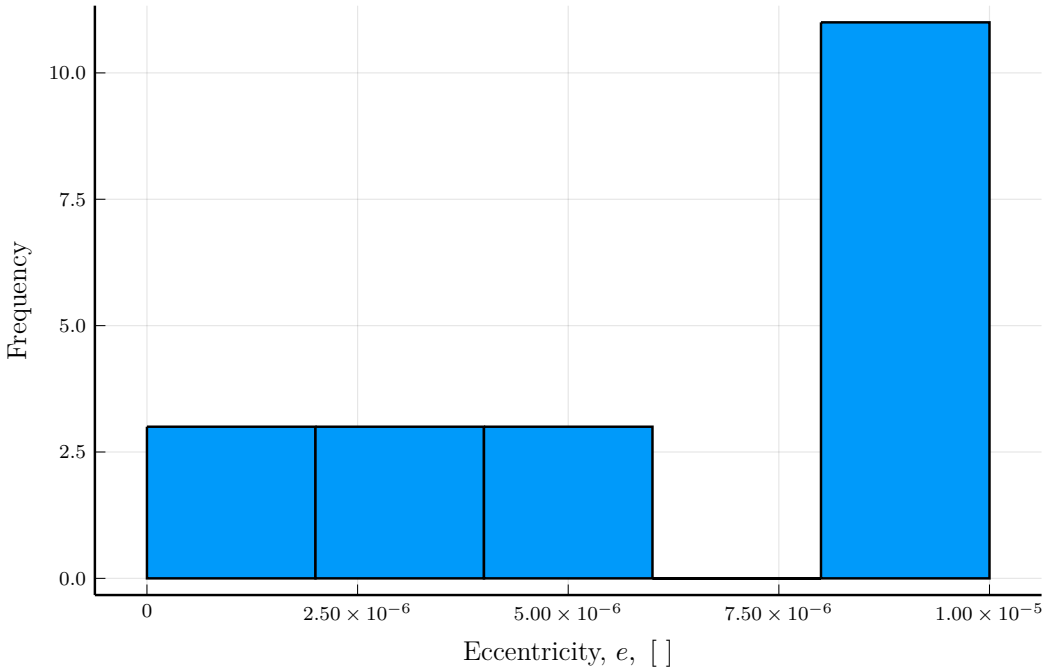


FIGURE 4.7: Histogram plot of the predicted eccentricity values for the repeatability analysis

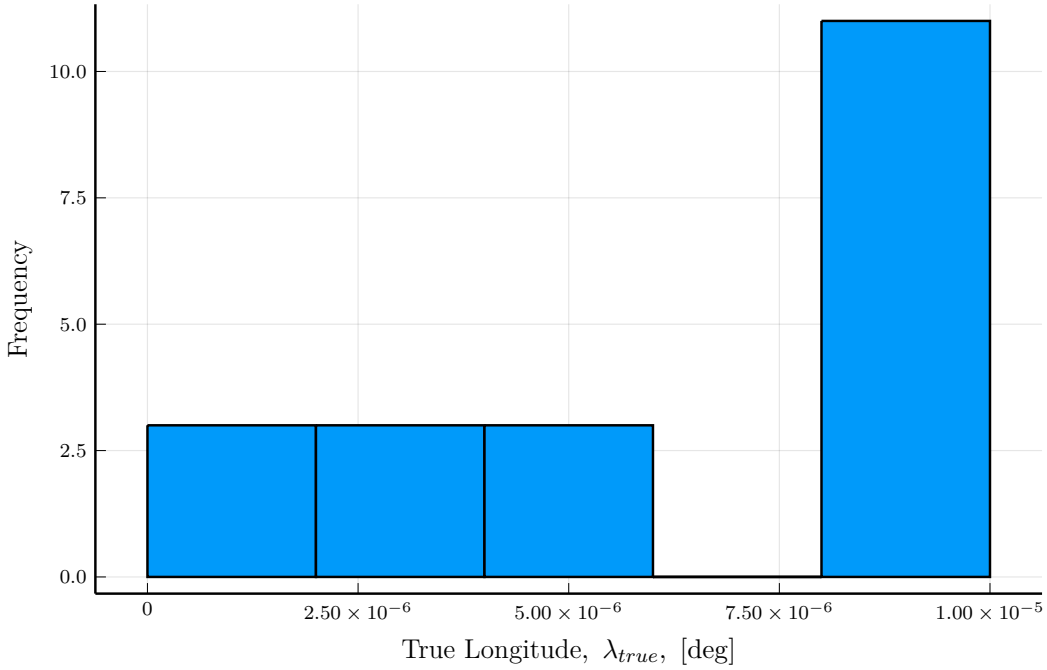


FIGURE 4.8: Histogram plot of the predicted true longitude values for the repeatability analysis

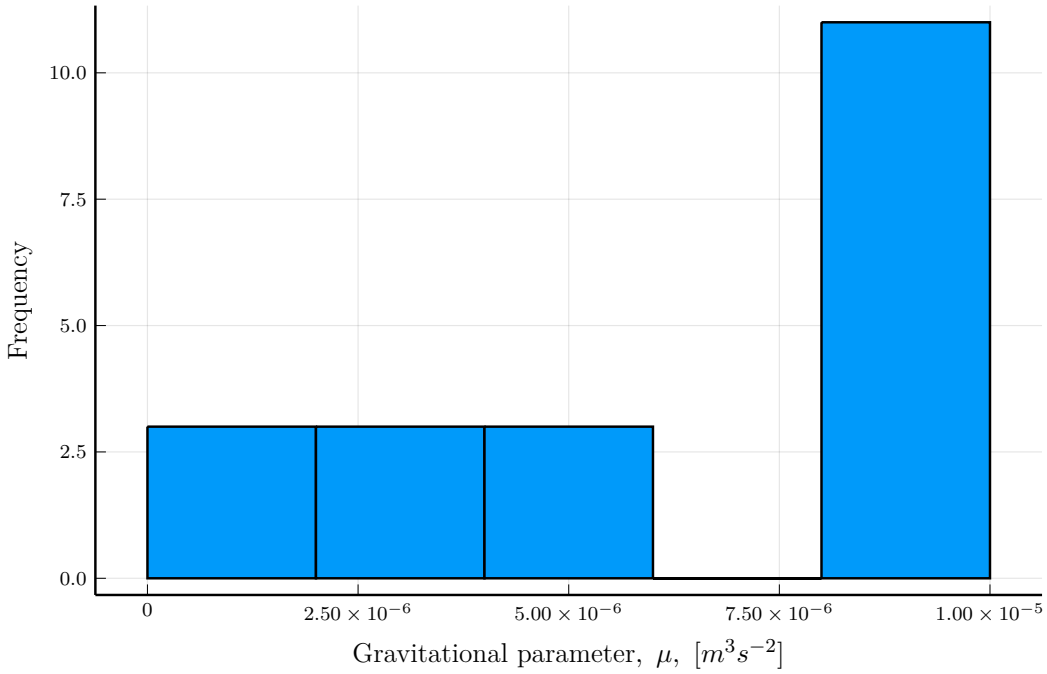


FIGURE 4.9: Histogram plot of the predicted gravitational parameter values for the repeatability analysis

4.1.3 Validating a wide set of orbital elements

Having examined the performance of the orbit determination algorithm for one initial state vector and having ensured the repeatability of results, different initial state

vectors can be tested to investigate the performance of the algorithm for a wide range of initial conditions. Since the assumption that the orbit of Dimorphos is a circular equatorial orbit was made, the constraints imposed to the algorithm in Table 3.8 will be used to generate 20 uniformly distributed initial state vectors, generate the 1000 corresponding simulated images for the 300 h observation window and let the algorithm perform the orbit determination procedure for each set separately.

The randomly generated values can be found in Table 4.5, while the predicted values by the algorithm can be found in Table 4.6. Mean absolute percentage error and percentage error values for the orbital elements and the gravitational parameter can be found in Table 4.7. From the last table, it can be observed that the algorithm is capable to maintain the mean absolute percentage error for the orbital elements and the percentage error for the gravitational parameter below 1%, with the exception of Set #13, where the mean absolute percentage error for the true longitude rises to a bit over 1%. Still, the results are encouraging and highlight that the orbit determination procedure using the metaheuristic algorithm is able to accurately determine the orbit of Dimorphos for a wide set of initial state vectors used.

TABLE 4.5: A set of 20 randomly generated initial state vectors to be examined by the orbit determination algorithm

Set ID	**Predicted** a [m]	**Predicted** e [ ]	**Predicted** λ_{true} [°]	**Predicted** μ [$m^3\,s^{-2}$]
#1	1179.191	7.08E-06	176.4690827	36.10088
#2	1201.78	7.81E-06	45.76288722	35.38039
#3	1187.705	9.42E-06	114.1760916	35.55443
#4	1175.078	8.20E-06	194.5325808	35.86933
#5	1218.245	3.87E-06	51.28376255	35.96662
#6	1172.066	7.31E-06	117.8167247	35.73679
#7	1180.437	9.87E-06	265.6392213	35.30164
#8	1179.318	5.88E-06	234.3064677	36.08385
#9	1169.051	6.05E-06	178.8169995	36.52474
#10	1217.469	4.21E-06	26.35699418	35.86065
#11	1172.636	6.17E-06	259.9427324	35.95775
#12	1160.609	8.45E-06	42.3836169	35.31176
#13	1175.347	1.34E-06	64.82213155	36.1625
#14	1190.598	6.08E-06	80.44576219	35.94703
#15	1176.127	2.39E-06	277.6675959	35.23262
#16	1194.874	2.01E-06	299.0331433	36.1295
#17	1181.791	3.22E-06	134.4751209	35.14443
#18	1169.172	6.47E-06	252.4735377	35.61801
#19	1163.808	9.61E-06	20.54482093	36.39913
#20	1189.343	5.45E-06	182.837113	35.40409

TABLE 4.6: Predicted values for the set of 20 randomly generated initial state vectors fitted using the orbit determination procedure

Set ID	a [m]	e [ ]	λ_{true} [°]	μ [$m^3\,s^{-2}$]
#1	1179.104	6.23E-06	176.4660337	36.09301
#2	1201.89	5.42E-06	45.74414527	35.39057
#3	1187.74	7.65E-06	114.1908176	35.55755
#4	1175.094	6.33E-06	194.5345504	35.89942
#5	1217.768	8.05E-07	51.29363966	35.92384
#6	1172.083	8.22E-06	117.7984577	35.7386
#7	1180.419	7.23E-06	265.5930119	35.30061
#8	1179.328	7.35E-06	234.3427047	36.11856
#9	1168.856	2.94E-06	178.8103088	36.5063
#10	1217.3	2.45E-06	26.39142728	35.84554
#11	1172.678	2.81E-06	259.9394816	35.96187
#12	1160.576	3.92E-06	42.4132583	35.30867
#13	1175.482	4.07E-06	64.8189723	36.23093
#14	1190.58	4.34E-06	80.43857884	35.94553
#15	1176.408	4.51E-06	277.6573113	35.25816
#16	1195.333	1.43E-06	299.0233667	36.17131
#17	1182.318	4.88E-06	134.4857526	35.19159
#18	1169.32	7.01E-06	252.4434273	35.63225
#19	1164.158	8.79E-06	20.53845288	36.43225
#20	1189.324	4.31E-06	182.8470415	35.44216

TABLE 4.7: Mean absolute percentage error (for each orbital element) and percentage error (for gravitational parameter) between each of the 20 randomly generated initial state vectors and the prediction by the orbit determination algorithm

Set ID	a % error	e % error	λ_{true} % error	μ % error
#1	0.007343	0.612383	0.950084044	0.021812
#2	0.0092	0.788967	0.387700943	0.028781
#3	0.002994	0.645716	0.384758555	0.00877
#4	0.002687	0.5414	0.147347713	0.08388
#5	0.003911	0.234648	0.457822753	0.118949
#6	0.001448	0.524816	0.115399907	0.005072
#7	0.001486	0.290656	0.224281739	0.002905
#8	0.032242	0.789692	0.736353777	0.096179
#9	0.016628	0.963294	0.491601402	0.050468
#10	0.013834	0.628688	0.433231264	0.04212
#11	0.003619	0.226005	0.487780784	0.01148
#12	0.002833	0.2232	0.519069465	0.008747
#13	0.006255	0.124871	1.177381897	0.189215
#14	0.001494	0.66484	0.495644826	0.004163
#15	0.023925	0.241478	0.759059406	0.072485
#16	0.038408	0.023748	0.147973541	0.115707
#17	0.044612	0.808658	0.33142692	0.134187
#18	0.012674	0.328433	0.487610497	0.039981
#19	0.030028	0.413739	0.589657965	0.091004
#20	0.035244	0.518182	0.594577046	0.107557

4.2 Possibility of using fewer images

Up until this point, a set of 1000 images taken by Hera over a time period of 300 h has been used for the orbit determination process. It would be however interesting to explore if the algorithm developed can be used for a smaller set of images. This would allow us to explore narrower time intervals and combine them to perform statistical analysis of the observational data. Furthermore, if the orbit determination can work for a smaller number of images, it can be ensured that the science output from the Hera mission is guaranteed even in case of an anomaly. For example, in case Hera encounters a critical error during the 300 h window of the Early Characterization Phase (ECP) and has to abort the observations in order to enter the safe system mode, it is important to understand the impact this would have for the science output. Therefore, let's attempt to determine the smallest possible number of images which the orbit determination algorithm needs in order to accurately reproduce the orbit of Dimorphos around Didymos.

To do that, let's explore how the mean absolute percentage error for each orbital element changes when the number of images is reduced. As previously discussed, given the fact that Dimorphos' orbit around Didymos is considered to be a circular equatorial orbit, only the semi-major axis a, eccentricity e and true longitude λ_{true} are used to determine the orbit. The optimization sequence is then executed for 10, 20, 40, 80, 100, 200, 400, 800 and 1000 images. The purpose of that is to create a semi-log plot which shows the mean absolute percentage error for each element as a function of the number of images. The result can be seen in Figure 4.10.

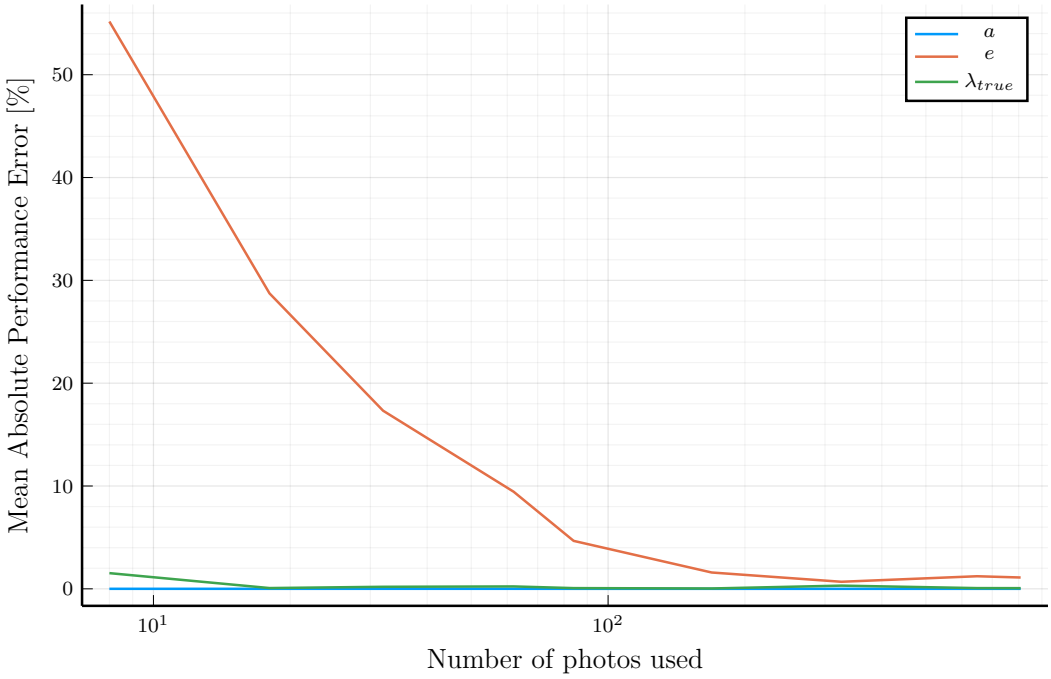


FIGURE 4.10: Mean absolute performance error as a function of the number of images used in the orbit determination procedure

From Figure 4.10, it can be immediately observed that the optimization method used can very accurately estimate the semi-major axis and the true longitude of the orbit even for a small number of images, with the error remaining under 2% even for 10 images. Let us also point out that in Figure 4.10 the values on the x-axis appear to begin for 8 images and end at around 800 images. This happens because even though the start point was set to be 10 images and the end point to be 1000, there is no guarantee that all images will be usable, for the reasons previously discussed in Section 2.3.4 and Section 2.4. Therefore, less usable images are available for the optimization procedure. It also appears that the eccentricity is the most difficult orbital element to fit, with the mean absolute percentage error being over 50% for 8 images and slowly decreasing as more images are provided to the algorithm. The mean absolute percentage error for the eccentricity appears to drop below an acceptable level of 2% if the optimization procedure is given a minimum of 200 usable images to work with.

This observation can then be tested. Let's consider the usual 300 h interval used so far and the 1000 images taken but with a small twist: The 1000 images will be split to 4 smaller sets of images. These smaller sets will then be used to determine the orbit of Dimorphos at each time interval beginning from the first image of the set and spanning to the last one. In theory, if the results obtained from Figure 4.10 are accurate, the optimization procedure should work for each one of the 4 smaller sets of images, with the mean absolute percentage error not rising above 2% for each orbital element.

TABLE 4.8: Mean absolute percentage error of the semi-major axis of the fitted orbit determined by each of the 4 sets of images

Set ID	Original Initial State [m]	Predicted Initial State [m]	% error
#1	1184.387	1184.3771246467256	0.0008342
#2	1184.4	1184.396528917144	0.000297
#3	1184.4	1184.395451922005	0.000382
#4	1184.39	1184.4085581356699	0.000728

TABLE 4.9: Mean absolute percentage error for the eccentricity of the fitted orbit determined by each of the 4 sets of images

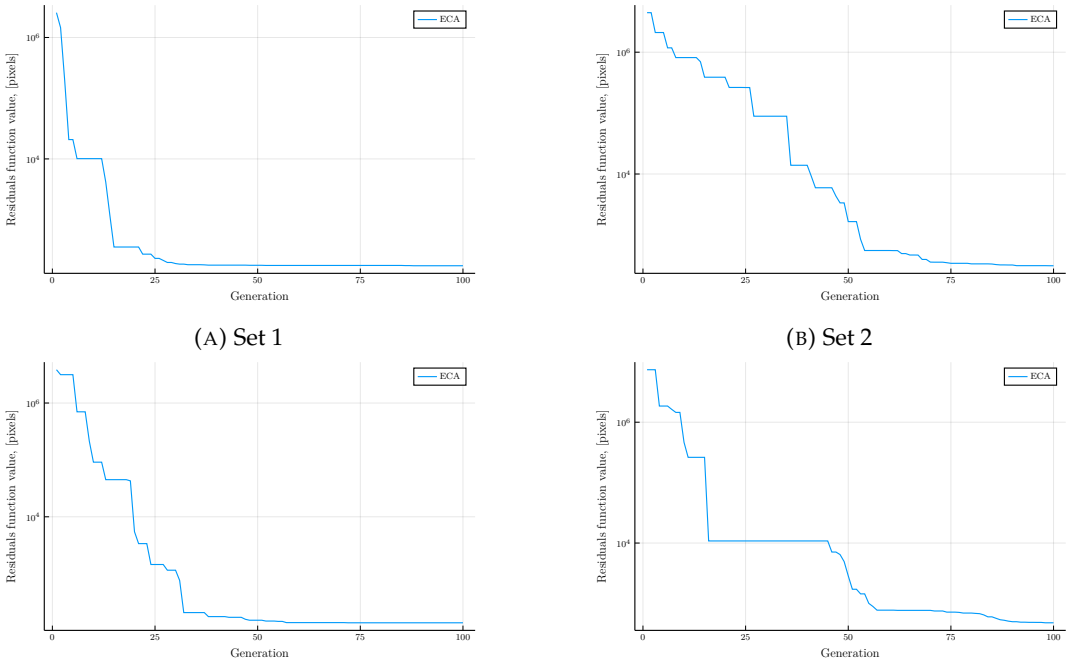
Set ID	Original Initial State [ ]	Predicted Initial State [ ]	% error
#1	0.00000732	8.706252408114852e-6	0.206641
#2	0.00280329	0.0027448908323160882	1.1001981
#3	0.00370323	0.0037068614854064858	1.283144
#4	0.00204865	0.002114827171445058	1.967733

TABLE 4.10: Mean absolute percentage error for the true longitude of the fitted orbit determined by each of the 4 sets of images

Set ID	Original Initial State [°]	Predicted Initial State [°]	% error
#1	138.25	138.2441625230415	0.025384
#2	239.885	239.39338720027598	1.296446
#3	342.505	342.1067193228484	1.343538
#4	84.4924	85.16567347401201	0.939121

Table 4.8 to Table 4.10 provide the mean absolute percentage error of each orbital element for each one of the 4 sets of images examined. Immediately, it can be determined that the claim made previously is valid: the error in all cases is kept below 2%. A general conclusion can be drawn: increasing the number of images will definitely lower the mean absolute percentage error, but a set of 250 images is enough to determine the orbit of Dimorphos with a mean absolute percentage error less than 2% for each orbital element.

This analysis is a bit more expensive in terms of computational resources. In total, 13771 function calls where required to fit the 4 orbits described in each image set, with the total run time being 1381.894 s or a bit over 23 min. Furthermore, the number of iterations was kept to 100 and from Figure 4.11 it can be determined that the algorithm tends to converge in this case as well for each image set.



(A) Set 1 (B) Set 2

(C) Set 3 (D) Set 4

FIGURE 4.11: Error at each iteration of ECA for each different set of images. 100 iterations appear to ensure convergence in this case as well.

4.3 Fitting of classical orbital elements

It was shown that the algorithm can perform orbit determination with smaller sets of images using the osculating elements produced if the J_2 error is considered. The question now is the following: can an even smaller sets of images be used to attempt to fit Keplerian orbital elements, produced without taking the effects of J_2 in consideration? Before exploring very narrow windows containing 30 images or less, let's first split up the original observation window of 300 h and 1000 images to a set of 10 narrower windows each corresponding to 30 h and 100 images each. After the generation of the observed images is completed, the equation of motion is updated in the integrator to:

$$\ddot{\vec{r}} = -G\left(m_{Didymos} + m_{Dimorphos}\right)\frac{\vec{r}}{r^3} \tag{4.1}$$

removing the J_2 perturbation from the equations of motion. The algorithm is then allowed to fit Keplerian orbital elements. The original orbital elements used for the

observed sets of images can be found in Table 4.11, while the fitted orbital elements can be found in Table 4.12, with the error being presented in Table 4.13. Since the fitting involves unperturbed keplerian elements, the % error for the semi-major axis and the eccentricity now refer to a percentage error between the predicted and the observed values, while the mean absolute percentage error is used only for λ_{true}.

TABLE 4.11: Orbital elements for each of the 10 sets of images used for the fitting of Keplerian orbital elements

Set ID	Mean a observed	Mean e observed	λ_{true} observed	μ observed
#1	1178.531	2.41E-03	39.236	35.5259008622723
#2	1178.559	2.47E-03	234.846	35.5259008622723
#3	1178.531	2.40E-03	70.762	35.5259008622723
#4	1178.557	2.43E-03	267.076	35.5259008622723
#5	1178.553	2.42E-03	102.819	35.5259008622723
#6	1178.534	2.40E-03	299.273	35.5259008622723
#7	1178.539	2.43E-03	134.412	35.5259008622723
#8	1178.545	2.42E-03	330.906	35.5259008622723
#9	1178.538	2.45E-03	166.564	35.5259008622723
#10	1178.552	2.42E-03	2.999	35.5259008622723

TABLE 4.12: Predicted orbital elements for each of the 10 sets of images used for the fitting of Keplerian orbital elements

Set ID	Mean a predicted	Mean e predicted	λ_{true} predicted	μ predicted
#1	1175.723	2.44E-05	39.206	35.54146
#2	1176.319	3.01E-04	235.212	35.58291
#3	1176.618	4.75E-03	71.272	35.61765
#4	1177.227	5.28E-04	267.152	35.65893
#5	1177.02	4.00E-03	102.923	35.53256
#6	1176.829	4.99E-06	299.226	35.57589
#7	1175.798	5.28E-04	134.785	35.44854
#8	1176.638	4.00E-03	330.856	35.42893
#9	1176.164	2.96E-03	166.633	35.60285
#10	1176.638	2.70E-03	3.178	35.58954

TABLE 4.13: Error and mean absolute percentage error for each of the orbital elements and gravitational parameter for the fitting of Keplerian orbital elements

Set ID	% error a	% error e	% MAPE λ_{true}	% error μ
#1	0.23833	98.98975	0.214901	0.0438
#2	0.190119	87.83111	0.345677	0.16048
#3	0.162318	139.8281	0.355191	0.25827
#4	0.112826	78.25133	0.189587	0.37446
#5	0.130064	65.13716	0.366515	0.01874
#6	0.144669	99.79221	0.148847	0.14071
#7	0.23254	78.28913	0.152632	0.21777
#8	0.161836	65.22607	0.308989	0.272958
#9	0.201447	20.91823	0.172781	0.21659
#10	0.162429	11.48062	0.388258	0.17914

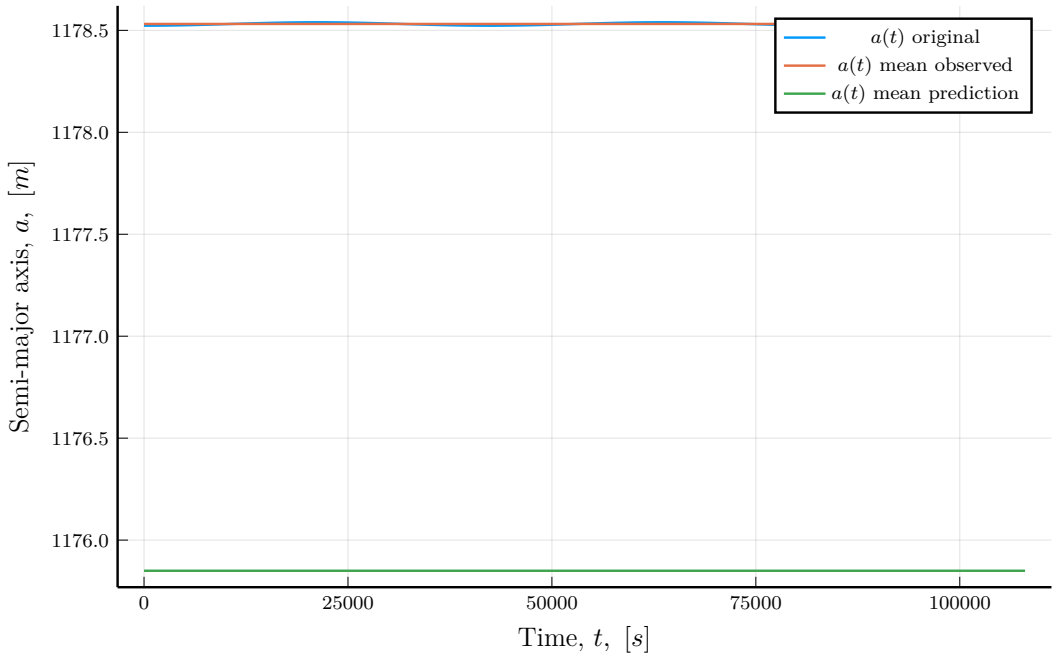


FIGURE 4.12: Original and predicted values using of the semi-major axis for the orbit of Dimorphos for a fit using Keplerian elements

Some extremely interesting observations can be made using Figure 4.12 to Figure 4.14. First of all, in Figure 4.12, it should be noted that the predicted mean value of the semi-major axis has an absolute error of about 2.6 m and the percentage error is about 0.1 to 0.2 % in most cases. In comparison, for the fitting of osculating orbital elements, the absolute error was below 0.1 m (as seen in Figure 4.2) and the percentage error was below 0.003%. It appears that the algorithm was taking advantage of the osculating elements to perform a more accurate fit during the orbit determination procedure. With the attempt to fit Keplerian elements, the variation of the orbital elements was

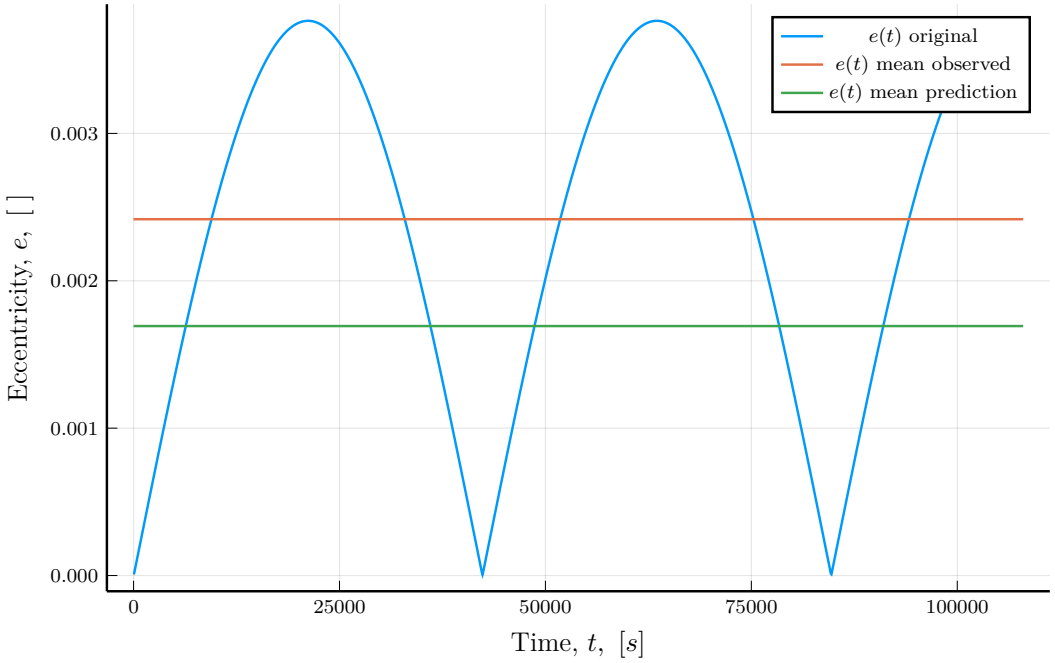


FIGURE 4.13: Original and predicted values of the eccentricity for the orbit of Dimorphos for a fit using Keplerian elements

effectively removed from the fitting procedure, and the orbit determination now has a detection limit equal to the spatial resolution: in this case, it also coincides with the average spatial resolution of 2.6 m that was obtained before using Figure 3.26.

While the error for the gravitational parameter and the true longitude remain at acceptable level, the same can't be said for the error of the eccentricity, which from Table 4.13 can be seen to vary from 11 to 139 %. The lack of the variation of parameters seems to hurt eccentricity the most, let's briefly examine why.

Let's assume an average value for the spatial resolution of 2.6 m, this means that each pixel corresponds to about 2.6 $\times$ 2.6 m. Let's now attempt to find the minimum change in eccentricity which can be detected using our camera by using the definition of the apoapsis. The radius of the apoapsis is defined as:

$$r_a = a(1 + e) \tag{4.2}$$

This equation can be solved for the eccentricity to result in:

$$e = \frac{r_a}{a} - 1 \tag{4.3}$$

Let's now use the value of 1178.523 m for the semi-major axis and since the minimum change for the location of the apoapsis has to be at least 2.6 m, let's add that value to the semi-major axis to represent the apoapsis to get:

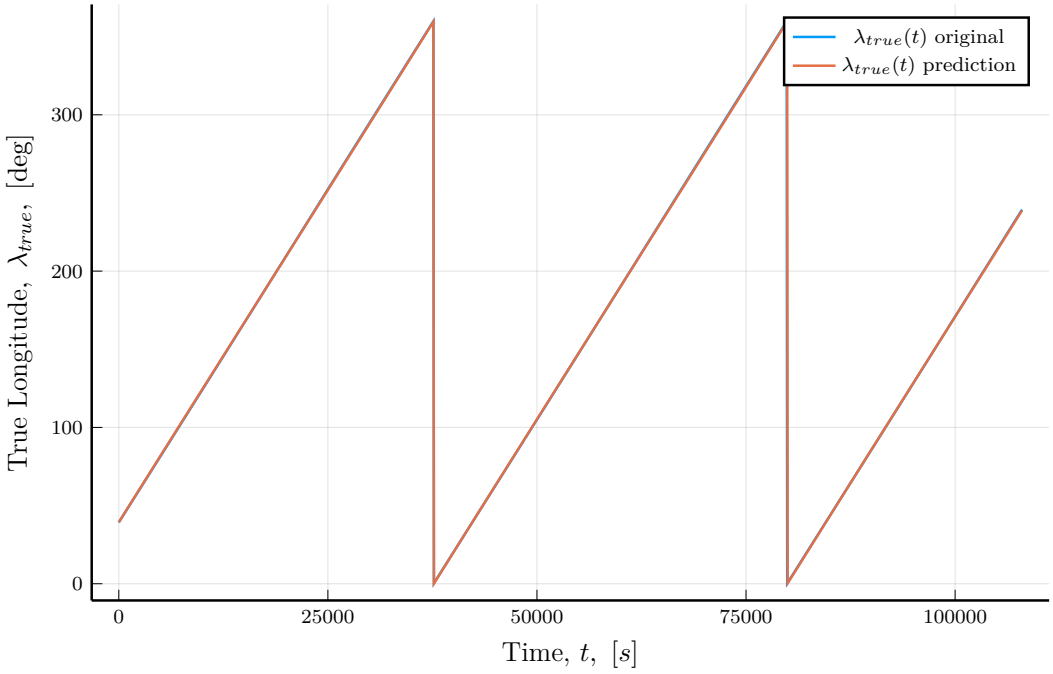


FIGURE 4.14: Original and predicted values of the true longitude for the orbit of Dimorphos for a fit using Keplerian elements

$$e_{min} = \frac{1181.123}{1178.523} - 1 \tag{4.4}$$

Which result in the detection limit for the eccentricity to be:

$$e_{min} = 0.002206151 \tag{4.5}$$

The same problem that was previously faced for the inclination is currently being faced. The removal of the variation of parameters during the fitting process has now added a detection limit for the eccentricity: The algorithm will guess a mean eccentricity value close to the observed mean value (within the detection limit), but the errors presented are completely random since they are below the detection limit for the eccentricity.

Let's see if this hypothesis is true. A single orbit with period T will now be considered, which will be split to 30 smaller intervals. Each interval will have a duration of $T/30$. Furthermore, let's increase the number of photographs taken by Hera to 3000: this is not representative of the Early Characterization phase, but will allow us to ensure that the algorithm has an adequate number of 100 pictures to perform the fit for each interval, as explained in Section 4.2. A J_2 perturbed orbit will then be propagated using the initial state vector found in Table 4.14.

Afterwards, the attempt to fit keplerian unperturbed orbits for each of the 30 intervals will be made, by disabling the J_2 perturbation during the optimization procedure.

TABLE 4.14: Initial state vector used to generate a single perturbed orbit

Orbital Element	Value
a	1178.523 m
e	0.0000078
i	0.0078 °
Ω	0 °
ω	0 °
λ_{true}	42.236 °

The idea is to use the results from the 30 intervals to recreate the perturbed orbit. If each orbital element is plotted as a function of time first for the entire perturbed orbit and then for each of the 30 unperturbed intervals separately, these plots can then be used in combination with the mean absolute performance error to determine if the orbit of Dimorphos can be determined this way.

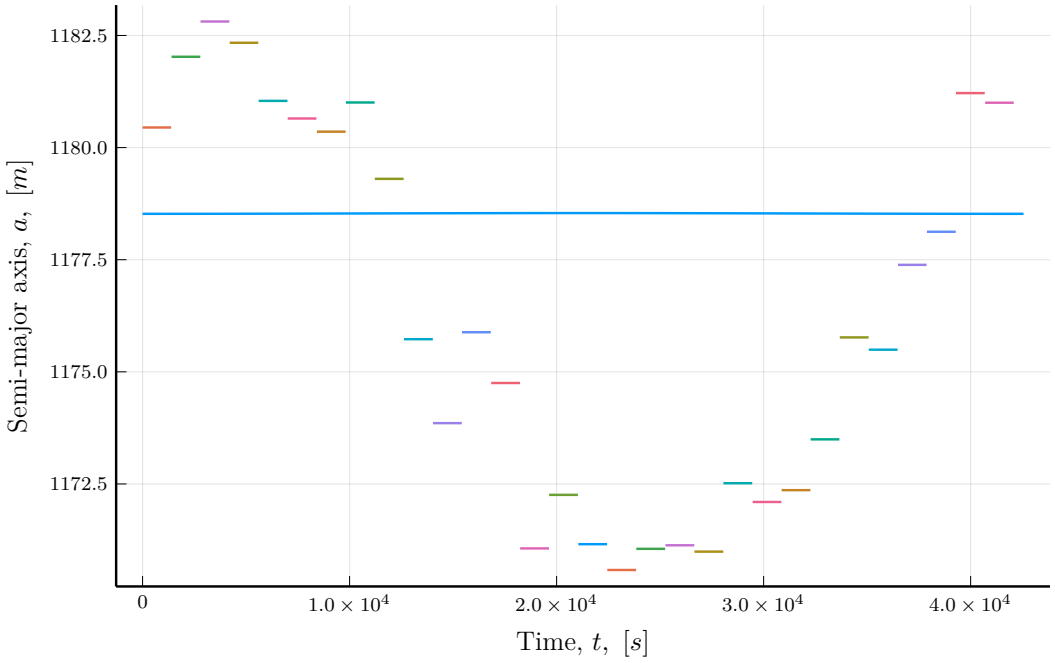


FIGURE 4.15: Semi-major axis as a function of time for one perturbed orbit (blue) and algorithm predicted values for each of the 30 intervals (unperturbed)

TABLE 4.15: Error and mean absolute percentage error for each of the orbital elements and gravitational parameter for the fitting of 30 intervals of keplerian orbital elements generated by the algorithm to a single perturbed orbit

Interval ID	% error a	% error e	% MAPE λ_{true}	% error μ
#1	0.163338	41.82043	0.406921	0.27929
#2	0.297142	285.6975	0.373882	0.306704
#3	0.363857	437.0626	0.037177	0.016027
#4	0.32361	451.7001	0.101099	0.023618
#5	0.213686	275.3576	0.445795	0.317136
#6	0.180359	506.4155	0.172257	0.148147
#7	0.155422	153.175	0.065536	0.019923
#8	0.210668	11.61107	0.57385	0.084445
#9	0.066174	81.07476	0.063499	1.18E-05
#10	0.237213	416.661	0.964563	0.547637
#11	0.396013	512.2132	0.072983	0.070337
#12	0.224043	101.2011	0.255138	0.52436
#13	0.320083	122.0531	0.12956	0.031978
#14	0.633029	428.555	0.176074	0.456872
#15	0.531735	183.3969	0.578246	0.2436
#16	0.625049	184.0258	0.429933	0.243547
#17	0.673864	317.4524	0.165579	0.014806
#18	0.633836	18.29076	0.690484	0.542636
#19	0.627127	158.9479	0.900639	0.86123
#20	0.639112	327.2695	0.459301	0.543674
#21	0.509661	394.2699	0.549015	0.45637
#22	0.545194	120.2425	0.645493	0.384993
#23	0.522804	45.35388	0.418875	0.142632
#24	0.426684	28.07996	0.37768	0.180813
#25	0.233808	186.1909	0.098883	0.67315
#26	0.25707	173.2645	0.34593	0.314486
#27	0.096551	160.1162	0.262752	0.524355
#28	0.033941	175.0147	0.968319	0.58576
#29	0.228399	377.1954	0.589882	0.524366
#30	0.210208	203.8602	0.464373	0.023453

Let's interpret Figure 4.15 to Figure 4.17 using the mean absolute percentage errors and the percentage errors from Table 4.15. It can be immediately observed that the algorithm can accurately measure the gravitational parameter μ, with the percentage error remaining below 1%. For the semi-major axis, the fitting from Figure 4.15 does not appear to be excellent, however the mean absolute percentage of each interval

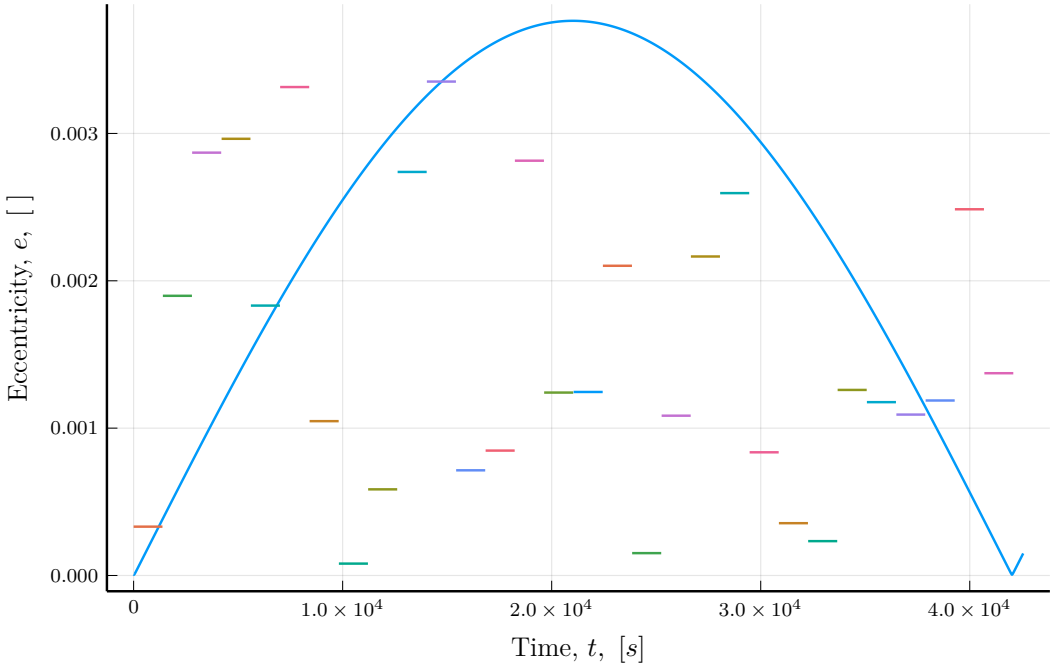


FIGURE 4.16: Eccentricity as a function of time for one perturbed orbit (blue) and algorithm predicted values for each of the 30 intervals (unperturbed)

still remains below 1%. The algorithm seems to be able to correctly identify the true longitude for each one of the 30 intervals, a fact which can be seen in Figure 4.17, with the mean absolute percentage error between the observed and the predicted values of the true longitude remaining below 1%.

Higher mean absolute percentage errors are only observed for the eccentricity. In Figure 4.16 it can be noted that the fitting values do not appear to follow the observed perturbed values for the eccentricity: they do remain however within the detection limit for the eccentricity which were identified before in Equation (4.5) despite the mean absolute percentage errors. This further supports the detection limit hypothesis for the eccentricity, meaning that the attempts to fit it using keplerian orbital elements will only result in random guesses by the algorithm within the detection limit.

4.4 Measuring the mass of Dimorphos using Didymos' wobble

One of the main objectives of Hera presented in Chapter 1 is the measurement of Dimorphos' mass by measuring the wobble of Didymos. This cannot take place during the Early Characterization Phase, due to the fact that the spatial resolution of the camera is greater than the wobble of Didymos, meaning that Didymos appears to be stationary at this distance. However at the Detailed Characterization Phase, Hera will be close enough to the binary asteroid system (at a distance of about 10 km) to

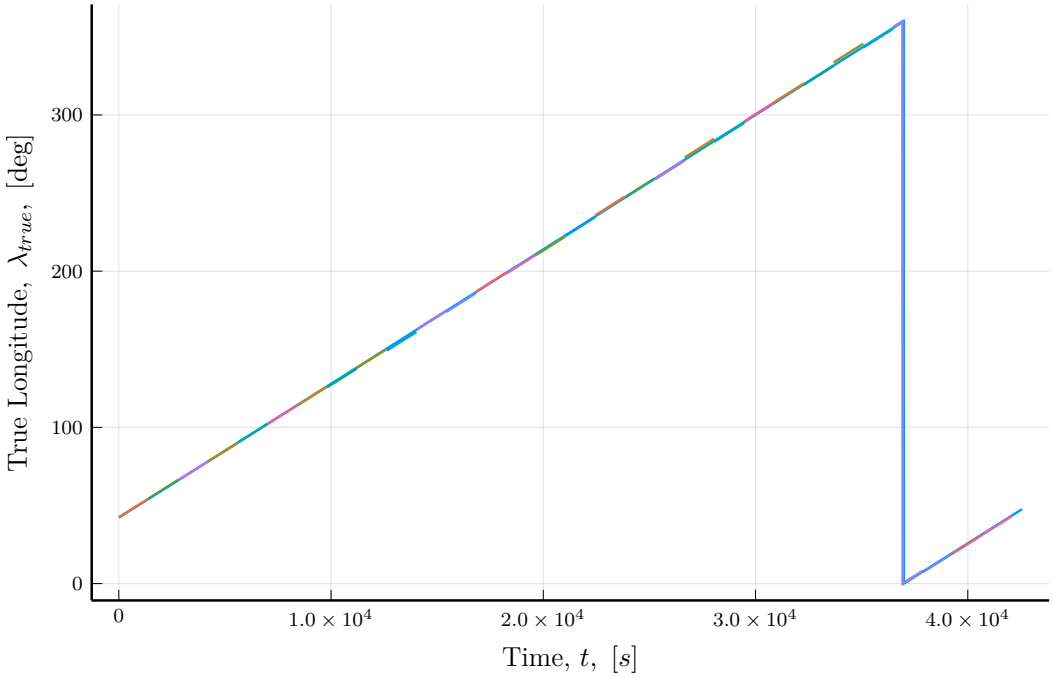


FIGURE 4.17: True longitude as a function of time for one perturbed orbit (blue) and algorithm predicted values for each of the 30 intervals (unperturbed)

enable the measurement of the wobble of Didymos. A strategy to determine the mass of Dimorphos using Didymos' wobble will now be presented.

From the previous sections, it has been determined that we can determine the orbit of Dimorphos with a mean absolute percentage error below 1% and the gravitational parameter of the system with an accuracy below 1%. This guessed initial state vector combined with the gravitational parameter can be propagated beyond the Early Characterization phase to include the Detailed Characterization Phase 1. It should be reminded that from the propagator, we obtain the relative motion as a function of time $r(t)$ and the relative velocities as a function of time $v(t)$. Didymos' wobble will be visible if the barycentric orbit of the body is computed using:

$$\vec{R}_{Didymos}(t) = -\frac{m_{Dimorphos}}{m_{Didymos} + m_{Dimorphos}}\vec{r}(t) \tag{4.6}$$

Since the gravitational parameter of the system was previously determined, Eq. (4.6) can be rewritten as:

$$\vec{R}_{Didymos}(t) = -\frac{Gm_{Dimorphos}}{G(m_{Didymos} + m_{Dimorphos})}\vec{r}(t) = -\frac{Gm_{Dimorphos}}{\mu}\vec{r}(t) \tag{4.7}$$

The methodology used in Chapter 2 will remain largely unchanged, with only some minor changes occuring. The following changes are implemented to the algorithm.

The 300 h observation window is moved from the Early Characterization Phase to the Detailed Characterization Phase. SPICE does include the planned orbit of Hera for all phases, so this change is fairly easy to make. The initial state vector (which is unknown for the second part of the analysis) used to generate the observed images is then propagated to also include the Detailed Characterization Phase. Using Equation (4.7), the position of Didymos is added to the analysis.

A new cost function is then defined:

$$f(m_{Dimorphos}) = SSR_{final} \tag{4.8}$$

Instead of measuring the pixel coordinates of Dimorphos, the pixel coordinates of Didymos are used now. From Equation (4.7), it can be determined that the only missing value to recreate the wobble of Didymos is the mass of Dimorphos, which has to be determined by minimizing the cost function. The predicted initial state vector which was determined by the algorithm before during the early characterization phase is used inside the cost function in combination with a mass value for Dimorphos to create the pixel coordinates of Didymos. The same idea as before is then used: The cost function has to be minimized to determine the actual mass of Dimorphos. The Evolutionary Centers Algorithm will be used again, the only difference is that the new cost function is used and the mass value is constrained to be within 40% of the currently expected value. The percentage error between the actual mass value of Dimorphos used to generate the simulated images and the predicted mass value is then computed and reported. This process is repeated 20 times to ensure the repeatability of results, which are presented in Table 4.16. A demo image of the observed and the predicted pixel coordinates can be found in Figure 4.18, while the actual wobble of Didymos can be seen in Figure 4.19.

From Table 4.16, a very important result is observed. Using the predicted orbit of Dimorphos which was determined during the Early Characterization Phase in combination with the measurement of Didymos' wobble during the Detailed Characterization Phase 1, the mass of Dimorphos can be determined with an accuracy greater than 0.1%. This fact indicates that combining the measurements from the Early Characterization Phase and the Detailed Characterization Phase 1 will achieve the binary asteroid's system orbit determination objective and the determination of Dimorphos' mass objective.

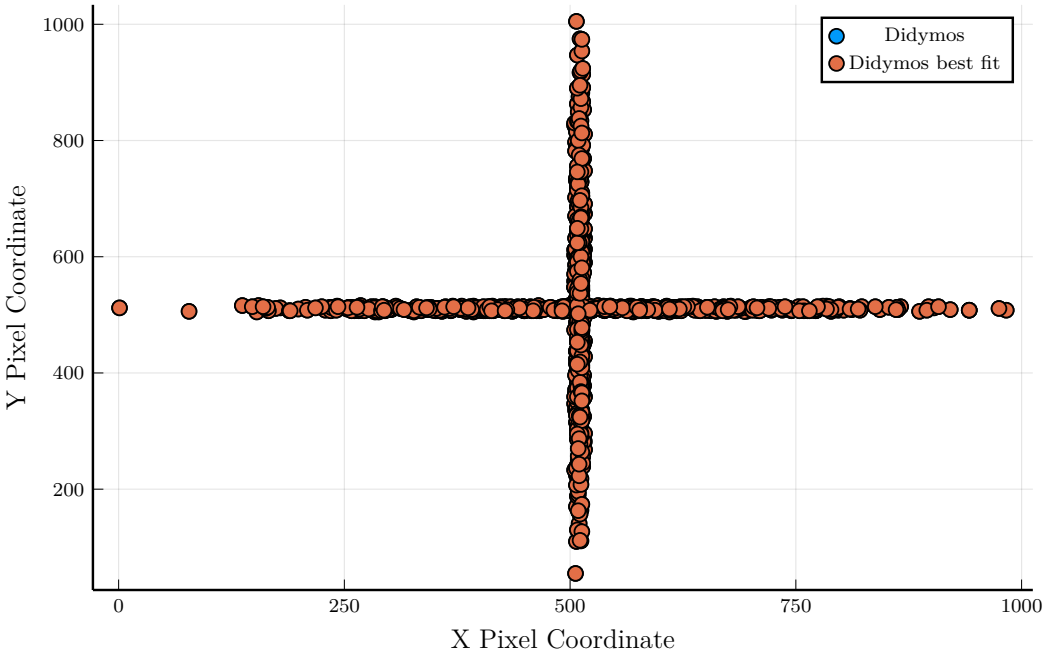


FIGURE 4.18: Observed and predicted pixel coordinates for the wobble of Didymos

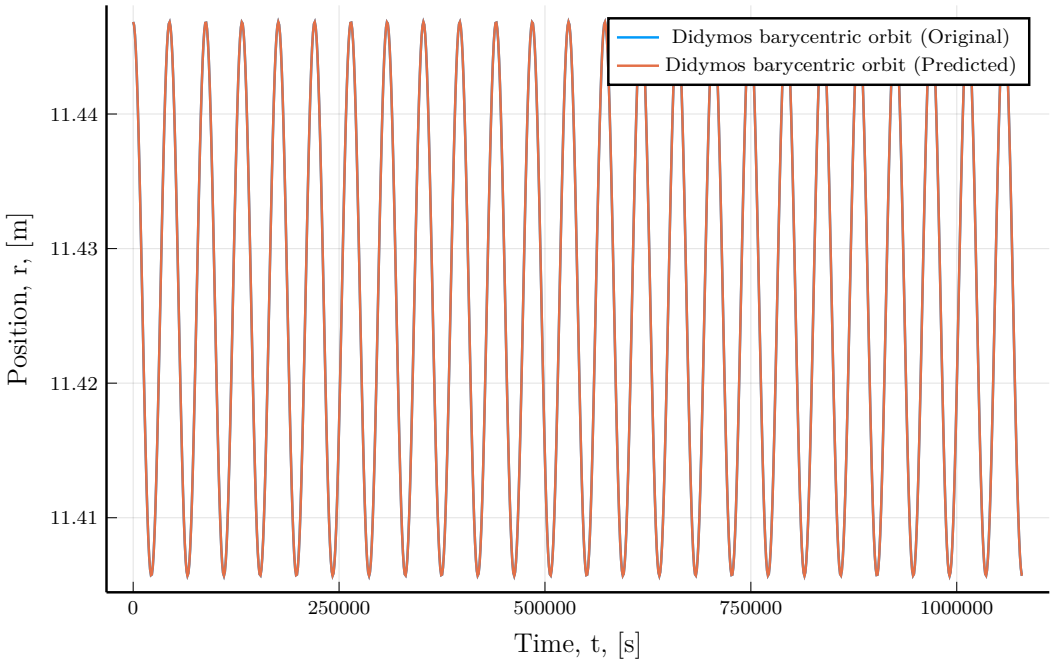


FIGURE 4.19: Observed and predicted position of Didymos with respect to the system's barycenter

TABLE 4.16: Observed and predicted values for the mass of Dimorphos and the percentage error between them for 20 separate runs. Predicted values were obtained using the measurement of Didymos' wobble.

Run ID	**Original** $m_{Dimorphos}$ [kg]	**Predicted** $m_{Dimorphos}$ [kg]	**% error**
#1	4.93267E+09	4.92849E+09	8.47724E-02
#2	4.99116E+09	4.99213E+09	1.95871E-02
#3	4.95488E+09	4.95272E+09	4.36942E-02
#4	4.94969E+09	4.95163E+09	3.92837E-02
#5	5.00108E+09	4.99843E+09	5.29111E-02
#6	4.90242E+09	4.90304E+09	1.25987E-02
#7	4.92256E+09	4.92256E+09	6.49365E-05
#8	4.93900E+09	4.94075E+09	3.54377E-02
#9	4.99729E+09	4.99571E+09	3.15657E-02
#10	4.87740E+09	4.88042E+09	6.17734E-02
#11	4.89311E+09	4.89522E+09	4.32507E-02
#12	4.95629E+09	4.95922E+09	5.92071E-02
#13	4.91093E+09	4.90868E+09	4.58944E-02
#14	4.92653E+09	4.92288E+09	7.42075E-02
#15	4.96495E+09	4.96318E+09	3.56274E-02
#16	4.85187E+09	4.85294E+09	2.21639E-02
#17	4.85187E+09	4.85294E+09	2.21639E-02
#18	5.00751E+09	5.00600E+09	3.00398E-02
#19	4.90195E+09	4.90111E+09	1.71146E-02
#20	5.02406E+09	5.02744E+09	6.72638E-02

Chapter 5

Discussion

A metaheuristic algorithm was used to initially perform orbit determination of Dimorphos orbit around Didymos during Hera's Early Characterization Phase. The orbit determination procedure is based on optimizing a cost function which tests for different initial state vectors and gravitational parameters until it converges to a solution. With this process it has been proven that the initial state vector of Dimorphos' orbit can be predicted with a mean absolute percentage error below 1%, while the gravitational parameter can be predicted with a percentage error below 1%. Furthermore, it has been showcased that the algorithm can work for a wide variety of orbits including Circular Equatorial, Elliptical Equatorial, Circular Inclined and other orbits.

The predicted initial state vector is then used in the Detailed Characterization Phase to attempt to locate the mass of Dimorphos by measuring the wobble of Didymos. The Detailed Characterization Phase is used for this in order for the spatial resolution of the camera to be low enough to measure the wobble of Didymos. A different cost function was defined for this purpose and was optimized using the same metaheuristic algorithm, namely the Evolutionary Centers Algorithm. Here, it has been demonstrated that by using the predicted initial state vector from the Early Characterization Phase, the mass of Dimorphos can be determined with an accuracy below 0.1%.

These results demonstrate that Hera's Asteroid Framing Camera can indeed achieve two of its main objectives, namely to measure the binary systems orbital period and semi-major axis with an accuracy of 1% or better and the measurement of Dimorphos mass by measuring the wobble of Didymos with an accuracy equal or less than 1 m.

It should also be noted that simulated images were used in this thesis. However, the orbit determination procedure and the wobble prediction procedure developed here can be adapted to accommodate the actual observations obtained by Hera fairly easily. Once the observed images are captured by Hera and are downlinked to Earth, the pixel coordinates of Didymos and Dimorphos have to be extracted from the images alongside with the distance of Hera from the two objects. Furthermore, the orbit of Hera can be changed from SPICE to the actual orbit that Hera will perform. With

these two additions, the algorithm should be able to work and produce similar results for the actual images obtained by Hera.

This algorithm can be expanded into the future to include more of Hera's objectives. Most notably, the same principles can be used to measure spin parameters, the rotation of Dimorphos and any irregular rotations that might exist. It is expected that Dimorphos' rotational behavior might be different due to DART's impact, so it is important to see what was the effect of the impact in Dimorphos' rotational behavior. This can be achieved by tracking some specific landmarks on the surface of Dimorphos. By grabbing a sufficient number of images with the tracked landmarks in combination with the orbit determination procedure used for this thesis, results for the rotation of Dimorphos could be extracted.

Appendix A

Validation of RK4 integrator

To validate the integrator developed for this thesis, a comparison is made using the PKEPLER routine developed by David Vallado [27]. The PKEPLER routine also only considers the effects of J_2, so the same integration step size of $dt = 60$ s should result in similar output values for the positions and the velocities of Didymos and Dimorphos. The total integration time chosen here is 300 h, in order to be the same as the analysis time interval. The values of the initial state vector used for this validation can be found in Table A.1 and Table A.2. Results were extracted using the following procedure:

1. Decide for an initial state vector.
2. Use the .m file of the PKEPLER routine to generate the positions and velocities of Dimorphos.
3. Save the generated data in .csv format and import it in Julia.
4. Generate the positions and velocities using the RK4 integrator.
5. Plot the comparison between the PKEPLER routine and the RK4 integrator.
6. Compute the mean absolute percentage error for the positions and velocities between the two methods.

The source code used for this procedures can be found in Appendix B.4. Mean absolute percentage errors between the two methods can be found in Table A.3. Based on the results presented here, it can be concluded that the RK4 integrator performs nominally and can be used for the purposes of this thesis.

TABLE A.1: Initial state used for integrator validation (positions)

Position	**Value** [m]
x	-1172.5872046095626
y	161.0414473856286
z	0.09837463435334828

TABLE A.2: Initial state used for integrator validation (velocities)

Velocity	**Value** [$\mathrm{m\,s^{-1}}$]
v_x	-0.023675517665779577
v_y	-0.17238774059049633
v_z	-0.00010530569131672698

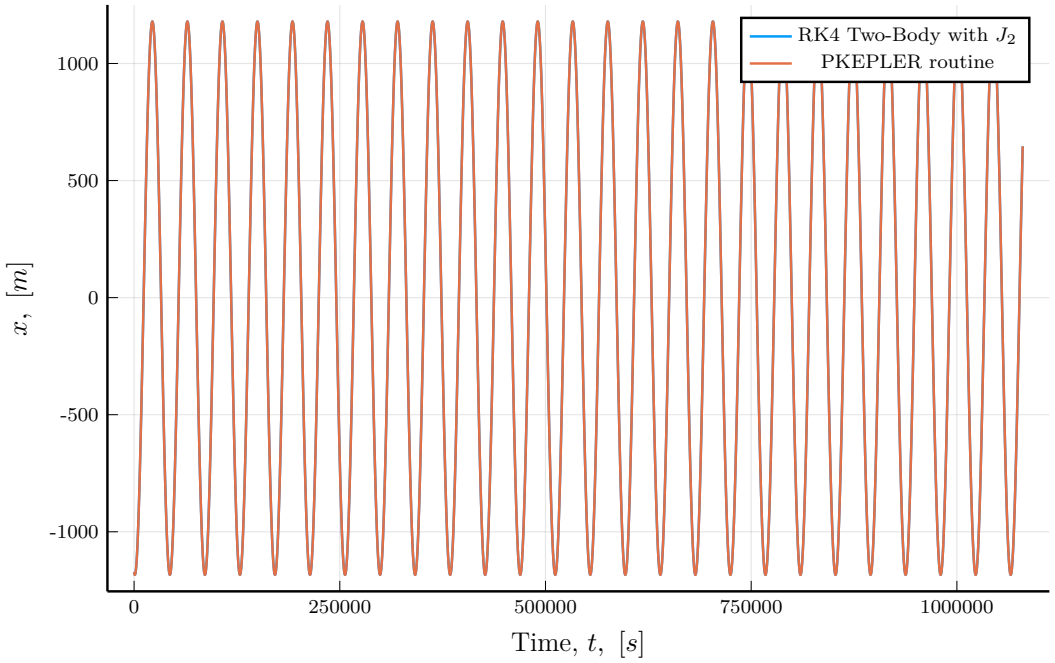


FIGURE A.1: Comparison between the developed RK4 integrator and the PKEPLER routine for the x position of Didymos

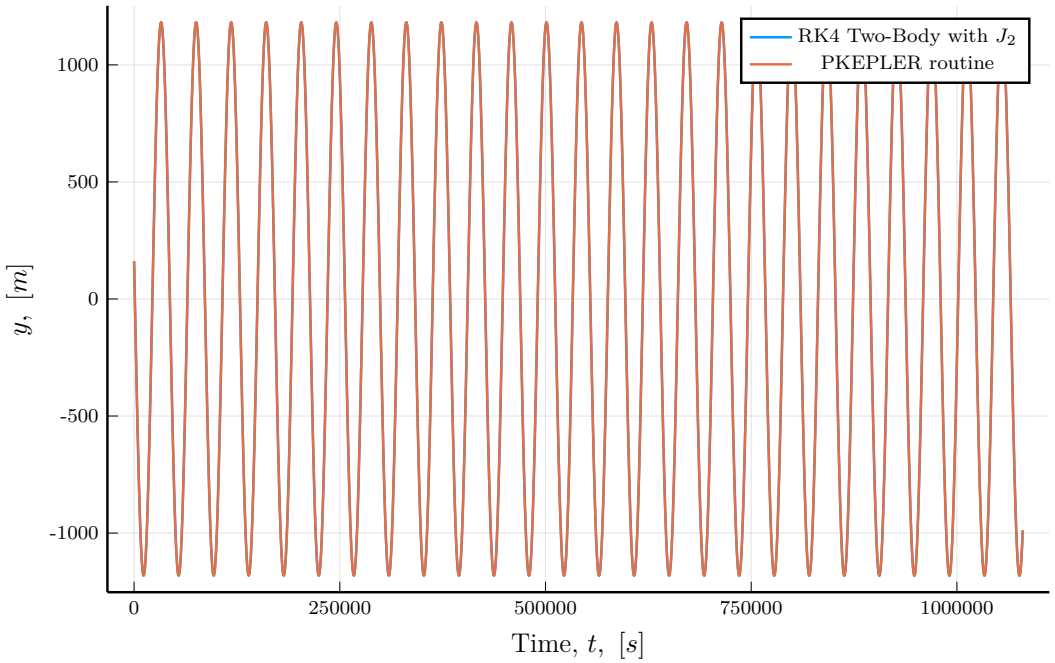


FIGURE A.2: Comparison between the developed RK4 integrator and the PKEPLER routine for the y position of Dimorphos

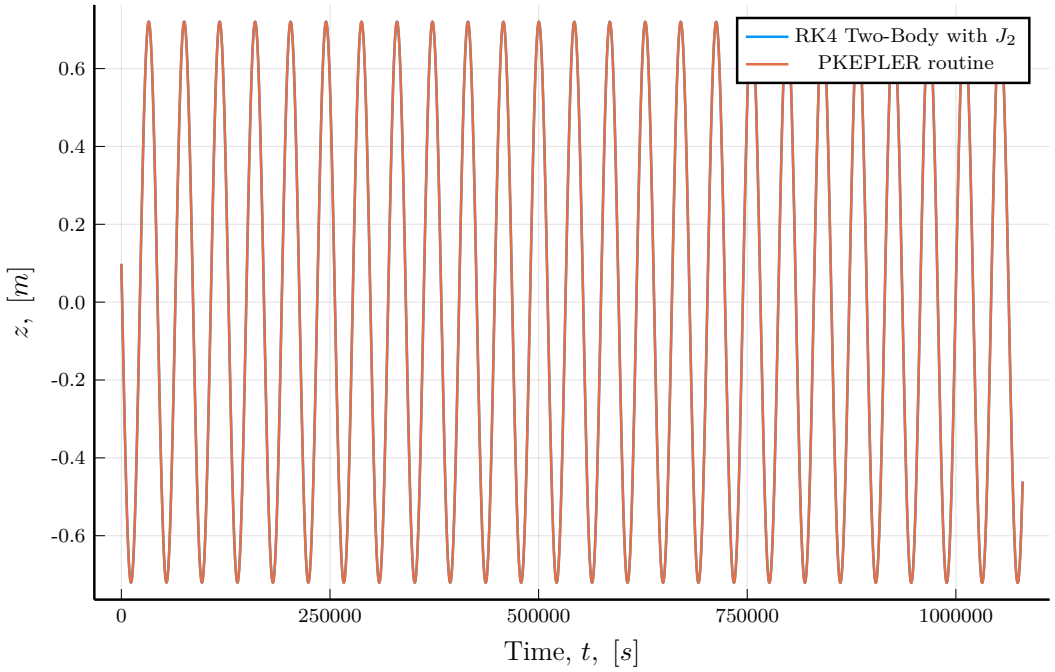


FIGURE A.3: Comparison between the developed RK4 integrator and the PKEPLER routine for the z position of Dimorphos

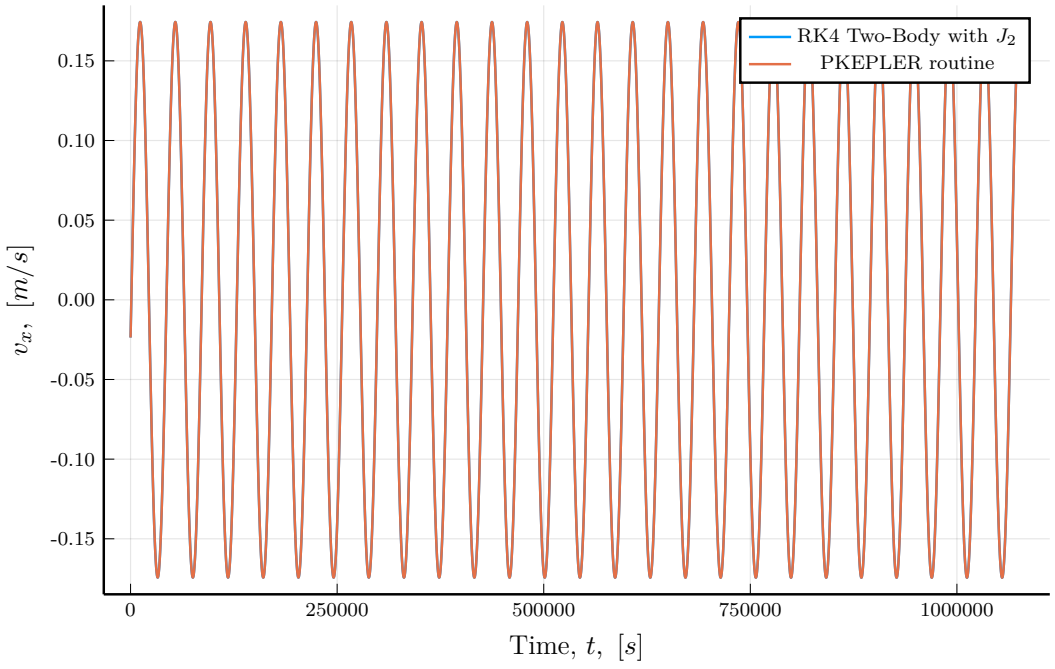


FIGURE A.4: Comparison between the developed RK4 integrator and the PKEPLER routine for the v_x velocity of Dimorphos

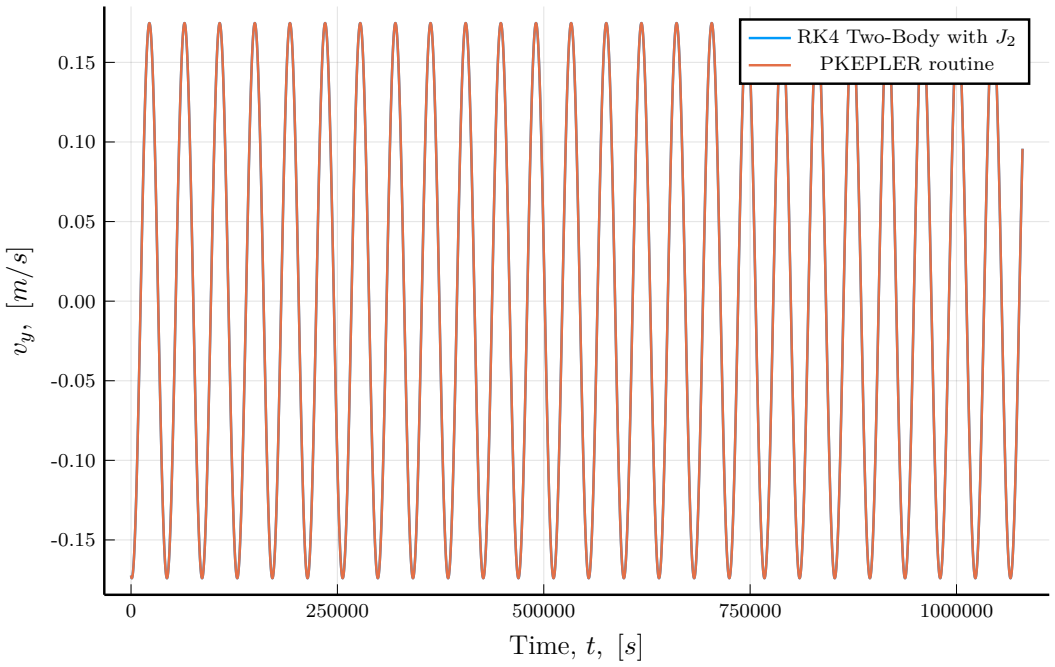


FIGURE A.5: Comparison between the developed RK4 integrator and the PKEPLER routine for the v_y velocity of Dimorphos

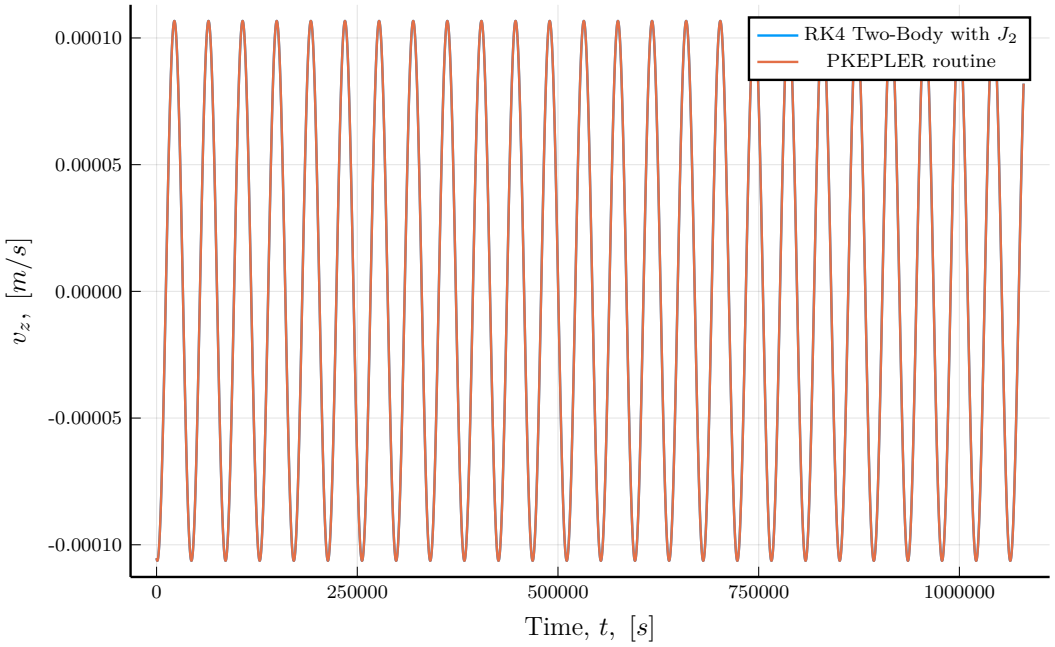


FIGURE A.6: Comparison between the developed RK4 integrator and the PKEPLER routine for the v_z velocity of Dimorphos

TABLE A.3: Mean absolute percentage error for each position and velocity

Parameter	**MAPE [%]**
x	5.45072E-06
y	5.33749E-06
z	6.02866E-06
v_x	3.44891E-06
v_y	1.74857E-06
v_z	5.99215E-07

Appendix B

Source Code

The source code used for this thesis is provided with an open source license at the author's Gitlab Repository. Code developed as part of this thesis can be found here: https://gitlab.com/retse/hera-orbit-determination. A few essential code samples are provided in the following sections, readers are however encouraged to visit the link to get the complete code if they wish to run the analysis on their own.

B.1 camera_spice_experiment.jl

```
using SPICE, ProgressBars, Plots, LinearAlgebra, Distributions, Random,
 ↪  DifferentialEquations, Metaheuristics, LaTeXStrings
include("orbital_utilities.jl")
include("propagators.jl")
include("spice_utilities.jl")
include("camera_utilities.jl")
include("optimization.jl")
include("results.jl")
include("plotting.jl")


function main()
        # Dimorphos orbit
        global a_dimorphos = rand(Uniform(1160, 1220)) # Semi-major axis
        global e_dimorphos = rand(Uniform(0.0000001, 0.00001)) #
        ↪  Eccentricity
        global i_dimorphos = rand(Uniform(0.00000001, 0.00001)) #
        ↪  Inclination
        global Omega_dimorphos = 0.0 # Longitude of the ascending node
        global omega_dimorphos = 0.0 # Argument of periapsis
        global M_dimorphos = rand(Uniform(0.0, 359.99)) # True anomaly

        # Determine what kind of orbit we are dealing with
        global dimorphos_orbit_type = type_of_orbit(e_dimorphos,
        ↪  i_dimorphos)
```

```
22          println("Orbit type is "*dimorphos_orbit_type)
23
24          # Compute initial position and velocity vector from orbital elements
25          r_vector, v_vector = orbital_elements_to_cartesian(a_dimorphos,
            ↪  e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos,
            ↪  M_dimorphos, mu_system)
26          x = r_vector[1]
27          y = r_vector[2]
28          z = r_vector[3]
29          vx = v_vector[1]
30          vy = v_vector[2]
31          vz = v_vector[3]
32
33          # Propagate Dimorphos
34          x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos,
            ↪  vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz,
            ↪  mu_system, start_time, end_time, total_photos,
            ↪  enable_perturbation)
35          # Select every xth element to match the photos taken and convert to
            ↪  km
36          photo_selector = LinRange(1, length(x_dimorphos), total_photos)
37          index_vector = zeros(Int64, total_photos)
38          for i = 1:length(index_vector)
39          index_vector[i] = floor(Int64, photo_selector[i])
40          end
41          x_dimorphos = x_dimorphos[index_vector]
42          y_dimorphos = y_dimorphos[index_vector]
43          z_dimorphos = z_dimorphos[index_vector]
44          vx_dimorphos = vx_dimorphos[index_vector]
45          vy_dimorphos = vy_dimorphos[index_vector]
46          vz_dimorphos = vz_dimorphos[index_vector]
47          t_vector = t_vector[index_vector]
48          dimorphos_coordinates = hcat(x_dimorphos/1000, y_dimorphos/1000,
            ↪  z_dimorphos/1000)
49
50          # Orbit start and end time (for SPICE)
51          spice_end_time = spice_start_time + end_time
52
53          # Initialize position arrays (x, y, z)
54          didymos_coordinates = zeros(Float64, total_photos, 3)
55          barycenter_coordinates = zeros(Float64, total_photos, 3)
56          barycenter_offset = zeros(Float64, 3)
57          barycenter_initial_coordinates = zeros(Float64, 3)
58          camera_position_didymos = zeros(Float64, total_photos, 3)
59          camera_position_dimorphos = zeros(Float64, total_photos, 3)
60          didymos_pixel_coordinates = zeros(Float64, total_photos, 2)
61          dimorphos_pixel_coordinates = zeros(Float64, total_photos, 2)
62
```

```
63          # HERA_AFC-1 coordinate system unit vectors
64          e_x = [1.0, 0.0, 0.0]
65          e_y = [0.0, 1.0, 0.0]
66          e_z = [0.0, 0.0, 1.0]
67
68          # Main SPICE calculations loop
69          focal_length = 10.6*10^-5 # [km]
70          iteration = 1
71          println("\nSPICE calculations:")
72          for current_time in ProgressBar(LinRange(start_time, end_time,
            ↪  total_photos))
73                  i = spice_start_time + current_time
74                  position_didymos, lt = spkpos("-658030", i, "J2000", "None",
                    ↪  "-999")
75                  position_system_barycenter, lt = spkpos("2065803", i,
                    ↪  "J2000", "None", "-999")
76                  position_dimorphos = dimorphos_coordinates[iteration, :]
77                  # Get didymos reference position or translate
                    ↪  hera_coordinates
78                  for j in 1:3
79                  barycenter_offset[j] = rand(barycenter_error_distribution) +
                    ↪  rand(hera_error_distribution)
80                  end
81                  # Apply offset to all coordinates
82                  for j in 1:3
83                  barycenter_coordinates[iteration, j] =
                    ↪  position_system_barycenter[j] + barycenter_offset[j]
84                  didymos_coordinates[iteration, j] = position_didymos[j] +
                    ↪  barycenter_offset[j]
85                  dimorphos_coordinates[iteration, j] =
                    ↪  barycenter_coordinates[iteration, j] +
                    ↪  position_dimorphos[j]
86                  end
87                  # Rotate HERA_AFC-1 camera reference frame assuming it is
                    ↪  always tracking the barycenter
88                  rotation_matrix =
                    ↪  compute_rotation_matrix(barycenter_coordinates[iteration,
                    ↪  :], e_z)
89                  barycenter_coordinates[iteration, :] = rotation_matrix *
                    ↪  barycenter_coordinates[iteration, :]
90                  didymos_coordinates[iteration, :] =  rotation_matrix *
                    ↪  didymos_coordinates[iteration, :]
91                  dimorphos_coordinates[iteration, :] =  rotation_matrix *
                    ↪  dimorphos_coordinates[iteration, :]
92                  # Add pointing error
93                  pointing_error_matrix =
                    ↪  error_rotation_matrix(random_error[iteration],
                    ↪  random_axis[iteration])
```

```
94              camera_position_didymos[iteration, :] = pointing_error_matrix
                ↪  * didymos_coordinates[iteration, :]
95              camera_position_dimorphos[iteration, :] =
                ↪  pointing_error_matrix * dimorphos_coordinates[iteration,
                ↪  :]
96              # Store pixel coordinates for Didymos and Dimorphos
97              for j in 1:2
98              didymos_pixel_coordinates[iteration, j] =
                ↪  focal_length*camera_position_didymos[iteration,
                ↪  j]/camera_position_didymos[iteration, 3]
99              dimorphos_pixel_coordinates[iteration, j] =
                ↪  focal_length*camera_position_dimorphos[iteration,
                ↪  j]/camera_position_dimorphos[iteration, 3]
100             end
101             iteration += 1
102         end
103
104         # Compute boundary vectors/points for the camera
105         boresight_vector = barycenter_coordinates[iteration-1, :]
106         rotation_matrix = compute_rotation_matrix(boresight_vector, e_z)
107         camera_id = -999110
108         boundary_vectors_matrix = get_boundary_vectors(camera_id)
109         line_generator = LinRange(0, 30, 10)
110
111         # Rotate camera boundary vectors to assume camera is always tracking
            ↪  barycenter
112         for i in 1:4
113         result = rotation_matrix * boundary_vectors_matrix[i, :]
114         for j in 1:3
115         boundary_vectors_matrix[i,j] = result[j]
116         end
117         end
118
119         # Plot 3D image for reference
120         pgfplotsx()
121         x_didymos, y_didymos, z_didymos = didymos_coordinates[:, 1],
            ↪  didymos_coordinates[:, 2], didymos_coordinates[:, 3]
122         x_dimorphos, y_dimorphos, z_dimorphos = dimorphos_coordinates[:, 1],
            ↪  dimorphos_coordinates[:, 2], dimorphos_coordinates[:, 3]
123         plt_3d = scatter3d(x_didymos, y_didymos, z_didymos, color = "blue",
            ↪  xlabel="x [km]", ylabel="y [km]", zlabel = "z [km]",
            ↪  label="Didymos", markersize = 1.5)
124         scatter3d!(plt_3d, x_dimorphos, y_dimorphos, z_dimorphos, color =
            ↪  "orange", markersize = 1.5, label = "Dimorphos (RK4)")
125
126         # Compute and plot camera boundary lines
127         sensor_boundary_points = zeros(Float64, 4, 3)
128         global x_boundaries = zeros(Float64, 4)
```

```
129         global y_boundaries = zeros(Float64, 4)
130         for i in 1:4
131         x_line, y_line, z_line = boundary_vectors_matrix[i, 1] *
            ↪  line_generator, boundary_vectors_matrix[i, 2] * line_generator,
            ↪  boundary_vectors_matrix[i, 3] * line_generator
132         sensor_boundary_points[i, 1] = x_line[length(x_line)]
133         sensor_boundary_points[i, 2] = y_line[length(y_line)]
134         sensor_boundary_points[i, 3] = z_line[length(z_line)]
135         x_boundaries[i] =
            ↪  focal_length*x_line[length(x_line)]/z_line[length(z_line)]
136         y_boundaries[i] =
            ↪  focal_length*y_line[length(y_line)]/z_line[length(z_line)]
137         camera_label = "Camera boundary line " * string(i)
138         #plot3d!(plt_3d, x_line, y_line, z_line, color="green",
            ↪  label=camera_label)
139         end
140
141         # Compute coordinates in pixels
142         x_pixel_didymos, y_pixel_didymos =
            ↪  convert_to_pixels(didymos_pixel_coordinates[:, 1],
            ↪  didymos_pixel_coordinates[:, 2], x_boundaries, y_boundaries)
143         x_pixel_dimorphos, y_pixel_dimorphos =
            ↪  convert_to_pixels(dimorphos_pixel_coordinates[:, 1],
            ↪  dimorphos_pixel_coordinates[:, 2], x_boundaries, y_boundaries)
144         x_pixel_boundaries, y_pixel_boundaries =
            ↪  convert_to_pixels(x_boundaries,y_boundaries, x_boundaries,
            ↪  y_boundaries)
145
146         # Reject images where we cannot see one object
147         for i in 1:length(x_pixel_didymos)
148                 if ismissing(x_pixel_didymos[i])
                    ↪  ismissing(x_pixel_dimorphos[i])
149                         x_pixel_didymos[i] = missing
150                         y_pixel_didymos[i] = missing
151                         x_pixel_dimorphos[i] = missing
152                         y_pixel_dimorphos[i] = missing
153                 end
154         end
155
156         # Drop images which are to be randomly dropped
157         for i in 1:length(dropped_images_index)
158                 x_pixel_didymos[dropped_images_index[i]] = missing
159                 y_pixel_didymos[dropped_images_index[i]] = missing
160                 x_pixel_dimorphos[dropped_images_index[i]] = missing
161                 y_pixel_dimorphos[dropped_images_index[i]] = missing
162         end
163
164         # Add centroid pixel error
```

```
165         global x_pixel_dimorphos = x_pixel_dimorphos .+
            ↪  x_centroid_pixel_error
166         global y_pixel_dimorphos = y_pixel_dimorphos .+
            ↪  y_centroid_pixel_error
167
168         usable_photos = length(collect(skipmissing(x_pixel_dimorphos)))
169         println("Usable photos: " *string(usable_photos))
170
171         # Perform orbit determination
172         if dimorphos_orbit_type == "Normal"
173                 minimum_error = residuals([a_dimorphos e_dimorphos
                    ↪  i_dimorphos Omega_dimorphos omega_dimorphos M_dimorphos
                    ↪  mu_system])
174                 println("Minimum error is " *string(minimum_error)* "
                    ↪  pixels")
175                 # Full OD problem, 6 orbital elements
176                 bounds = [1190.0-30 0.0001 0.0 0.0 0.0 0.0
                    ↪  0.8*mu_system_original;
177                 1190.0+30 0.9 90.0 359.99 359.99 359.99
                    ↪  1.2*mu_system_original]
178                 # Minimize square mean error to find best orbital elements
179                 result = optimize(residuals, bounds, ECA(information =
                    ↪  Information(f_optimum = minimum_error), options =
                    ↪  Options(debug=true, iterations=100, store_convergence =
                    ↪  true)))
180                 # Extract final guess from optimization
181                 a_final = minimizer(result)[1]
182                 e_final = minimizer(result)[2]
183                 i_final = minimizer(result)[3]
184                 Omega_final = minimizer(result)[4]
185                 omega_final = minimizer(result)[5]
186                 M_final = minimizer(result)[6]
187                 mu_final = minimizer(result)[7]
188                 elseif dimorphos_orbit_type == "EEO"
189                 minimum_error = residuals([a_dimorphos e_dimorphos
                    ↪  omega_dimorphos M_dimorphos mu_system])
190                 println("Minimum error is " *string(minimum_error)* "
                    ↪  pixels")
191         # Elliptical Equatorial orbit, OD with 5 orbital elements
192                 # Bounds for optimization
193                 bounds = [1190.0-30 0.00000001 0.0 0.0
                    ↪  0.8*mu_system_original;
194                 1190.0+30 0.9 359.9 359.99 1.2*mu_system_original]
195                 result = optimize(residuals, bounds, ECA(information =
                    ↪  Information(f_optimum = minimum_error), options =
                    ↪  Options(debug=true, iterations=100, store_convergence =
                    ↪  true)))
196                 # Extract final guess from optimization
```

```
197             a_final = minimizer(result)[1]
198             e_final = minimizer(result)[2]
199             i_final = i_dimorphos
200             Omega_final = Omega_dimorphos
201             omega_final = minimizer(result)[3]
202             M_final = minimizer(result)[4]
203             mu_final = minimizer(result)[5]
204             elseif dimorphos_orbit_type == "CIO"
205             minimum_error = residuals([a_dimorphos e_dimorphos
                ↪ i_dimorphos Omega_dimorphos M_dimorphos mu_system])
206             println("Minimum error is " *string(minimum_error)* "
                ↪ pixels")
207         # Circular inclined orbit, OD with 5 orbital elements
208             # Bounds for optimization
209             result = optimize(residuals, bounds, ECA(information =
                ↪ Information(f_optimum = minimum_error), options =
                ↪ Options(debug=true, iterations=100, store_convergence =
                ↪ true)))
210             # Extract final guess from optimization
211             a_final = minimizer(result)[1]
212             e_final = minimizer(result)[2]
213             i_final = minimizer(result)[3]
214             Omega_final = minimizer(result)[4]
215             omega_final = omega_dimorphos
216             M_final = minimizer(result)[5]
217             mu_final = minimizer(result)[6]
218             elseif dimorphos_orbit_type == "CEO"
219             minimum_error = residuals([a_dimorphos e_dimorphos
                ↪ M_dimorphos mu_system])
220             println("Minimum error is " *string(minimum_error)* "
                ↪ pixels")
221         # Circular equatorial orbit, OD with 4 orbital elements
222             # Bounds for optimization
223             bounds = [1190.0-30 0.0000001 0.0 0.8*mu_system_original;
224             1190.0+30 0.00001 359.99 1.2*mu_system_original]
225             result = optimize(residuals, bounds, ECA(information =
                ↪ Information(f_optimum = minimum_error), options =
                ↪ Options(debug=true, iterations=100, store_convergence =
                ↪ true)))
226             # Extract final guess from optimization
227             a_final = minimizer(result)[1]
228             e_final = minimizer(result)[2]
229             i_final = i_dimorphos
230             Omega_final = Omega_dimorphos
231             omega_final = omega_dimorphos
232             M_final = minimizer(result)[3]
233             mu_final = minimizer(result)[4]
234         end
```

```
235
236         # Plot performance
237         f_calls, best_f_values = convergence(result)
238         plt_performance = plot(1:length(f_calls), best_f_values,
            ↪  xlabel="Generation", ylabel="Residuals function value, [pixels]",
            ↪  label="ECA", yaxis=:log, widen=true, formatter=:plain, legend =
            ↪  :topright)
239         savefig(plt_performance, ".\\Results\\performance_plot.pdf")
240
241         # Save orbital elements plots vs time
242         original_orbital_elements = [a_dimorphos, e_dimorphos, i_dimorphos,
            ↪  Omega_dimorphos, omega_dimorphos, M_dimorphos]
243         fitted_orbital_elements = [a_final, e_final, i_final, Omega_final,
            ↪  omega_final, M_final]
244         propagate_and_plot_orbital_elements_vs_time(original_orbital_elements,
            ↪  fitted_orbital_elements)
245
246         # Compute percentage error of the predicted orbit relative to the
            ↪  initial orbit
247         initial_elements = [a_dimorphos, e_dimorphos, i_dimorphos,
            ↪  Omega_dimorphos, omega_dimorphos, M_dimorphos]
248         final_elements = [a_final, e_final, i_final, Omega_final,
            ↪  omega_final, M_final, mu_final]
249         println(initial_elements)
250         println(final_elements)
251         compute_mean_absolute_percentage_error(initial_elements,
            ↪  final_elements)
252         GM_percentage_error =
            ↪  100*(mu_system_original-mu_final)/mu_system_original
253         println("Error on GM_system is " * string(GM_percentage_error) * "
            ↪  %")
254
255         # Print number of usable photos
256         println("Total photos: " * string(total_photos))
257         println("Usable photos: " * string(usable_photos))
258         # Compute pixel and 3D points from dimorphos as a result of the
            ↪  optimization
259         x_pixel_fit_dimorphos, y_pixel_fit_dimorphos =
            ↪  propagate_and_compute_dimorphos_pixel_points(a_final, e_final,
            ↪  i_final, Omega_final, omega_final, M_final, start_time, end_time,
            ↪  step_size, spice_start_time, mu_final)
260         dimorphos_fit_coordinates =
            ↪  propagate_and_compute_dimorphos_3D_points(a_final, e_final,
            ↪  i_final, Omega_final, omega_final, M_final, start_time, end_time,
            ↪  step_size, spice_start_time, mu_final)
261
262         # Don't plot fit 3D points which don't exist in photos
```

```
263         x_fit_dimorphos = Array{Union{Missing, Float64}, 1}(missing,
            ↪  total_photos)
264         z_fit_dimorphos = Array{Union{Missing, Float64}, 1}(missing,
            ↪  total_photos)
265         y_fit_dimorphos = Array{Union{Missing, Float64}, 1}(missing,
            ↪  total_photos)
266         for i=1:size(dimorphos_coordinates)[1]
267                 x_fit_dimorphos[i] = dimorphos_fit_coordinates[i, 1]
268                 y_fit_dimorphos[i] = dimorphos_fit_coordinates[i, 2]
269                 z_fit_dimorphos[i] = dimorphos_fit_coordinates[i, 3]
270         end
271         for i in 1:length(x_pixel_dimorphos)
272                 if ismissing(x_pixel_fit_dimorphos[i])
                    ↪  ismissing(y_pixel_fit_dimorphos[i])
273                 x_fit_dimorphos[i] = missing
274                 y_fit_dimorphos[i] = missing
275                 z_fit_dimorphos[i] = missing
276         end
277 end
278
279 scatter3d!(plt_3d, x_fit_dimorphos, y_fit_dimorphos, z_fit_dimorphos, color
    ↪  = "purple", markersize = 1.5, label = "Dimorphos (Final guess)")
280
281 # Plot 2D image simulation (camera plane)
282 plt_2d = scatter(didymos_pixel_coordinates[:, 1],
    ↪  didymos_pixel_coordinates[:, 2], label= "Didymos")
283 scatter!(dimorphos_pixel_coordinates[:, 1], dimorphos_pixel_coordinates[:,
    ↪  2], label = "Dimorphos")
284 scatter!(x_boundaries, y_boundaries, label = "Camera boundaries")
285 savefig(plt_3d, ".\\Results\\3d_plot.pdf")
286
287 # Plot image using pixels
288 plt_pixel = scatter(x_pixel_didymos, y_pixel_didymos, label = "Didymos",
    ↪  widen=true, formatter=:plain, legend = :topright, xlabel = "X Pixel
    ↪  Coordinate", ylabel = "Y Pixel Coordinate")
289 scatter!(x_pixel_dimorphos, y_pixel_dimorphos, label = "Dimorphos")
290 scatter!(x_pixel_fit_dimorphos, y_pixel_fit_dimorphos, label = "Dimorphos
    ↪  best fit")
291 scatter!(x_pixel_boundaries, y_pixel_boundaries, label= "Pixel boundaries")
292 savefig(plt_pixel, ".\\Results\\pixel_error_example.pdf")
293 display(plt_pixel)
294 end
295
296 load_hera_spice_kernels()
297
298 # System properties
299 G = 6.67430*10^-11
300 mass_error = 0.02 # in percentage
```

```
mass_error_distribution = Uniform(-mass_error, mass_error)
mass_didymos = 5.32*10^11 + rand(mass_error_distribution)*5.32*10^11
global radius_didymos = 390 # [m]
global J2_didymos = 0.011432722
global mass_dimorphos = 4.94*10^9 + rand(mass_error_distribution)*4.94*10^9
global mu_system = G*(mass_didymos+mass_dimorphos)
global mu_system_original = mu_system
global c = 3.0*10^8
global enable_perturbation = true

# Time properties
hour = 3600.0
number_of_orbits = 25
# Photos to be taken
photos_per_orbit = 40
global total_photos = photos_per_orbit * number_of_orbits
global start_time = 0.0
global end_time = 300*hour
global step_size = floor((end_time-start_time)/total_photos)
global spice_start_time = utc2et("2027-01-28T08:14:58")
global flux = 1358

# Errors definition
# Centroid pixel error
pixel_error = 4
pixel_error_distribution = Uniform(-pixel_error, pixel_error)
global x_centroid_pixel_error = Int64(round(rand(pixel_error_distribution)))
global y_centroid_pixel_error = Int64(round(rand(pixel_error_distribution)))
# Barycenter position error
global error_barycenter = 10/1000 # [km]
global barycenter_error_distribution = Normal(0.0, error_barycenter)
# HERA position error
global error_hera_position = 10/1000 # [km]
global hera_error_distribution = Normal(0.0, error_hera_position)
# HERA pointing error
global pointing_error = deg2rad(1)
global pointing_error_distribution = Normal(0.0, pointing_error)
global random_error = rand(pointing_error_distribution, total_photos)
global random_axis = rand(1:2, total_photos)
# Randomly dropped images
number_of_dropped_images = Int64(round(0.04*total_photos))
dropped_image_distribution = Uniform(1, total_photos)
global dropped_images_index = zeros(Int64, number_of_dropped_images)
for p=1:number_of_dropped_images
dropped_images_index[p] = Int64(round(rand(dropped_image_distribution)))
end
main()

```

```
# Free used kernels
kclear()
```

B.2 orbital_utilities.jl

```
function eccentric_anomaly_calculator(mean_anomaly, e)
        #=
        Compute E using Newton-Raphson
        Input:
        mean_anomaly, [rad]
        e, eccentricity []
        Output
        E, eccentric anomaly at desired time [rad]
        =#

        # Define initial value for E_0)
        if mean_anomaly < pi
        E_0 = mean_anomaly - e
        else
        E_0 = mean_anomaly + e
        end
        # Define f and f dot
        f(E) = mean_anomaly - E + e*sin(E)
        fdot(E) =  -1 + e*cos(E)
        # Stopping criteria
        N = 15  # Number of significant digits to be computed
        max_repetitions = 1000000
        es = 0.5 * 10^(2.0 - N)  # Scarborough Criterion
        ea = 100
        E_prev = E_0
        repetitions = 0
        # Main Newton-Raphson loop
        while ea > es
        repetitions = repetitions + 1
        E_next = E_prev - (f(E_prev) / fdot(E_prev))
        if E_next == 0
        return E_next
        end
        ea = abs((E_next - E_prev) * 100 / E_next)
        E_prev = E_next
        if repetitions > max_repetitions
        error("Max repetitions reached without achieving desired accuracy for
        ↪  E!")
        end
        end
```

```
        E = E_prev
        return E
end


function orbital_elements_to_cartesian(a, e, i, Omega, omega, ni, mu)
        #=
        Computes cartesian position and velocity vector given some orbital elements
        Input:
        a [m]
        e []
        i [deg]
        Omega [deg]
        omega [deg]
        ni [deg]
        mu [m^3/(kg*s^2)] (GM)
        Output:
        r_vector, v_vector
        =#

        if e < 10^-5
        if i < 10^-1
        # Circular equatorial orbit
        Omega = 0.0
        omega = 0.0
        else
        # Circular inclined orbit
        omega = 0.0
        end
        # In this case, p=a
        p = a
        else
        if i < 10^-1
        # Elliptical equatorial orbit
        Omega = 0.0
        end
        p = a*(1-e^2)
        end
        r_pqw = [(p*cosd(ni))/(1+e*cosd(ni)); (p*sind(ni))/(1+e*cosd(ni)); 0.0]
        v_pqw = [-sqrt(mu/p)*sind(ni); sqrt(mu/p)*(e+cosd(ni)); 0.0]
        transformation = [cosd(Omega)*cosd(omega)-sind(Omega)*sind(omega)*cosd(i) -cosd(Omega)*sind(omega)-sind(Omega)*cosd(omega)*cosd(i) sind(Omega)*sind(i);
```

```
81          sind(Omega)*cosd(omega)+cosd(Omega)*sind(omega)*cosd(i)
            ↪ -sind(Omega)*sind(omega)+cosd(Omega)*cosd(omega)*cosd(i)
            ↪ -cosd(Omega)*sind(i);
82          sind(omega)*sind(i) cosd(omega)*sind(i) cosd(i)]
83          r_vector = transformation*r_pqw
84          v_vector = transformation*v_pqw
85          return r_vector, v_vector
86  end
87
88
89  function cartesian_to_orbital_elements(r_vector, v_vector, mu)
90          #=
91          Computes cartesian position and velocity vector given
    ↪  some orbital elements
92          Input:
93          r_vector [m] , v_vector [m/s]
94          mu [m^3/(kg*s^2)] (GM)
95          Output:
96          a [m] (Semi-major axis)
97          e [] (Eccentricity)
98          i [deg] (Inclination)
99          Omega [deg] (Longitude of the ascending node)
100         omega [deg] (Argument of pericenter)
101         ni [deg] (True anomaly)
102         M [deg] (Mean anomaly)
103         =#
104
105         # Compute magnitude and angular momentum
106         r = sqrt(r_vector[1]^2 + r_vector[2]^2 + r_vector[3]^2)
107         v = sqrt(v_vector[1]^2 + v_vector[2]^2 + v_vector[3]^2)
108         h_vector = cross(r_vector, v_vector)
109         h = sqrt(h_vector[1]^2 + h_vector[2]^2 + h_vector[3]^2)
110         n_vector = cross([0.0,0.0,1.0], h_vector)
111         n = sqrt(n_vector[1]^2 + n_vector[2]^2 + n_vector[3]^2)
112         e_vector = (((v^2 - (mu/r))*r_vector) -
            ↪  (dot(r_vector,v_vector)*v_vector))/mu
113         e = sqrt(e_vector[1]^2 + e_vector[2]^2 + e_vector[3]^2)
114         ksi = ((v^2)/2) - (mu/r)
115         a = - mu/(2*ksi)
116         i = acosd(h_vector[3]/h)
117         current_orbit = dimorphos_orbit_type
118         Omega = acosd(n_vector[1]/n)
119         if n_vector[2] < 0
120         Omega = 360 - Omega
121         end
122         if current_orbit == "EEO"
123         omega = acosd(e_vector[1]/e)
124         if e_vector[2] < 0
```

```
125         omega = 360 - omega
126         end
127         else
128         omega = acosd(dot(n_vector, e_vector)/(n*e))
129         if e_vector[3] < 0
130         omega = 360 - omega
131         end
132         end
133         if current_orbit == "CIO"
134         u = acosd(dot(n_vector, r_vector)/(n*r))
135         if r_vector[3] < 0
136         u = 360 - u
137         end
138         return a, e, i, Omega, omega, u
139         elseif current_orbit == "CEO"
140         lambda_true = acosd(r_vector[1]/r)
141         if r_vector[2] < 0
142         lambda_true = 360 - lambda_true
143         end
144         return a, e, i, Omega, omega, lambda_true
145         else
146         ni = acosd(dot(e_vector, r_vector)/(e*r))
147         if dot(r_vector, v_vector) < 0
148         ni = 360 - ni
149         end
150         M = deg2rad(ni) - 2*e*sind(ni) + ((3*e^2/4)+(e^4/8))*sind(2*ni) -
            ↪ (e^3/3)*sind(3*ni) + (5*e^4/32)*sind(4*ni)
151         #M = atan((sqrt(1-e^2)*sind(ni)/(1+e*cosd(ni))),
            ↪ (e+cosd(ni))/(1+e*cosd(ni))) -
            ↪ (e*(sqrt(1-e^2)*sind(ni)/(1+e*cosd(ni))))
152         M = rad2deg(M)
153         return a, e, i, Omega, omega, M
154         end
155 end
156
157
158 function sight_algorithm(r1_vector, r2_vector)
159         #=
160         Computes if there is sight between two points in space
    ↪ (assuming Didymos to be the obstuction)
161         INPUT:
162         r1_vector, Vector(Float64, 3), position vector of the
    ↪ first point in [m]
163         r2_vector, Vector(Float64, 3), position vector of the
    ↪ second point in [m]
164         OUTPUT:
165         0, if there is no line of sight between the two points
166         1, if there is line of sight between the two points
```

```
167         =#
168         radius_didymos = 325 # [m]
169         r1_magnitude = sqrt(r1_vector[1]^2 + r1_vector[2]^2 +
            ↪  r1_vector[3]^2)
170         r2_magnitude = sqrt(r2_vector[1]^2 + r2_vector[2]^2 +
            ↪  r2_vector[3]^2)
171         tau_min = (r1_magnitude^2 - dot(r1_vector,
            ↪  r2_vector))/(r1_magnitude^2 + r2_magnitude^2 - (2*dot(r1_vector,
            ↪  r2_vector)))
172         line_of_sight = false
173         if tau_min < 0  tau_min > 1
174         line_of_sight = true
175         else
176         parametric_tau_min = ((1-tau_min)*r1_magnitude^2) + (dot(r1_vector,
            ↪  r2_vector)*tau_min)
177         if parametric_tau_min >= radius_didymos
178         line_of_sight = true
179         end
180         end
181         if line_of_sight == false
182         println("Shadow!")
183         end
184         return Int64(line_of_sight)
185 end
186
187
188 function geometrical_shadow_check(x_dimorphos, y_dimorphos, z_dimorphos,
    ↪  x_sun, y_sun, z_sun)
189         radius_didymos = 325 # [m]
190         radius_sun = 696340000 # [m]
191         distance = sqrt(x_sun^2 + y_sun^2 + z_sun^2)
192         a_umbra = asin((radius_sun - radius_didymos)/distance)
193         a_penumbra = asin((radius_sun + radius_didymos)/distance)
194         # Initially assume we are in sunlight, compute magnitudes and dot
            ↪  product
195         shadow = 1
196         r_dimorphos_magnitude = sqrt(x_dimorphos^2 + y_dimorphos^2 +
            ↪  z_dimorphos^2)
197         r_sun_magnitude = sqrt(x_sun^2 + y_sun^2 + z_sun^2)
198         dot_product = x_dimorphos*x_sun + y_dimorphos*y_sun +
            ↪  z_dimorphos*z_sun
199         # Decide if we should evaluate umbra/penumbra possibility
200         if dot_product < 0
201         dot_product = -x_dimorphos * x_sun - y_dimorphos * y_sun -
            ↪  z_dimorphos * z_sun
202         angle_s = acos(dot_product/(r_dimorphos_magnitude*r_sun_magnitude))
203         sat_horizontal = r_dimorphos_magnitude*cos(angle_s)
204         sat_vertical = r_dimorphos_magnitude*sin(angle_s)
```

```
205         x = radius_didymos/sin(a_penumbra)
206         pen_vertical = tan(a_penumbra)*(x+sat_horizontal)
207         if sat_vertical <= pen_vertical
208         y = radius_didymos/sin(a_umbra)
209         umb_vertical = tan(a_umbra)*(y-sat_horizontal)
210         if sat_vertical <= umb_vertical
211         shadow = 0
212         else
213         # Assume that Solar Intensity decreases linearly, calculate shadow
            ↪ value
214         slope = (1-0)/(pen_vertical-umb_vertical)
215         sat_vertical = sat_vertical - umb_vertical
216         shadow = slope*sat_vertical
217         end
218         end
219         end
220         return shadow
221 end
222
223
224 function propagate_and_compute_dimorphos_pixel_points(a_dimorphos,
    ↪ e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos, M_dimorphos,
    ↪ start_time, end_time, step_size, spice_start_time, mu)
225         # Compute initial position and velocity vector from orbital elements
226         r_vector, v_vector = orbital_elements_to_cartesian(a_dimorphos,
            ↪ e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos,
            ↪ M_dimorphos, mu)
227         x = r_vector[1]
228         y = r_vector[2]
229         z = r_vector[3]
230         vx = v_vector[1]
231         vy = v_vector[2]
232         vz = v_vector[3]
233
234         # Propagate Dimorphos
235         x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos,
            ↪ vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz, mu,
            ↪ start_time, end_time, total_photos, enable_perturbation)
236         # Select every xth element to match the photos taken
237         photo_selector = LinRange(1, length(x_dimorphos), total_photos)
238         index_vector = zeros(Int64, total_photos)
239         for i = 1:length(index_vector)
240         index_vector[i] = floor(Int64, photo_selector[i])
241         end
242         x_dimorphos = x_dimorphos[index_vector]
243         y_dimorphos = y_dimorphos[index_vector]
244         z_dimorphos = z_dimorphos[index_vector]
245         vx_dimorphos = vx_dimorphos[index_vector]
```

```
246         vy_dimorphos = vy_dimorphos[index_vector]
247         vz_dimorphos = vz_dimorphos[index_vector]
248         t_vector = t_vector[index_vector]
249         dimorphos_coordinates = hcat(x_dimorphos/1000, y_dimorphos/1000,
            ↪  z_dimorphos/1000)
250
251         # Orbit start and end time (for SPICE)
252         spice_end_time = spice_start_time + end_time
253
254         # Initialize position arrays (x, y, z)
255         didymos_coordinates = zeros(Float64, total_photos, 3)
256         barycenter_coordinates = zeros(Float64, total_photos, 3)
257         barycenter_offset = zeros(Float64, 3)
258         barycenter_initial_coordinates = zeros(Float64, 3)
259         camera_position_didymos = zeros(Float64, total_photos, 3)
260         camera_position_dimorphos = zeros(Float64, total_photos, 3)
261         didymos_pixel_coordinates = zeros(Float64, total_photos, 2)
262         dimorphos_pixel_coordinates = zeros(Float64, total_photos, 2)
263
264         # HERA_AFC-1 coordinate system unit vectors
265         e_x = [1.0, 0.0, 0.0]
266         e_y = [0.0, 1.0, 0.0]
267         e_z = [0.0, 0.0, 1.0]
268
269         # Main SPICE calculations loop
270         focal_length = 10.6*10^-5 # [km]
271         iteration = 1
272         for current_time in LinRange(start_time, end_time, total_photos)
273         i = spice_start_time + current_time
274         position_didymos, lt = spkpos("-658030", i, "J2000", "None", "-999")
275         position_system_barycenter, lt = spkpos("2065803", i, "J2000",
            ↪  "None", "-999")
276         position_dimorphos = dimorphos_coordinates[iteration, :]
277         # Get didymos reference position or translate hera_coordinates
278         for j in 1:3
279         barycenter_offset[j] = 0.0
280         end
281         # Apply offset to all coordinates
282         for j in 1:3
283         barycenter_coordinates[iteration, j] = position_system_barycenter[j]
            ↪  + barycenter_offset[j]
284         didymos_coordinates[iteration, j] = position_didymos[j] +
            ↪  barycenter_offset[j]
285         dimorphos_coordinates[iteration, j] =
            ↪  barycenter_coordinates[iteration, j] + position_dimorphos[j]
286         end
287         # Rotate HERA_AFC-1 camera reference frame assuming it is always
            ↪  tracking the barycenter
```

```
288         rotation_matrix =
            ↪ compute_rotation_matrix(barycenter_coordinates[iteration, :],
            ↪ e_z)
289         barycenter_coordinates[iteration, :] = rotation_matrix *
            ↪ barycenter_coordinates[iteration, :]
290         didymos_coordinates[iteration, :] =  rotation_matrix *
            ↪ didymos_coordinates[iteration, :]
291         dimorphos_coordinates[iteration, :] =  rotation_matrix *
            ↪ dimorphos_coordinates[iteration, :]
292         # Add pointing error
293         pointing_error_matrix =
            ↪ error_rotation_matrix(random_error[iteration],
            ↪ random_axis[iteration])
294         camera_position_didymos[iteration, :] = pointing_error_matrix *
            ↪ didymos_coordinates[iteration, :]
295         camera_position_dimorphos[iteration, :] = pointing_error_matrix *
            ↪ dimorphos_coordinates[iteration, :]
296         # Store pixel coordinates for Didymos and Dimorphos
297         for j in 1:2
298         didymos_pixel_coordinates[iteration, j] =
            ↪ focal_length*camera_position_didymos[iteration,
            ↪ j]/camera_position_didymos[iteration, 3]
299         dimorphos_pixel_coordinates[iteration, j] =
            ↪ focal_length*camera_position_dimorphos[iteration,
            ↪ j]/camera_position_dimorphos[iteration, 3]
300         end
301         iteration += 1
302         end
303
304         local_x_pixel_didymos, local_y_pixel_didymos =
            ↪ convert_to_pixels(didymos_pixel_coordinates[:, 1],
            ↪ didymos_pixel_coordinates[:, 2], x_boundaries, y_boundaries)
305         local_x_pixel_dimorphos, local_y_pixel_dimorphos =
            ↪ convert_to_pixels(dimorphos_pixel_coordinates[:, 1],
            ↪ dimorphos_pixel_coordinates[:, 2],x_boundaries, y_boundaries)
306
307         # Reject images where we cannot see one object
308         for i in 1:length(local_x_pixel_didymos)
309         if ismissing(local_x_pixel_didymos[i])
            ↪ ismissing(local_x_pixel_dimorphos[i])
310         local_x_pixel_didymos[i] = missing
311         local_y_pixel_didymos[i] = missing
312         local_x_pixel_dimorphos[i] = missing
313         local_y_pixel_dimorphos[i] = missing
314         end
315         end
316         return local_x_pixel_dimorphos, local_y_pixel_dimorphos
317 end
```

```
318
319
320 function propagate_and_compute_dimorphos_3D_points(a_dimorphos, e_dimorphos,
    ↪ i_dimorphos, Omega_dimorphos, omega_dimorphos, M_dimorphos, start_time,
    ↪ end_time, step_size, spice_start_time, mu)
321         # Compute initial position and velocity vector from orbital elements
322         r_vector, v_vector = orbital_elements_to_cartesian(a_dimorphos,
            ↪ e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos,
            ↪ M_dimorphos, mu)
323         x = r_vector[1]
324         y = r_vector[2]
325         z = r_vector[3]
326         vx = v_vector[1]
327         vy = v_vector[2]
328         vz = v_vector[3]
329
330         # Propagate Dimorphos
331         x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos,
            ↪ vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz, mu,
            ↪ start_time, end_time, total_photos, enable_perturbation)
332         # Select every xth element to match the photos taken
333         photo_selector = LinRange(1, length(x_dimorphos), total_photos)
334         index_vector = zeros(Int64, total_photos)
335         for i = 1:length(index_vector)
336         index_vector[i] = floor(Int64, photo_selector[i])
337         end
338         x_dimorphos = x_dimorphos[index_vector]
339         y_dimorphos = y_dimorphos[index_vector]
340         z_dimorphos = z_dimorphos[index_vector]
341         vx_dimorphos = vx_dimorphos[index_vector]
342         vy_dimorphos = vy_dimorphos[index_vector]
343         vz_dimorphos = vz_dimorphos[index_vector]
344         t_vector = t_vector[index_vector]
345         dimorphos_coordinates = hcat(x_dimorphos/1000, y_dimorphos/1000,
            ↪ z_dimorphos/1000)
346
347         # Orbit start and end time (for SPICE)
348         spice_end_time = spice_start_time + end_time
349
350         # Initialize position arrays (x, y, z)
351         didymos_coordinates = zeros(Float64, total_photos, 3)
352         barycenter_coordinates = zeros(Float64, total_photos, 3)
353         barycenter_offset = zeros(Float64, 3)
354         barycenter_initial_coordinates = zeros(Float64, 3)
355         camera_position_didymos = zeros(Float64, total_photos, 3)
356         camera_position_dimorphos = zeros(Float64, total_photos, 3)
357         didymos_pixel_coordinates = zeros(Float64, total_photos, 2)
358         dimorphos_pixel_coordinates = zeros(Float64, total_photos, 2)
```

```
359
360         # HERA_AFC-1 coordinate system unit vectors
361         e_x = [1.0, 0.0, 0.0]
362         e_y = [0.0, 1.0, 0.0]
363         e_z = [0.0, 0.0, 1.0]
364
365         # Main SPICE calculations loop
366         focal_length = 10.6*10^-5 # [km]
367         iteration = 1
368         for current_time in LinRange(start_time, end_time, total_photos)
369         i = spice_start_time + current_time
370         position_didymos, lt = spkpos("-658030", i, "J2000", "None", "-999")
371         position_system_barycenter, lt = spkpos("2065803", i, "J2000",
            ↪  "None", "-999")
372         position_dimorphos = dimorphos_coordinates[iteration, :]
373         # Get didymos reference position or translate hera_coordinates
374         for j in 1:3
375         barycenter_offset[j] = 0.0
376         end
377         # Apply offset to all coordinates
378         for j in 1:3
379         barycenter_coordinates[iteration, j] = position_system_barycenter[j]
            ↪  + barycenter_offset[j]
380         didymos_coordinates[iteration, j] = position_didymos[j] +
            ↪  barycenter_offset[j]
381         dimorphos_coordinates[iteration, j] =
            ↪  barycenter_coordinates[iteration, j] + position_dimorphos[j]
382         end
383         # Rotate HERA_AFC-1 camera reference frame assuming it is always
            ↪  tracking the barycenter
384         rotation_matrix =
            ↪  compute_rotation_matrix(barycenter_coordinates[iteration, :],
            ↪  e_z)
385         barycenter_coordinates[iteration, :] = rotation_matrix *
            ↪  barycenter_coordinates[iteration, :]
386         didymos_coordinates[iteration, :] =  rotation_matrix *
            ↪  didymos_coordinates[iteration, :]
387         dimorphos_coordinates[iteration, :] =  rotation_matrix *
            ↪  dimorphos_coordinates[iteration, :]
388         # Add pointing error
389         pointing_error_matrix =
            ↪  error_rotation_matrix(random_error[iteration],
            ↪  random_axis[iteration])
390         camera_position_didymos[iteration, :] = pointing_error_matrix *
            ↪  didymos_coordinates[iteration, :]
391         camera_position_dimorphos[iteration, :] = pointing_error_matrix *
            ↪  dimorphos_coordinates[iteration, :]
392         # Store pixel coordinates for Didymos and Dimorphos
```

```
393         for j in 1:2
394         didymos_pixel_coordinates[iteration, j] =
            ↪  focal_length*camera_position_didymos[iteration,
            ↪  j]/camera_position_didymos[iteration, 3]
395         dimorphos_pixel_coordinates[iteration, j] =
            ↪  focal_length*camera_position_dimorphos[iteration,
            ↪  j]/camera_position_dimorphos[iteration, 3]
396         end
397         iteration += 1
398         end
399         return dimorphos_coordinates
400 end
401
402
403 function type_of_orbit(e, i)
404         # Returns orbit type given eccentricity and inclination
405         if e < 10^-5
406         if i < 10^-1
407         # Circular equatorial orbit
408         answer = "CEO"
409         else
410         # Circular inclined orbit
411         answer = "CIO"
412         end
413         else
414         if i < 10^-1
415         # Elliptical equatorial orbit
416         answer = "EEO"
417         else
418         # Normal orbit
419         answer = "Normal"
420         end
421         end
422         return answer
423 end
424
425 function propagate_and_dimorphos(local_mass_dimorphos)
426         # Compute initial position and velocity vector from orbital elements
427         r_vector, v_vector = orbital_elements_to_cartesian(a_dimorphos,
            ↪  e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos,
            ↪  M_dimorphos, mu_system_original)
428         x = r_vector[1]
429         y = r_vector[2]
430         z = r_vector[3]
431         vx = v_vector[1]
432         vy = v_vector[2]
433         vz = v_vector[3]
434
```

```
435         # Propagate Dimorphos
436         x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos,
            ↪ vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz,
            ↪ mu_system_original, start_time, end_time, total_photos,
            ↪ enable_perturbation)
437         # Select every xth element to match the photos taken and convert to
            ↪ km
438         photo_selector = LinRange(1, length(x_dimorphos), total_photos)
439         index_vector = zeros(Int64, total_photos)
440         for i = 1:length(index_vector)
441         index_vector[i] = floor(Int64, photo_selector[i])
442         end
443         x_dimorphos = x_dimorphos[index_vector]
444         y_dimorphos = y_dimorphos[index_vector]
445         z_dimorphos = z_dimorphos[index_vector]
446         vx_dimorphos = vx_dimorphos[index_vector]
447         vy_dimorphos = vy_dimorphos[index_vector]
448         vz_dimorphos = vz_dimorphos[index_vector]
449         t_vector = t_vector[index_vector]
450         dimorphos_coordinates = hcat(x_dimorphos/1000, y_dimorphos/1000,
            ↪ z_dimorphos/1000)
451
452         x_didymos = (G*local_mass_dimorphos/mu_system_original)*x_dimorphos
453         y_didymos = (G*local_mass_dimorphos/mu_system_original)*y_dimorphos
454         z_didymos = (G*local_mass_dimorphos/mu_system_original)*z_dimorphos
455         didymos_coordinates = hcat(x_didymos/1000, y_didymos/1000,
            ↪ z_didymos/1000)
456
457         # Orbit start and end time (for SPICE)
458         spice_end_time = spice_start_time + end_time
459
460         # Initialize position arrays (x, y, z)
461         #didymos_coordinates = zeros(Float64, total_photos, 3)
462         barycenter_coordinates = zeros(Float64, total_photos, 3)
463         barycenter_offset = zeros(Float64, 3)
464         barycenter_initial_coordinates = zeros(Float64, 3)
465         camera_position_didymos = zeros(Float64, total_photos, 3)
466         camera_position_dimorphos = zeros(Float64, total_photos, 3)
467         didymos_pixel_coordinates = zeros(Float64, total_photos, 2)
468         dimorphos_pixel_coordinates = zeros(Float64, total_photos, 2)
469
470         # HERA_AFC-1 coordinate system unit vectors
471         e_x = [1.0, 0.0, 0.0]
472         e_y = [0.0, 1.0, 0.0]
473         e_z = [0.0, 0.0, 1.0]
474
475         # Main SPICE calculations loop
476         focal_length = 10.6*10^-5 # [km]
```

```
477         iteration = 1
478         for current_time in LinRange(start_time, end_time, total_photos)
479         i = spice_start_time + current_time
480         position_didymos = didymos_coordinates[iteration, :]
481         position_system_barycenter, lt = spkpos("2065803", i, "J2000",
            ↪  "None", "-999")
482         position_dimorphos = dimorphos_coordinates[iteration, :]
483         # Get didymos reference position or translate hera_coordinates
484         for j in 1:3
485         barycenter_offset[j] = rand(barycenter_error_distribution) +
            ↪  rand(hera_error_distribution)
486         end
487         # Apply offset to all coordinates
488         for j in 1:3
489         barycenter_coordinates[iteration, j] = position_system_barycenter[j]
            ↪  + barycenter_offset[j]
490         didymos_coordinates[iteration, j] = barycenter_coordinates[iteration,
            ↪  j] + position_didymos[j]
491         dimorphos_coordinates[iteration, j] =
            ↪  barycenter_coordinates[iteration, j] + position_dimorphos[j]
492         end
493         # Rotate HERA_AFC-1 camera reference frame assuming it is always
            ↪  tracking the barycenter
494         rotation_matrix =
            ↪  compute_rotation_matrix(barycenter_coordinates[iteration, :],
            ↪  e_z)
495         barycenter_coordinates[iteration, :] = rotation_matrix *
            ↪  barycenter_coordinates[iteration, :]
496         didymos_coordinates[iteration, :] =  rotation_matrix *
            ↪  didymos_coordinates[iteration, :]
497         dimorphos_coordinates[iteration, :] =  rotation_matrix *
            ↪  dimorphos_coordinates[iteration, :]
498         # Add pointing error
499         pointing_error_matrix =
            ↪  error_rotation_matrix(random_error[iteration],
            ↪  random_axis[iteration])
500         camera_position_didymos[iteration, :] = pointing_error_matrix *
            ↪  didymos_coordinates[iteration, :]
501         camera_position_dimorphos[iteration, :] = pointing_error_matrix *
            ↪  dimorphos_coordinates[iteration, :]
502         # Store pixel coordinates for Didymos and Dimorphos
503         for j in 1:2
504         didymos_pixel_coordinates[iteration, j] =
            ↪  focal_length*camera_position_didymos[iteration,
            ↪  j]/camera_position_didymos[iteration, 3]
505         dimorphos_pixel_coordinates[iteration, j] =
            ↪  focal_length*camera_position_dimorphos[iteration,
            ↪  j]/camera_position_dimorphos[iteration, 3]
```

```
506         end
507         iteration += 1
508         end
509 
510         # Compute boundary vectors/points for the camera
511         boresight_vector = barycenter_coordinates[iteration-1, :]
512         rotation_matrix = compute_rotation_matrix(boresight_vector, e_z)
513         camera_id = -999110
514         boundary_vectors_matrix = get_boundary_vectors(camera_id)
515         line_generator = LinRange(0, 30, 10)
516 
517         # Rotate camera boundary vectors to assume camera is always tracking
            ↪ barycenter
518         for i in 1:4
519         result = rotation_matrix * boundary_vectors_matrix[i, :]
520         for j in 1:3
521         boundary_vectors_matrix[i,j] = result[j]
522         end
523         end
524 
525         # Compute and plot camera boundary lines
526         sensor_boundary_points = zeros(Float64, 4, 3)
527         global x_boundaries = zeros(Float64, 4)
528         global y_boundaries = zeros(Float64, 4)
529         for i in 1:4
530         x_line, y_line, z_line = boundary_vectors_matrix[i, 1] *
            ↪ line_generator, boundary_vectors_matrix[i, 2] * line_generator,
            ↪ boundary_vectors_matrix[i, 3] * line_generator
531         sensor_boundary_points[i, 1] = x_line[length(x_line)]
532         sensor_boundary_points[i, 2] = y_line[length(y_line)]
533         sensor_boundary_points[i, 3] = z_line[length(z_line)]
534         x_boundaries[i] =
            ↪ focal_length*x_line[length(x_line)]/z_line[length(z_line)]
535         y_boundaries[i] =
            ↪ focal_length*y_line[length(y_line)]/z_line[length(z_line)]
536         camera_label = "Camera boundary line " * string(i)
537         #plot3d!(plt_3d, x_line, y_line, z_line, color="green",
            ↪ label=camera_label)
538         end
539 
540         # Compute coordinates in pixels
541         local_x_pixel_didymos, local_y_pixel_didymos =
            ↪ convert_to_pixels(didymos_pixel_coordinates[:, 1],
            ↪ didymos_pixel_coordinates[:, 2], x_boundaries, y_boundaries)
542 
543         return local_x_pixel_didymos, local_y_pixel_didymos
544 end
```

B.3 optimization.jl

```
function residuals(orbit)
        # Select elements based on orbit type
                if dimorphos_orbit_type == "Normal"
                a = orbit[1]
                e = orbit[2]
                i = orbit[3]
                Omega = orbit[4]
                omega = orbit[5]
                M = orbit[6]
                mu = orbit[7]
        elseif dimorphos_orbit_type == "EEO"
                a = orbit[1]
                e = orbit[2]
                i = i_dimorphos
                Omega = Omega_dimorphos
                omega = orbit[3]
                M = orbit[4]
                mu = orbit[5]
        elseif dimorphos_orbit_type == "CIO"
                a = orbit[1]
                e = orbit[2]
                i = orbit[3]
                Omega = orbit[4]
                omega = omega_dimorphos
                M = orbit[5]
                mu = orbit[6]
        else
                a = orbit[1]
                e = orbit[2]
                i = i_dimorphos
                Omega = Omega_dimorphos
                omega = omega_dimorphos
                M = orbit[3]
                mu = orbit[4]
        end
        # Compute dimorphos pixel points for given orbit and compute xhi2
        observed_x, observed_y = x_pixel_dimorphos, y_pixel_dimorphos
        predicted_x, predicted_y =
        ↪  propagate_and_compute_dimorphos_pixel_points(a, e, i, Omega,
        ↪  omega, M, start_time, end_time, step_size, spice_start_time, mu)
        xhi2 = sum_of_squared_residuals(observed_x, observed_y, predicted_x,
        ↪  predicted_y)
        return Float64(xhi2)
end

```

```
43
44 function wobble(masses)
45         current_dimorphos_mass = masses[1]
46         # Compute dimorphos pixel points for given orbit and compute xhi2
47         observed_x, observed_y = x_pixel_didymos, y_pixel_didymos
48         predicted_x, predicted_y =
           ↪  propagate_and_dimorphos(current_dimorphos_mass)
49         xhi2 = sum_of_squared_residuals(observed_x, observed_y, predicted_x,
           ↪  predicted_y)
50         return Float64(xhi2)
51 end
52
53
54 function sum_of_squared_residuals(observed_x, observed_y, predicted_x,
   ↪  predicted_y)
55         x_residuals = observed_x - predicted_x
56         y_residuals = observed_y - predicted_y
57         # Clean up missing values
58         x_residuals = collect(skipmissing(x_residuals))
59         y_residuals = collect(skipmissing(y_residuals))
60         if isempty(x_residuals)  isempty(y_residuals)
61                 return 10^8
62         else
63                 squared_sum = sum(x_residuals.^2) + sum(y_residuals.^2)
64         end
65         return squared_sum
66 end
67
```

B.4 validate_propagator.jl

```
1 using SPICE, ProgressBars, Plots, LinearAlgebra, Distributions, Random,
   ↪  DifferentialEquations, Metaheuristics, DataFrames, CSV, LaTeXStrings
2 include("orbital_utilities.jl")
3 include("propagators.jl")
4 include("spice_utilities.jl")
5 include("camera_utilities.jl")
6 include("optimization.jl")
7 include("results.jl")
8 include("plotting.jl")
9
10
11 function main()
12         # Dimorphos orbit
13         global a_dimorphos = 1183.593 # Semi-major axis
```

```
14          global e_dimorphos = 0.00000098 # Eccentricity
15          global i_dimorphos = 0.035 # Inclination
16          global Omega_dimorphos = 0.0 # Longitude of the ascending node
17          global omega_dimorphos = 0.0 # Argument of periapsis
18          global M_dimorphos = 172.18 # True anomaly
19
20          # Determine what kind of orbit we are dealing with
21          global dimorphos_orbit_type = type_of_orbit(e_dimorphos,
            ↪  i_dimorphos)
22          println("Orbit type is "*dimorphos_orbit_type)
23
24          # Compute initial position and velocity vector from orbital elements
25          r_vector, v_vector = orbital_elements_to_cartesian(a_dimorphos,
            ↪  e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos,
            ↪  M_dimorphos, mu_system)
26          x = r_vector[1]
27          y = r_vector[2]
28          z = r_vector[3]
29          vx = v_vector[1]
30          vy = v_vector[2]
31          vz = v_vector[3]
32
33          println("Initial State Vector")
34          println(x)
35          println(y)
36          println(z)
37          println(vx)
38          println(vy)
39          println(vz)
40
41          # Propagate Dimorphos
42          x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos,
            ↪  vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz,
            ↪  mu_system, start_time, end_time, total_photos,
            ↪  enable_perturbation)
43
44          vallado_x = CSV.read("x_vector.csv", DataFrame, header=0)
45          vallado_y = CSV.read("y_vector.csv", DataFrame, header=0)
46          vallado_z = CSV.read("z_vector.csv", DataFrame, header=0)
47          vallado_vx = CSV.read("vx_vector.csv", DataFrame, header=0)
48          vallado_vy = CSV.read("vy_vector.csv", DataFrame, header=0)
49          vallado_vz = CSV.read("vz_vector.csv", DataFrame, header=0)
50
51          pgfplotsx()
52          x_plot = plot(t_vector, x_dimorphos, label = L"\textnormal{RK4
            ↪  Two-Body with}\ J_2", xlabel = "Time, " * L"t,\ [s]", ylabel =
            ↪  L"x,\ [m]", widen=true, formatter=:plain, legend = :topright)
```

```
53          plot!(x_plot, t_vector, vallado_x.Column1, label = "PKEPLER
            ↪ routine")
54          y_plot = plot(t_vector, y_dimorphos, label = L"\textnormal{RK4
            ↪ Two-Body with}\ J_2", xlabel = "Time, " * L"t,\ [s]", ylabel =
            ↪ L"y,\ [m]", widen=true, formatter=:plain, legend = :topright)
55          plot!(y_plot, t_vector, vallado_y.Column1, label = "PKEPLER
            ↪ routine")
56          z_plot = plot(t_vector, z_dimorphos, label = L"\textnormal{RK4
            ↪ Two-Body with}\ J_2", xlabel = "Time, " * L"t,\ [s]", ylabel =
            ↪ L"z,\ [m]", widen=true, formatter=:plain, legend = :topright)
57          plot!(z_plot, t_vector, vallado_z.Column1, label = "PKEPLER
            ↪ routine")
58          vx_plot = plot(t_vector, vx_dimorphos, label = L"\textnormal{RK4
            ↪ Two-Body with}\ J_2", xlabel = "Time, " * L"t,\ [s]", ylabel =
            ↪ L"v_{x},\ [m/s]", widen=true, formatter=:plain, legend =
            ↪ :topright)
59          plot!(vx_plot, t_vector, vallado_vx.Column1, label = "PKEPLER
            ↪ routine")
60          vy_plot = plot(t_vector, vy_dimorphos, label = L"\textnormal{RK4
            ↪ Two-Body with}\ J_2", xlabel = "Time, " * L"t,\ [s]", ylabel =
            ↪ L"v_{y},\ [m/s]", widen=true, formatter=:plain, legend =
            ↪ :topright)
61          plot!(vy_plot, t_vector, vallado_vy.Column1, label = "PKEPLER
            ↪ routine")
62          vz_plot = plot(t_vector, vz_dimorphos, label = L"\textnormal{RK4
            ↪ Two-Body with}\ J_2", xlabel = "Time, " * L"t,\ [s]", ylabel =
            ↪ L"v_{z},\ [m/s]", widen=true, formatter=:plain, legend =
            ↪ :topright)
63          plot!(vz_plot, t_vector, vallado_vz.Column1, label = "PKEPLER
            ↪ routine")
64          savefig(x_plot, ".\\Results\\validate_x_data.pdf")
65          savefig(y_plot, ".\\Results\\validate_y_data.pdf")
66          savefig(z_plot, ".\\Results\\validate_z_data.pdf")
67          savefig(vx_plot, ".\\Results\\validate_vx_data.pdf")
68          savefig(vy_plot, ".\\Results\\validate_vy_data.pdf")
69          savefig(vz_plot, ".\\Results\\validate_vz_data.pdf")
70  end
71
72  # System properties
73  G = 6.67430*10^-11
74  mass_didymos = 5.32*10^11
75  global radius_didymos = 390 # [m]
76  global J2_didymos = 0.011432722
77  global mass_dimorphos = 4.94*10^9
78  global mu_system = G*(mass_didymos+mass_dimorphos)
79  global c = 3.0*10^8
80  global enable_perturbation = true
81
```

```
# Time properties
hour = 3600.0
number_of_orbits = 25
# Photos to be taken
photos_per_orbit = 40
global total_photos = photos_per_orbit * number_of_orbits
global start_time = 0.0
global end_time = 300*hour
global step_size = floor((end_time-start_time)/total_photos)

main()
```

B.5 results.jl

```
function compute_percentage_error(actual_elements, predicted_elements)
        #=
        Computes percentage error given two sets of orbital elements
        Input:
        actual_elements, Vector(Float64, 6) containing actual orbital elements
        predicted_elements, Vector(Float64, 6) containing predicted orbital elements
        Output:
        percentage_errors, Vector(Float64, 6) containing percentage (%) errors on initial element values
        =#

        percentage_errors = zeros(Float64, 6)
        for i in 1:6
                if i>=3 && i<=6
                        # Ensure conversion to radians for angles
                        actual_elements[i] = deg2rad(actual_elements[i])
                        predicted_elements[i] = deg2rad(predicted_elements[i])
                end
                percentage_errors[i] = 100 * (abs(predicted_elements[i] - actual_elements[i])/actual_elements[i])
                println("Percentage error for element " * string(i) * " is " * string(percentage_errors[i]))
        end
        return Nothing
end


```

```
function compute_mean_absolute_percentage_error(actual_elements, predicted_elements)
        #=
        Computes mean absolute percentage errors for two orbits, provided with the initial orbital elements
        Input:
        actual_elements, Vector(Float64, 6) containing actual orbital elements
        predicted_elements, Vector(Float64, 6) containing predicted orbital elements
        Output:
        percentage_errors, Vector(Float64, 6) mean absolute percentage error (%) for each element
        =#

        # Dimorphos orbit
        a_dimorphos = actual_elements[1] # Semi-major axis
        e_dimorphos = actual_elements[2] # Eccentricity
        i_dimorphos = actual_elements[3] # Inclination
        Omega_dimorphos = actual_elements[4] # Argument of periapsis
        omega_dimorphos = actual_elements[5] # Longitude of the ascending node
        M_dimorphos = actual_elements[6] # Mean anomaly

        # Compute initial position and velocity vector from orbital elements
        r_vector, v_vector = orbital_elements_to_cartesian(a_dimorphos, e_dimorphos, i_dimorphos, Omega_dimorphos, omega_dimorphos, M_dimorphos, mu_system)
        x = r_vector[1]
        y = r_vector[2]
        z = r_vector[3]
        vx = v_vector[1]
        vy = v_vector[2]
        vz = v_vector[3]

        # Propagate Dimorphos
        x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos, vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz, mu_system, start_time, end_time, total_photos, enable_perturbation)
        # Select every xth element to match the photos taken
        photo_selector = LinRange(1, length(x_dimorphos), total_photos)
        index_vector = zeros(Int64, total_photos)
        for i = 1:length(index_vector)
                index_vector[i] = floor(Int64, photo_selector[i])
        end
        x_dimorphos = x_dimorphos[index_vector]
        y_dimorphos = y_dimorphos[index_vector]
```

```
62          z_dimorphos = z_dimorphos[index_vector]
63          vx_dimorphos = vx_dimorphos[index_vector]
64          vy_dimorphos = vy_dimorphos[index_vector]
65          vz_dimorphos = vz_dimorphos[index_vector]
66          t_vector = t_vector[index_vector]
67
68          # Compute orbital elements vs time for the fitted orbit
69          vector_size = size(x_dimorphos)[1]
70          a_vector = zeros(Float64, vector_size)
71          e_vector = zeros(Float64, vector_size)
72          i_vector = zeros(Float64, vector_size)
73          Omega_vector = zeros(Float64, vector_size)
74          omega_vector = zeros(Float64, vector_size)
75          M_vector = zeros(Float64, vector_size)
76          for i in 1:vector_size
77                  a_vector[i], e_vector[i], i_vector[i], Omega_vector[i],
                    ↪  omega_vector[i], M_vector[i] =
                    ↪  cartesian_to_orbital_elements([x_dimorphos[i],
                    ↪  y_dimorphos[i], z_dimorphos[i]], [vx_dimorphos[i],
                    ↪  vy_dimorphos[i], vz_dimorphos[i]], mu_system)
78                  i_vector[i] = deg2rad(i_vector[i])
79                  Omega_vector[i] = deg2rad(Omega_vector[i])
80                  omega_vector[i] = deg2rad(omega_vector[i])
81                  M_vector[i] = deg2rad(M_vector[i])
82          end
83
84          observed_values = [a_vector e_vector i_vector Omega_vector
            ↪  omega_vector M_vector]
85
86          # Extract final guess from optimization
87          a_final = predicted_elements[1]
88          e_final = predicted_elements[2]
89          i_final = predicted_elements[3]
90          Omega_final = predicted_elements[4]
91          omega_final = predicted_elements[5]
92          M_final = predicted_elements[6]
93
94          # Compute initial position and velocity vector from orbital elements
95          r_vector, v_vector = orbital_elements_to_cartesian(a_final, e_final,
            ↪  i_final, Omega_final, omega_final,M_final, mu_system)
96          x = r_vector[1]
97          y = r_vector[2]
98          z = r_vector[3]
99          vx = v_vector[1]
100         vy = v_vector[2]
101         vz = v_vector[3]
102
103         # Propagate Dimorphos
```

```
104         x_dimorphos, y_dimorphos, z_dimorphos, vx_dimorphos, vy_dimorphos,
            ↪ vz_dimorphos, t_vector = runge_kutta_4(x, y, z, vx, vy, vz,
            ↪ mu_system, start_time, end_time, total_photos,
            ↪ enable_perturbation)
105         # Select every xth element to match the photos taken
106         photo_selector = LinRange(1, length(x_dimorphos), total_photos)
107         index_vector = zeros(Int64, total_photos)
108         for i = 1:length(index_vector)
109                 index_vector[i] = floor(Int64, photo_selector[i])
110         end
111         x_dimorphos = x_dimorphos[index_vector]
112         y_dimorphos = y_dimorphos[index_vector]
113         z_dimorphos = z_dimorphos[index_vector]
114         vx_dimorphos = vx_dimorphos[index_vector]
115         vy_dimorphos = vy_dimorphos[index_vector]
116         vz_dimorphos = vz_dimorphos[index_vector]
117         t_vector = t_vector[index_vector]
118
119         # Compute orbital elements vs time for the fitted orbit
120         vector_size = size(x_dimorphos)[1]
121         a_vector = zeros(Float64, vector_size)
122         e_vector = zeros(Float64, vector_size)
123         i_vector = zeros(Float64, vector_size)
124         Omega_vector = zeros(Float64, vector_size)
125         omega_vector = zeros(Float64, vector_size)
126         M_vector = zeros(Float64, vector_size)
127         for i in 1:vector_size
128                 a_vector[i], e_vector[i], i_vector[i], Omega_vector[i],
                    ↪ omega_vector[i], M_vector[i] =
                    ↪ cartesian_to_orbital_elements([x_dimorphos[i],
                    ↪ y_dimorphos[i], z_dimorphos[i]],
                    ↪ [vx_dimorphos[i],vy_dimorphos[i], vz_dimorphos[i]],
                    ↪ mu_system)
129                 i_vector[i] = deg2rad(i_vector[i])
130                 Omega_vector[i] = deg2rad(Omega_vector[i])
131                 omega_vector[i] = deg2rad(omega_vector[i])
132                 M_vector[i] = deg2rad(M_vector[i])
133         end
134
135         predicted_values = [a_vector e_vector i_vector Omega_vector
            ↪ omega_vector M_vector]
136
137         percentage_errors = zeros(Float64, 6)
138         for i in 1:6
139                 sum = 0.0
140                 for j in 1:vector_size
141                         sum += abs((observed_values[j, i] -
                            ↪ predicted_values[j, i])/observed_values[j, i])
```

```
142                 end
143                 percentage_errors[i] = 100 * sum / (vector_size)
144                 println("Mean absolute percentage error for element " *
                    ↪  string(i) * " is " * string(percentage_errors[i]) * "
                    ↪  %")
145         end
146         return percentage_errors
147 end
```

Bibliography

[1] David Akin. *Akin's Laws of Spacecraft Design*. URL: https://spacecraft.ssl.umd.edu/akins_laws.html. (accessed: 05/03/2022).

[2] Peter Schulte et al. "The Chicxulub Asteroid Impact and Mass Extinction at the Cretaceous-Paleogene Boundary". In: *Science* 327.5970 (2010), pp. 1214–1218. DOI: 10.1126/science.1177265. URL: https://www.science.org/doi/abs/10.1126/science.1177265.

[3] Near-Earth Objects Coordination Center European Space Agency. *Definition and Assumptions*. URL: https://neo.ssa.esa.int/definitions-assumptions. (accessed: 08/03/2022).

[4] Minor Planet Center International Astronomical Union. *Data available from Minor Planet Center*. URL: https://minorplanetcenter.net/data. (accessed: 08/03/2022).

[5] Center for Near Earth Object Studies NASA Jet Propulsion Laboratory. *Disovery statistics for Near-Earth Asteroids*. URL: https://cneos.jp/.nasa.gov/stats/totals.html. (accessed: 08/03/2022).

[6] Jet Propulsion Laboratory. *Eyes on Asteroids*. URL: https://www.jpl.nasa.gov/asteroid-watch/eyes-on-asteroids. (accessed: 08/03/2022).

[7] T. Gehrels, M.S. Matthews, and A.M. Schumann. *Hazards Due to Comets and Asteroids*. Space science series. University of Arizona Press, 1994. ISBN: 9780816515059. URL: https://books.google.gr/books?id=xXWZolI9NkUC.

[8] AG Aleksandrova et al. "The preventive destruction of a hazardous asteroid". In: *Astronomy Reports* 60.6 (2016), pp. 611–619.

[9] Ben J. Zimmerman and Bong Wie. "Computational Validation of Nuclear Explosion Energy Coupling Models for Asteroid Fragmentation". In: *AIAA/AAS Astrodynamics Specialist Conference*. DOI: 10.2514/6.2014-4146. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2014-4146. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2014-4146.

[10] Thomas J Ahrens and Alan W Harris. "Deflection and fragmentation of near-Earth asteroids". In: *Nature* 360.6403 (1992), pp. 429–433.

[11] National Aeronautics & Space Administration. *Near-Earth Object Survey and Deflection Analysis of Alternatives: Report to Congress*. 2007. URL: https://web.archive.org/web/20160303220543/http://www.nasa.gov/pdf/171331main_NEO_report_march07.pdf.

[12] Alaa Hussein, Oshri Rozenheck, and Carlos Manuel Entrena Utrilla. "From detection to deflection: Mitigation techniques for hidden global threats of natural space objects with short warning time". In: *Acta Astronautica* 126 (2016). Space Flight Safety, pp. 488–496. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2016.06.013. URL: https://www.sciencedirect.com/science/article/pii/S009457651630306X.

[13] Bong Wie. "Dynamics and control of gravity tractor spacecraft for asteroid deflection". In: *Journal of guidance, control, and dynamics* 31.5 (2008), pp. 1413–1423.

[14] Megan Bruck Syal, J Michael Owen, and Paul L Miller. "Deflection by kinetic impact: Sensitivity to asteroid properties". In: *Icarus* 269 (2016), pp. 50–61.

[15] A.F. Cheng et al. "Asteroid Impact and Deflection Assessment mission: Kinetic impactor". In: *Planetary and Space Science* 121 (2016), pp. 27–35. ISSN: 0032-0633. DOI: https://doi.org/10.1016/j.pss.2015.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0032063315300337.

[16] Ian Carnelli, Andrés Galvéz, and Franco Ongaro. "Learning to deflect near earth objects: industrial design of the Don Quijote mission". In: *57th International Astronautical Congress*. 2006, A3–5.

[17] Andrew S Rivkin et al. "The double asteroid redirection test (DART): planetary defense investigations and requirements". In: *The Planetary Science Journal* 2.5 (2021), p. 173.

[18] Patrick Michel et al. "European component of the AIDA mission to a binary asteroid: Characterization and interpretation of the impact of the DART mission". In: *Advances in Space Research* 62.8 (2018). Past, Present and Future of Small Body Science and Exploration, pp. 2261–2272. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2017.12.020. URL: https://www.sciencedirect.com/science/article/pii/S0273117717308967.

[19] Jesus Gil-Fernández et al. "HERA autonomous Guidance, Navigation and Control experiments: enabling better asteroid science & future missions". In: (2019).

[20] Jeff Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: https://epubs.siam.org/doi/10.1137/141000671.

[21] Julia programming language. *Julia Micro-Benchmarks*. URL: https://julialang.org/benchmarks/. (accessed: 13/03/2022).

[22] Ulrich Herfort and Carlos M. Casas. "Trajectory preparation for the approach of spacecraft rosetta to comet 67P/Churyumov-Gerasimenko". In: Proceedings of the 25th International Symposium on Space Flight Dynamics ISSFD. 2015.

[23] C.H. Acton. "Ancillary Data Services of NASA's Navigation and Ancillary Information Facility". In: *Planetary and Space Science* 44.1 (1996), pp. 65–70.

[24] ESA SPICE service. *Operational Kernels for the Hera mission*. URL: https://www.cosmos.esa.int/web/spice. (accessed: 13/03/2022).

[25] A Pellacani et al. "HERA vision based GNC and autonomy". In: *Proceedings of the 8th European Conference for Aeronautics and Space Sciences (EUCASS), Madrid, Spain*. 2019, pp. 1–4.

[26] Andrew S Rivkin et al. "The double asteroid redirection test (DART): planetary defense investigations and requirements". In: *The Planetary Science Journal* 2.5 (2021), p. 173.

[27] D.A. Vallado, W.D. McClain, and J. Wertz. *Fundamentals of Astrodynamics and Applications*. Space technology library. Microcosm Press, 2013. ISBN: 9781881883197.

[28] R.P. Canale and D. Steven C. Chapra. *Numerical Methods for Engineers*. McGraw-Hill Education, 2014. ISBN: 9780073397924.

[29] D.J. Scheeres. *Orbital Motion in Strongly Perturbed Environments: Applications to Asteroid, Comet and Planetary Satellite Orbiters*. Springer Praxis Books. Springer Berlin Heidelberg, 2014. ISBN: 9783642431630.

[30] Harrison F Agrusa et al. "A benchmarking and sensitivity study of the full two-body gravitational dynamics of the DART mission target, binary asteroid 65803 Didymos". In: *Icarus* 349 (2020), p. 113849.

[31] SP Naidu et al. "Radar observations and a physical model of binary near-Earth asteroid 65803 Didymos, target of the DART mission". In: *Icarus* 348 (2020), p. 113777.

[32] P Scheirich and P Pravec. "Modeling of lightcurves of binary asteroids". In: *Icarus* 200.2 (2009), pp. 531–547.

[33] Shantanu P Naidu and Jean-Luc Margot. "Near-Earth asteroid satellite spins under spin–orbit coupling". In: *The Astronomical Journal* 149.2 (2015), p. 80.

[34] Tatsuaki Okada et al. "Thermal Infrared and Multiband Imaging to Investigate S-type Binary Asteroid Didymos and Dimorphos in Hera Mission". In: *7th IAA Planetary Defense Conference*. 2021, p. 188.

[35] Jena Optronic. *ASTROhead camera datasheet*. URL: https://www.jena-optronik.de/products/star-sensors/astrohead.html. (accessed: 13/06/2022).

[36] ESA. *Cometwatch 6 November - Target Locked!* URL: https://blogs.esa.int/rosetta/2014/11/10/cometwatch-6-november-target-locked/. (accessed: 14/06/2022).

[37] Leonora Bianchi et al. "A survey on metaheuristics for stochastic combinatorial optimization". In: *Natural Computing* 8.2 (2009), pp. 239–287.

[38] M.J. Apter. *The Computer Simulation of Behaviour*. Routledge Library Editions: Artificial Intelligence. Taylor & Francis, 2018. ISBN: 9781351021005.

[39] Marco Dorigo, Gianni Di Caro, and Luca M Gambardella. "Ant algorithms for discrete optimization". In: *Artificial life* 5.2 (1999), pp. 137–172.

[40] David B Fogel. *Artificial intelligence through simulated evolution*. Wiley-IEEE Press, 1998.

[41] Russell Eberhart and James Kennedy. "A new optimizer using particle swarm theory". In: *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*. Ieee. 1995, pp. 39–43.

[42] Jesús-Adolfo Mejía-de Dios and Efrén Mezura-Montes. "A New Evolutionary Optimization Method Based on Center of Mass". In: *Decision Science in Action: Theory and Applications of Modern Decision Analytic Optimisation*. Ed. by Kusum Deep, Madhu Jain, and Said Salhi. Singapore: Springer Singapore, 2019, pp. 65–74. ISBN: 978-981-13-0860-4. DOI: 10.1007/978-981-13-0860-4_6. URL: https://doi.org/10.1007/978-981-13-0860-4_6.

[43] Jesús Mejía, Mehmet Hakan Satman, and Pietro Monticone. *Metaheuristics.jl v3.2.7*. 2022. URL: https://doi.org/10.5281/zenodo.6623647.

[44] S. Liang, X. Li, and J. Wang. *Advanced Remote Sensing: Terrestrial Information Extraction and Applications*. Elsevier Science, 2012. ISBN: 9780123859556.

[45] Ralph B d'Agostino. "An omnibus test of normality for moderate and large size samples". In: *Biometrika* 58.2 (1971), pp. 341–348.