COSMOS FRAMEWORK: SIMULATION AND OPTIMIZATION OF 3D SEARCH

Cosmin-Gabriel Samoilă[1], Damian Dinoiu[2], Emil-Ioan Sluşanschi[3]

In modern numerical simulations, in order to be reliable, a computer code needs to be accurate and fast. The Framework for Combining Optimization and Simulation Software (COSMOS) is a tool which allows for the seamless integration of numerical simulations with appropriate optimizations algorithm. In this paper we present the integration of different numerical solutions in order to simulate a search in a continuous 3D space. The simulations are performed on different computational scenarios revealing significant improvements and useful insights into alternative optimisation solutions.

Keywords: simulation, optimization, high performance computing, COSMOS

1. Introduction

The simulations of real-world environments have become more complex, introducing an ever-increasing amount of data processing requiring computationally intensive workloads coupled with appropriate optimization packages.
COSMOS [9] is a framework allowing the integration of different software and optimization tools. Moreover, it can serve as a wrapper for sending tasks on a HPC[1] system. Its architecture is divided into three independent modules, namely:

- **The Controller module** interacts with external modules by parsing the configuration files, commands the broker and the optimizer modules and processes the output of the tasks;
- **The Broker module** manages the resources by scheduling tasks using different heuristics based on task length and machine load;
- **The Optimization module** is providing data to the Controller module in order to determine the next parameters of the numerical simulation.

This paper presents an extension of the COSMOS framework through the addition of numerical simulations. Consequently, it is necessary to introduce new modules that interact with the Controller module and that focus on the post-optimization process in order to determine the optimal steps of the numerical simulation. The structure is based on three main stages, namely: reconstructing of a 3D continuous space with a mobile camera, gathering information from the scene and processing

[1]Doctorand, SDIALA Doctoral School, University "Politehnica" of Bucharest, Romania, e-mail: cosmin.samoila@upb.ro

[2]Engineer, Faculty of Computer Science and Automatic Control, University "Politehnica" of Bucharest, Romania, e-mail: damiandinoiu@gmail.com

[3]Professor PhD Eng, Faculty of Computer Science and Automatic Control, University "Politehnica" of Bucharest, Romania, , e-mail: emil.slusanschi@upb.ro

[1]High Performance Computing

it; and the optimization process that will determine the next step of the numerical simulation, based on the previously collected data.

COSMOS main modules can work individually, opening the possibility of bringing separate implementations and integrating various other external simulation modules. The main goal is to extend the coverage area of simulations, by introducing a new problem, an informative search in a continuous 3D space.

Being given the initial view of the scene as input, it is wanted to start the search from a random position in order to determine the coordinates of the camera from the input perspective, with data collected across the search. In order to achieve the goal, the simulation should be optimized by generating the best next steps of the search. All these will be synchronized by the main COSMOS modules.

Furthermore, it will be possible for both simulation and optimization processes to be used in solving real problems for unmanned aerial vehicles. The scope can include tracking endangered species [6], monitoring vegetation in precision agriculture [7]. This domain is vast and rapidly growing, due to the continuous, significant improvements in technology.

As previously stated, projects can be easily divided into three main components that will require integration:

- Reconstruction of the scene, where a physically based render Mitsuba2.0 is used. This render permits the modification of the scene layout, camera specification, position, and orientation, in order to simulate the continuous 3D space;
- The extraction of the information from the scene, where the main module introduced is You Only Look Once: Unified, Real-Time Object Detection, which, according to R. Girshick et al. [8], will detect and provide data for the object mapping, in order to estimate the coordinates of the object;
- The optimization, in which case, based on the results given from the second component, the next best step in the simulation is going to be computed by analyzing the objects and their position in the scene.

After implementing the proposed solution, a complete package is expected to be introduced in COSMOS Framework, enhancing the capabilities of this framework. It will be able to render various 3D scenes only by modifying the input file, to recognize different object in the scene and to estimate their position. Furthermore, being given an input file (initial image) and a starting position, by combining the features described above, the initial position of the camera, from where the given picture was taken, will be found in an optimized manner.

Section 2 presents a summary of the related work in the domain of computer simulations and optimizations, introducing COSMOS framework. Section 3 provides theoretical details of the solution that will be implemented, also describing the other tools and frameworks which were used. A comprehensive presentation of the results is available in Section 4. A summary of the paper is depicted in Section 5, together with ideas on further work and research.

2. **Related work**

ASE (Atomic Simulation Environment) [5] is an international open-source software instrument written in Python, serving the main purpose of setting up, controlling, visualising and analysing atomistic software simulations. This environment

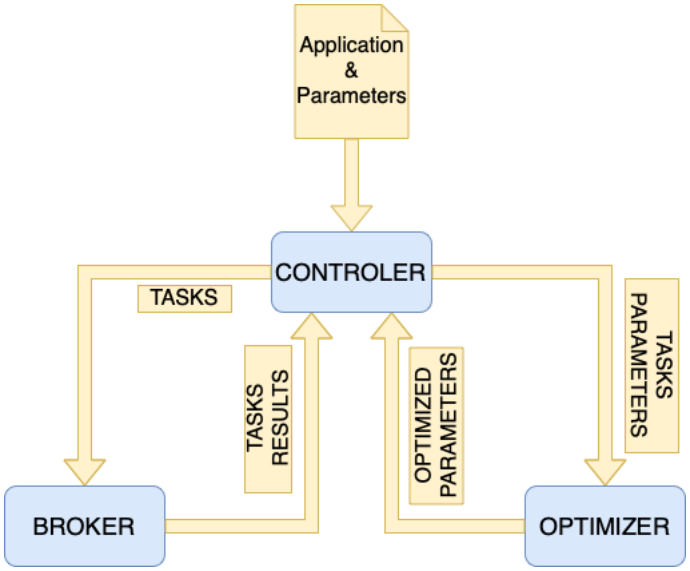

Fig. 1. Architecture of COSMOS Framework

is suitable for simulations, as it uses powerful Python libraries, such as NumPy, allowing the execution of complex tasks. According to its authors [5], ASE is considered to serve as "front-end for atomistic simulations where atomic structures and parameters controlling simulations can be easily defined".

An interactive software systems providing a solution for parameter estimation and identification for mathematical models, belonging to categories such as differential algebraic equations, Laplace transformations, ordinary differential equations, steady-state systems or systems of one-dimensional time-dependent partial differential equations is offered in EASY-FIT [10]. This software offers four optimization routines and a number of differential equation solvers.

A similar approach, offering an interactive software environment combining numerical simulation code with optimization software packages is proposed by Rasch and Bücker in EFCOSS [1]. EFCOSS is focused on optimal experimental design and is presented as an extension of a previous implementation [2] which was mainly focused on parameter fitting.

COSMOS - Framework for Combining Optimization and Simulation Software [9] provides an interface for merging various simulation and optimization software packages. According to [9], simulation software packages consist of mathematical models describing real-world occurrences, such as chemical reactions, electronic circuits, biological processes etc., translated into computationally intensive computer programs. COSMOS – which we are going to use in this paper – supplies a stable interface between optimization programs and simulation packages, schedules tasks in HPC environment, provides optimization hints based on the application output and offers an abstraction for the underlying hardware computing architecture. A simplified architecture of COSMOS can be seen in Figure 1.

3. Proposed solution

The following Section offers an in-depth presentation of the solution which was implemented. As previously mentioned, it consists of three main stages - the scene reconstruction, the image processing and object mapping and the optimization, which will be detailed one by one.

3.1. Scene reconstruction and camera movement

This section will analyze the reconstruction of the scene, by describing the components of the scene and the discretized movement of the camera used in the target search.

Scene components. In an attempt to reconstruct a real-world scene, the following components were introduced:

- camera with known features: model, field of view, resolution, position and orientation;
- emitter: a point-like source of light specifying the position and the intensity;
- the plane surface that represents the ground;
- different objects loaded as polygon file format with own texture - shapes define surfaces that mark transitions between different types of materials.

A preview can be seen in Figure 2a.

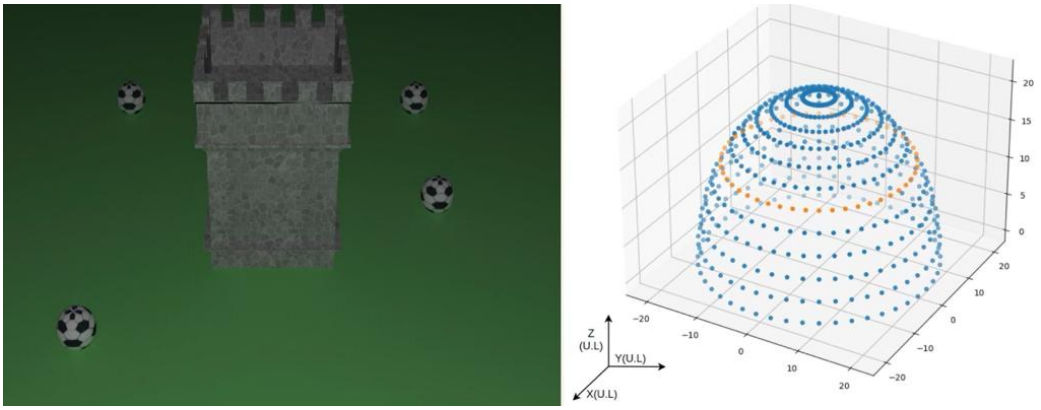

Fig. 2. a) Scene preview b) Camera discretized positions

Camera movement. The main goal of the project is to find the camera position based on an input image, starting from a random point in space. In order to reduce the number of possible states of the camera, its movement will be discretized to a set of points located on an hemisphere of known radius. The camera is able to move alongside the hemisphere, by changing the polar coordinates with a certain angle each step, as expressed in the following equation:

$$\begin{aligned} radius &= \sqrt{pos_x^2 + pos_y^2 + pos_z^2} \\ \theta &= \arccos \frac{pos_z}{radius} \pm angle\_step \\ \phi &= \arctan \frac{pos_y}{pos_z} \pm angle\_step \end{aligned} \tag{1}$$

The movement is characterized by four possible actions: movement alongside latitude (North and South) by changing θ and movement alongside longitude (West and East) by changing ϕ. A preview of the discretized set of points is illustrated in Figure 2b. To obtain a more accurate search, the angle for the longitude will decrease directly proportional with the height. All these considerations lead to a possible movement of the camera on the horizontal plane as well as on the upped hemisphere of the plane, as depicted in Figure 2b. In this image, as well as in Figure 5 and Figure 6 one can see possible positions of the camera while searching for the target position.

The scene that is considered in the paper is static and it contains static elements. The only movable part is the camera and the setup of the problem is such that we are looking for the original position of the camera starting from an arbitrary position of the scene. The only constraint that is currently implemented is that the camera is oriented towards the center of the scene.

If the scene or the objects in the scene (e.g. the balls) are dynamic, the probability of the framework being able to find the original vantage point of the camera is decreasing significantly. To date, we did not conduct such a study.

3.2. Image coordinates and object mapping

In this subsection, the methods used in extracting the information from the images created are presented. First, the method used in recognition of the objects is described. Further, the output is explained, together with its interpretation. The main purpose is to estimate the object's positions in the scene, since this data will serve in the target search and optimization methods.

3.2.1. *Image coordinates.* After each movement step of the camera, based on the new image, an object recognition is performed in order to extract the image coordinates of the objects. The bounding boxes, which surrounds the objects, are extracted as pairs of pixels from the image.

3.2.2. *Object mapping in the scene.* Now, having the pixels coordinates of the objects, it is wanted to estimate the real position in the scene. Firstly, the position of the object will be estimated relatively to the current camera position. Secondly, by rotating the coordinates in the scene surface, the real coordinates will be computed.

Note: Axes correspond to the scene space, not the image space. For better correlation, they have also been kept when referring to the images' space.

Position on X axis. First of all, the depth estimation is calculated. As per [4], the depth from a perspective camera can be calculated using the following formula:

$$d = \frac{f * y_c}{f * \sin\theta_x - (v_c - v_b) * \cos\theta_x} * \frac{1}{s} \tag{2}$$

In equation 2, f represents the focal length of the camera, θ_x expresses the camera tilt in radians, y_c is the camera height, v_c the camera optical center, v_b the object center in pixel coordinates on X axis and s, the scale. Knowing that the camera is always oriented to the center of the scene, v_c will also represent the center coordinates in the image. As a result, when v_c equals v_b, the distance expressed in equation 2 will represent the distance on the X axis of the surface plane form the camera to the center of the scene.

$$d_0 = \frac{y_c}{\sin\theta_x} * \frac{1}{s} \tag{3}$$

Combining equations 2 and 3, the distance to the center can be expressed, on X axis, as a difference:

$$\Delta_x = \frac{(v_c - v_b) * y_c * \cos\theta_x}{f * \sin{\theta_x}^2 - (v_c - v_b) * \cos\theta_x * \sin\theta_x} * \frac{1}{s} \tag{4}$$

Furthermore it can be seen that this distance is not dependent of the movement on the Y axis. The center of each object that is located at a distance d is on a line parallel to Y axis. This can also be observed in Figure 3a, where the blue line is the line that crosses the center of the image, and the green lines are situated at different distances d_1, d_2 and d_3 from the center.

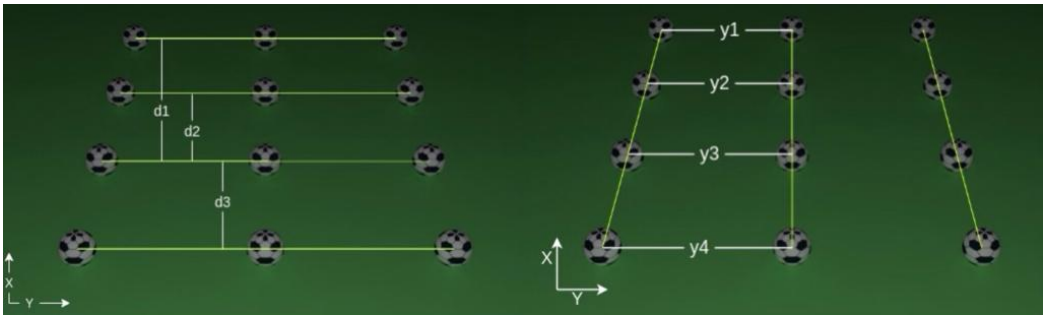

Fig. 3. a)Deformation on X axis b) Vanishing point

Position on Y axis. The perspective projection of any set of parallel lines which are not parallel to the image plane will converge to a "vanishing point" [3]. Due this perspective error, the dimensions of the objects in a picture will be affected. However, all objects situated at the same distance from the Y axis will be on the same line that converges to the vanishing point, as represented by the green lines in Figure 3b.

The distances y_1, y_2, y_3 and y_4 represent the distance expressed in pixels from the center of objects. In the real 3D space, the object are situated at the same distance one from another, while the distances in the picture differ, because of the perspective error. Knowing that the camera is oriented towards the scene center, it can be determined that the vanishing point will be situated on the X axis, the middle green line from Figure 3b, letting the possibility to express the distances as a simple linear equation:

$$\Delta_Y = a_y * \Delta_{pixels} + b_y \tag{5}$$

Δ_{pixels} represents the distance in pixels between object centers on Y axis.

Now, in order to determinate the slope a_y and the intercept b_y, the distance on X axis determined in the previous section will be used. With the vanishing point situated on the X axis, the slopes and intercepts will decrease inversely proportional with the distance between the camera and the object. Knowing the camera position and the distance between the objects and the center of the scene on X axis, Δ_x, the

slopes and intercepts can be expressed as a first-degree equation with their slopes and intercepts determined experimentally:

$$\begin{aligned} a_y &= a_s * \Delta_x + b_s \\ b_y &= a_i * \Delta_x + b_i \end{aligned} \tag{6}$$

Coordinates rotation. The two distances determined in the previous section were determined relatively to the current camera position. In order to obtain the real 3D world coordinates, Δ_x and Δ_y are rotated in the surface plane with ϕ, the camera polar coordinate from equation 1.

$$\begin{aligned} pos_x &= \Delta_x * \cos\phi + \Delta_y * \sin\phi \\ pos_y &= -\Delta_x * \sin\phi + \Delta_y * \cos\phi \end{aligned} \tag{7}$$

Where pos_x and pos_y represented the real 3D coordinates of the object.

3.3. Optimization

This section will describe the methods used for the target search, as well as the ones used in the optimization process. These methods relay on the object coordinates, pos_x and pos_y, and coordinates $camera_x$, $camera_y$ and $camera_z$ described in the previous sections. As input, there are two sets of pairs which contain the bounding boxes of the objects, from the initial view and from the current camera. For both sets, the positions of the objects are estimated based on the current camera position, with the goal being to minimize the difference between the two coordinates. The difference will be analyzed from two perspectives, based on the possible camera moves: on latitude (North and South) and on longitude (West and East). Finally, combining these two results, the next best step will be determined.

3.3.1. *Optimization on longitude.* Starting from the current camera view, the exact coordinates of the objects - pos_x and pos_y, can be determined. However, for the initial view, the position can only be estimated relatively to the initial camera, so Δ_X and Δ_Y can be found. Combining this information into equation 1, we can express the polar coordinate of the initial camera position as follows:

$$2\Delta_Y \cos\phi^2 - 2(pos_x + pos_y)(\Delta_X + \Delta_Y)\cos\phi + (pos_x + pos_y)^2 - 1 = 0 \tag{8}$$

In order to solve the equation, two sub-problems need to be approached. First of all, because Δ_X and Δ_Y are dependent on the height of the camera, the value of $\cos\phi$ will be approximated and not precise. Second of all, given the objects placed in the scene, a validation is required in order determine if object mapping between the initial stage view and the current stage view was done correctly.

Regarding the value obtained for $\cos\phi$, the problem will persist until the camera height will be found. At that point, the exact polar coordinate, ϕ, will be found. Even if the solution from equation 8 will not produce the exact solution, it will always point in right direction, letting the possibility to have a step-by-step approach. Figure 4 represents the equation of two approximations: one from the correct camera height (green) and one from an incorrect position (orange). The purple squares represent the closest points for the camera trajectory to follow in order to obtain the desired position. The first one will always be analyzed. The one on the orange line represents the desired position, with a slight error which comes

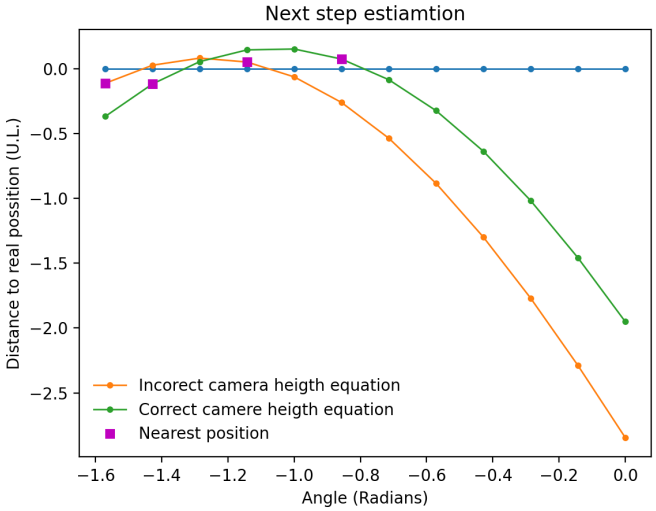

Fig. 4. Next step estimation

from the object mapping. As it can be observed, the solution for the green line is two steps away from the real target, but it is a real improvement from a simple step-by-step approach. At the next iteration, a closer camera height will be obtained, introducing the possibility to obtain a better position on the lateral movement.

3.**3**.2. *Optimization on latitude.* The object correspondence is also a problem that will affect the optimization on latitude. Knowing the equations for Δ_X and Δ_y, it is possible to state that for given pixels coordinates of an object, the distance between the object and the scene center will grow simultaneously with the camera height. Furthermore, the object-scene center distances corresponding to multiple objects will be preserved; consequently, the order of the objects will remain the same irrespective of any other parameters.

There are three projections of the positions of the camera on the scene:

- pos_z1 - corresponding to the light blue objects in Figure 5
- pos_z2 - corresponding to the light purple objects in Figure 5
- pos_z3 - corresponding to the grey objects in Figure 5

In Figure 5, the points on the right side are the real coordinates estimations of three objects - 1, 2 and 3 - which were made using the same pixel coordinates extracted from pos_z2. However, these coordinates could have also been extracted from pos_z1 or from pos_z3. As a result, there are two main conclusions:

- the objects will remain in the same order despite other parameters; this order is determined by the distance to the scene center;
- the greater the camera's height (from z1 to z3), the greater the distance from the objects to the center of the scene.

These conclusions confirm the ideas previously expressed, validating the correspondence between the objects from the initial picture and from the current picture. This also shows that it is not possible to put together an exact equation is not possible in order to have an accurate estimation. As a solution, it is possible to store all the

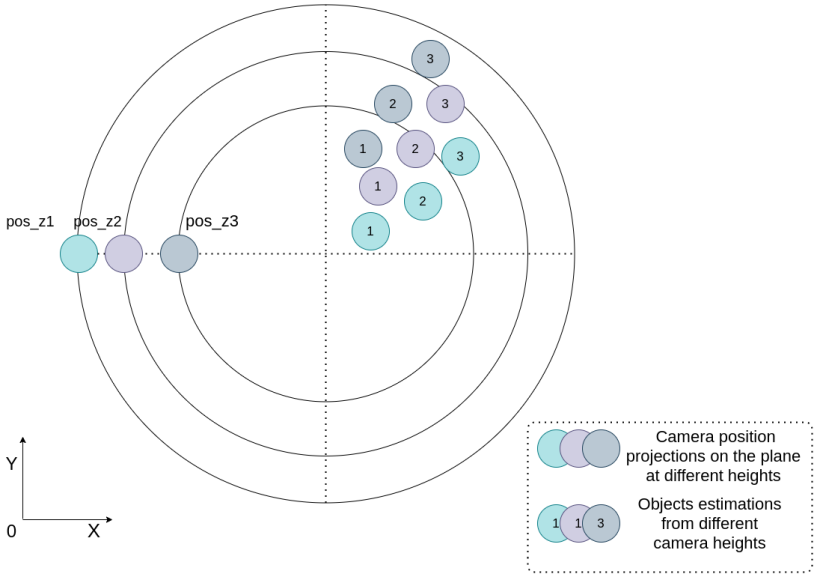

Fig. 5. Project architecture

desire data for object mapping, letting the possibility to test all the possible combinations. This approach will obtain the correct camera height in just one evaluation, by choosing the position from where the distance to the center of an object is the closest to the one from the current camera view.

4. **Results and evaluation**

To assess the results of this project we will consider the following categories of interest: the approximation of the positioning of the object mapping in the scene and the optimization of the search performed in the 3D space. The units of measurement are adapted to Mitsuba 2.0's system of measurement, having the generic name of 'units of length' (U.L.).

After implementing the algorithm that approximates the positioning of the objects in the scene, the results obtained were good, proving the correctness of the algorithm. First of all, for the approximation of Y axis, the slopes and intercepts were empirically determined, using the least square algorithm. Furthermore, by combining both estimations, on X axis, which is computed using equation 4 and on Y axis, which is empirically determined, the distance to the center of the scene can be estimated. This distance is calculated as square root of the sum of the estimations' squares.

To emphasize the results obtained for the optimized version of the search, a step-by-step search was performed at first. Figure 6a shows the results of this approach.

The blue line represents the trajectory obtained if only the longitude search is performed; the orange one, only the latitude search; the green one represents the case for which for the longitude and latitude search are done. For the first two, the desired height is reached and the desired ϕ, spherical coordinate of the camera, is reached respectively. As a result, the third one, which is the combination of these

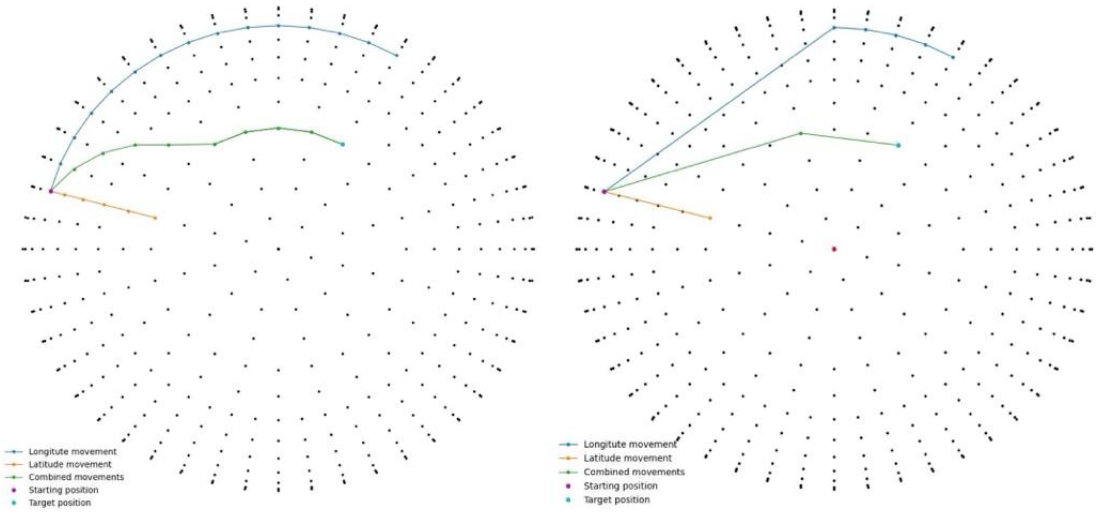

Fig. 6. a) Simple search example b) Optimized search example

two, will reach the desired position of the camera in the 3D space. In all three cases, the solution is reached in linear time, the slight changes being due to the fact that there are less positions of the camera for high values of height.

The trajectories of the optimized solutions are illustrated in Figure 6. Like previously, the blue line represents the longitude movement. The first step consists of reaching the closest point to the solution of equation 8. Since the exact position of the camera isn't available, the next steps will be done in the same direction, until the desired ϕ will be reached. As expected, the method used to obtain the height of the camera will acquire the solution within one step, represented by the orange line and also by the first step of the green line. The optimized solution reached the desired position within two steps, because the solution of equation 8 will be correct since the camera height value will be precise.

Figure 7 illustrates the comparison between the optimized and the simple solution. Even though the optimized search on latitude is not pointing towards the right solution after the first step, it is visible that it comes with an improvement of five steps.

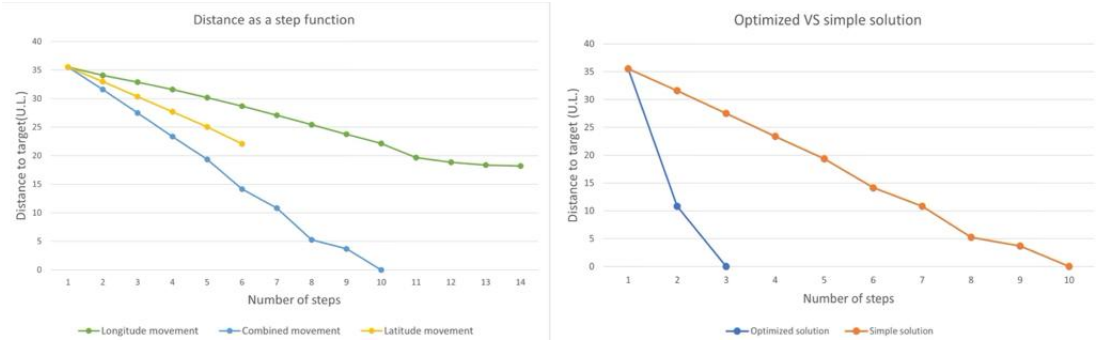

Fig. 7. Optimized vs simple solution

The number of steps for the simple solution depends on the number of camera states, which is given by the movement angle. In the worst case scenario, there are $\pi/angle\_step$ steps on longitude and $(\pi/2)/angle\_step$ steps on latitude. The improvement is expressed in percentage, by the following formula:

$$Improvement = -100 * \frac{3 * \pi - 4 * angle\_step}{3 * \pi} \tag{9}$$

In Figure 8, the results for $angle\_step = \pi/24$ are illustrated. It can be observed that the simple solution needs 36 steps to reach the solution, in the worst case scenario, while the optimized solution only requires 2 steps, with a decrease of -94.44% in the number of necessary steps to reach the optimum position.

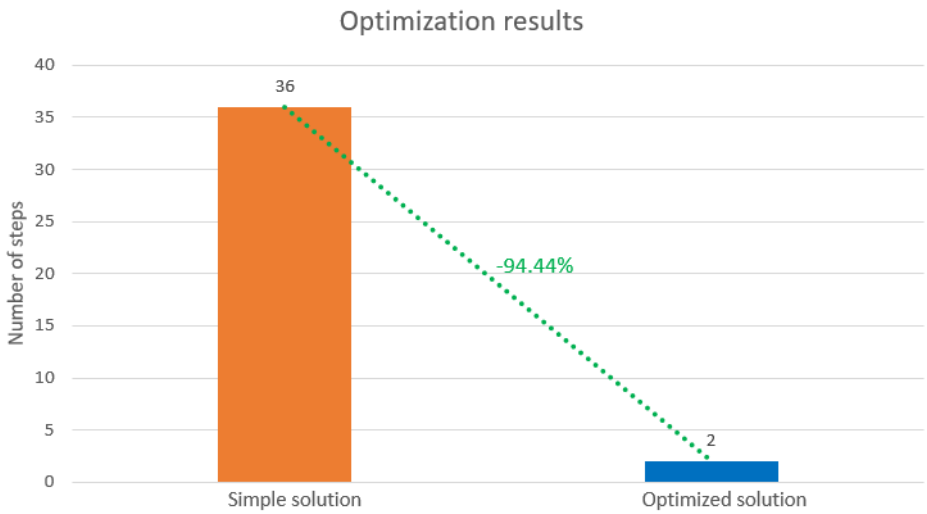

Fig. 8. Distance as a step function graphic

The validation of the simulation using the COSMOS framework was done on a significant number of starting camera positions. All instances concluded with the framework correctly identifying the desired original position. In the above evaluation we only gave a couple of relevant examples to illustrate the process.

5. Conclusion and further work

This paper focuses on how the COSMOS framework is an adequate solution for numerous kinds of simulations corresponding to real life problems. Its architecture allows the seamless integration of different tools in order to perform a consistent simulation, in a different domain to the one it was originally designed for. Also, we employed the post-optimization method of the Optimizer module within the COSMOS framework and obtained significantly improved results. An optimized search in a continuous 3D space was performed by assembling different tools within a single processing pipeline. The problem was split into three smaller problems, namely: the scene reconstruction, the object mapping, and the optimization. Scene reconstruction was tackled using the Mitsuba2.0 renderer, which turned out to be a good match with the COSMOS framework. The image processing and the object mapping part

used YOLO, also interacting as expected with COSMOS. The optimization component was designed to find the original position where the camera took the picture of the scene. The obtained results are good, proving that the search can be optimized and also that the COSMOS framework can be used in a software pipeline in conjunction with any application, not just in scientific computing.

Nevertheless, some constraints were encountered, and they can constitute the starting point for future work, in areas such as: recognizing objects in optimized searches, arbitrary camera orientation and movement, and real time processing.

REFERENCES

[1] *A. Rasch, H.M. Bücker.* Efcoss: An interactive environment facilitating optimal experimental design. April 2010. ACM Trans. Math. Softw. 37, 2, Article 13 (April 2010), 37 pages.

[2] *C.H. Bischof, H.M. Bücker, B. Lang.* An interactive environment for supporting the transition from simulation to optimization. Scientific Programming, vol. 11, no. 4, pp. 263-272, 2003.

[3] *Foley, James D and Van Dam, Andries and others.* Fundamentals of interactive computer graphics. 2, 1982. Addison-Wesley Reading, MA.

[4] *Hoiem, Derek and Efros, Alexei A and Hebert, Martial.* Putting objects in perspective. International Journal of Computer Vision, 80(1):3–15, 2008.

[5] *Larsen, Ask Hjorth and Mortensen, Jens Jørgen and others.* The atomic simulation environment—a python library for working with atoms. Journal of Physics: Condensed Matter, 29(27):273002, 2017.

[6] *Linchant, Julie and Lisein, Jonathan and Semeki, Jean and Lejeune, Philippe and Vermeulen, Cédric.* Are unmanned aircraft systems (uas s) the future of wildlife monitoring? a review of accomplishments and challenges. Mammal Review, 45(4):239–252, 2015.

[7] *Popović, Marija and Vidal-Calleja, Teresa and Hitz, Gregory and Sa, Inkyu and Siegwart, Roland and Nieto, Juan.* Multiresolution mapping and informative path planning for uav-based terrain monitoring. pages 1382–1388, 2017.

[8] *Redmon, Joseph and Divvala, Santosh and Girshick, Ross and Farhadi, Ali.* You only look once: Unified, real-time object detection. pages 779–788, 2016.

[9] *Samoila, Cosmin-Gabriel and Slusanschi, Emil-Ioan.* Cosmos-framework for combining optimization and simulation software. 22nd International Conference on Control Systems and Computer Science (CSCS), pp. 162-169, 2019.

[10] *Schittkowski, Klaus.* Easy-fit: A software system for data fitting in dynamical systems. 23:153–169, 03 2002.