AIAA SciTech Forum
23-27 January 2023, National Harbor, MD & Online
AIAA SCITECH 2023 Forum

10.2514/6.2023-2506

Vision-based Distributed Pose Estimation using Spacecraft Constellations

Saptarshi Bandyopadhyay*, Vinod Gehlot†, William Seto‡, Amir Rahmani§
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109, USA

Jonathan Becktor¶
Technical University of Denmark, Lyngby, Denmark

Spencer Kraisler‖, Shahriar Talebi**, Aditya Deole††, Niyousha Rahimi‡‡, Mehran Mesbahi§§
Dept. of Aeronautics & Astronautics, University of Washington, Seattle, Washington 98195, USA

In this paper, we present a simulation framework and consensus algorithms to track the pose (position and attitude) of an uncooperative/defunct spacecraft (*Target* spacecraft) using a fleet of small spacecraft (*Ego* spacecraft) in Low Earth Orbit (LEO). Since it is not possible to put traditional active sensors (like differential GPS, inter-satellite communication) on the uncooperative/ defunct Target spacecraft, the fleet of Ego spacecraft have to rely on cameras and inter-spacecraft communication among themselves to estimate the relative pose of the Target spacecraft. We present an end-to-end simulation pipeline for simulating high-fidelity orbital mechanics of multiple spacecraft in Earth orbit and generating photo-realistic images of the Target spacecraft using cameras onboard each Ego spacecraft. We then present a class of consensus algorithms (both in Eucledian and Riemannian manifolds) that help combine the individual pose estimates of the Target spacecraft from all the Ego spacecraft, under mild assumptions on the communication network topology. We provide numerical simulation results to prove the effectiveness of this approach.

I. Introduction

Spacecraft teams in Earth orbit can perform a variety of applications like in-space assembly of large-scale space structures and service of defunct spacecraft. The main focus of this paper is to track the pose (position and attitude) of an uncooperative/ defunct spacecraft (called *Target* spacecraft) using a fleet of small spacecraft (called *Ego* spacecraft). Since it is not possible to put traditional active sensors (like differential GPS, inter-satellite communication) on the uncooperative/ defunct Target spacecraft, the fleet of Ego spacecraft have to rely on cameras and inter-spacecraft communication among themselves to estimate the relative pose of the Target spacecraft. The main contributions of this paper are in these two areas:

1) We present an end-to-end simulation pipeline for simulating high-fidelity orbital mechanics of multiple spacecraft in Earth orbit, generating photo-realistic images of the Target spacecraft using cameras onboard each Ego spacecraft, independently estimating the pose of the Target spacecraft onboard each Ego spacecraft using state-of-the-art Convolutional and Recurrent Neural Networks (CNN and RNN) algorithms,[1] and robustly

*Robotics Technologist, Mobility and Robotic Systems Section, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California 91109, USA.
†Postdoctoral Researcher, Mobility and Robotic Systems Section, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California 91109, USA.
‡Robotics Technologist, Mobility and Robotic Systems Section, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California 91109, USA.
§Group Supervisor, Maritime and Multi-Agent Autonomy Group, Mobility and Robotic Systems Section, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California 91109, USA. **AIAA Life Member**
¶Ph.D. Student, Technical University of Denmark, Anker Engelunds Vej 1 Bygning 101A, 2800 Kgs. Lyngby, Denmark,
‖Ph.D. Student, Robotics, Aerospace and Information Networks (RAIN) Lab, University of Washington, Seattle, WA 98195, USA,
**Postdoctoral Researcher, Robotics, Aerospace and Information Networks (RAIN) Lab, University of Washington, Seattle, WA 98195, USA,
††Ph.D. Student, Robotics, Aerospace and Information Networks (RAIN) Lab, University of Washington, Seattle, WA 98195, USA,
‡‡Ph.D. Student, Robotics, Aerospace and Information Networks (RAIN) Lab, University of Washington, Seattle, WA 98195, USA,
§§Professor, Robotics, Aerospace and Information Networks (RAIN) Lab, University of Washington, Seattle, WA 98195, USA **AIAA Associate Fellow**

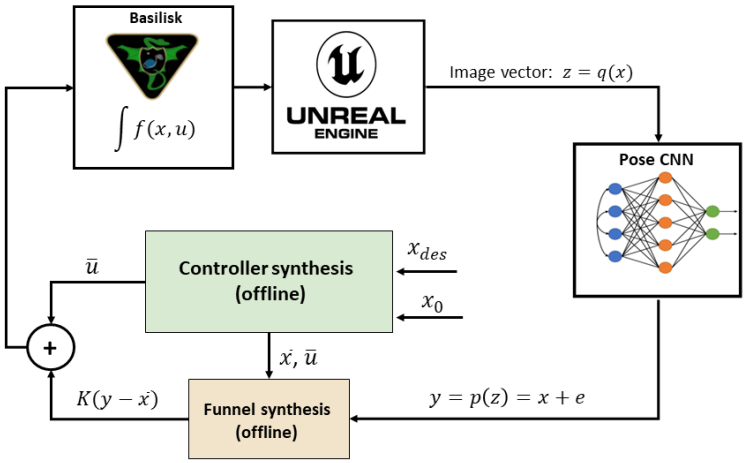


Figure 1 Vision Simulation Architecture from Ref. 1

maintaining the pose of each Ego spacecraft to further help in the estimation process. This paper builds on our prior work on the simulation pipeline for a single Ego spacecraft[2].

2) We present a class of consensus algorithms (both in Eucledian and Riemannian manifolds) that help combine the individual pose estimates of the Target spacecraft from all the Ego spacecraft, under mild assumptions on the communication network topology. We also provide bounds on the fused pose estimate of the Target spacecraft after the consensus process.

This paper is organized as follows: Sec. II describes the end-to-end simulation pipeline with one Target spacecraft and multiple Ego spacecraft. Sec. III describes known results on consensus algorithms in Euclidean space, to set the stage for the new results in the next section. Sec. IV desribes our new results on consensus algorithms in $SO(3)$ manifold. Sec. V provides numerical simulations to demonstrate the effectiveness of these new algorithms and Sec. VI concludes the paper.

II. End-to-end Simulation Pipeline

The End-to-end simulation pipeline provides a platform for rapid evaluation and testing of multi-agent algorithms involving vision subsystems. Multi-agent vision simulation is complex and requires a large amount of data. The simulation of vision-based estimation and control-based algorithms for space applications is a challenging task that involves choreographing several processes running at different timescales with numerous large data producers and consumers. The size of the 3D environment/scene, consisting of several meshes, materials, visual effects, and lighting, can quickly reach tens of gigabytes of data to render a single 1280 x 1024 grayscale image; gigabytes of data to render a single 2-megabyte image of a simulated vision camera. The simulation also needs to model the image transformation pipeline required by the pose estimator. This includes image resizing, rescaling, and segmentation to obtain the Region Of Interest (ROI). The simulation is furthermore used to create a dataset for the CNN models. These use the transformed image to produce a three, four, or nine-parameter attitude vector. The pipeline connecting the 3D scene to the attitude output forms Large-Scale (LS) processes in the simulation framework. The End-to-End Simulation (E2ES) pipeline also includes several Small Scale (SS) modules that solve differential equations and optimization problems: astrodynamics, spacecraft dynamics, and controllers, among others. Using the estimated attitude from the LS modules, the SS simulation modules compute the control and dynamics data that drives the LS visual pipeline and the state vectors of all the dynamic objects. The E2ES framework provides an agile framework for rapidly evaluating control and coordination algorithms for multi-agent spacecraft incorporating vision algorithms and subsystems.

In Ref. 2, we developed an end-to-end simulation pipeline to model the vision-based control and estimation for a single Ego-Target spacecraft pair. This pipeline simulated the scenario in which an Ego spacecraft estimated the attitude of a passive and uncooperative Target spacecraft using its onboard vision camera. Fig 1. shows the high-level block diagram of the simulation components described in Ref. 2. Here, Basilisk, Unreal Engine, Pose CNN, and Controller are the significant components of this pipeline. Basilisk models the planetary dynamics and the Ego and Target spacecraft dynamics. It receives inputs from the controller and supplies the state vector of the planetary objects and the spacecraft

to the Unreal Engine. The Unreal Engine is a 3D rendering pipeline that models the scene containing the planet and the target spacecraft and renders the 2D image modeling the Ego spacecraft's vision camera. The Pose CNN, implemented using PyTorch, produces an estimate of the target spacecraft using the rendered image of the scene from Unreal Engine. The Pose CNN also includes the image transformation pipeline, which resizes, segments, and obtains an ROI for the CNN. The controller, implemented in Python and MATLAB, uses the attitude information resulting from the Pose CNN and the state vector of all the components from Basilisk to synthesize the controller for the Ego spacecraft. The file-based data transfer accomplished the movement of data between the modules.

A. Motivation

Small spacecraft such as Cubesats and the recent Mars helicopter, Ingenuity, demonstrate Commercial-Of-The-Shelf (COTS) hardware's ability to successfully survive and operate in a space environment [cite mars-heli-hardware-paper]. Ingenuity, in particular, demonstrated its ability to scout, using vision and inertial sensors, the terrain to enhance the exploration capability of the Perseverance Rover greatly. The COTS hardware used in these small spacecraft has another advantage: reduced cost. For example, the reduced cost of their development could allow for multiple small, cooperative spacecraft to accompany high-budget missions to provide robust estimation from their unique vantage point. Many multi-agent estimation algorithms are at low NASA Technology Readiness Level (TRL) — i.e., TRL < 4 — and validation tools, including software and hardware, are needed to push the TRL levels of these algorithms and methodologies. Hence, the development of the multi-agent E2ES framework was motivated by the need to simulate simultaneously the hardware and software of vision-based control and estimations algorithms running on multiple robotic spacecraft.

subsectionEnd-to-End Simulation Requirements

The primary requirement of the E2ES framework is the provide low TRL multi-agent vision-based algorithms, the Software-In-the-Loop (SIL), and Hardware-In-the-Loop (HIL) simulation capability for testing and validation. Broader requirements for the E2ES framework include:

- Distributed multi-vehicle modular architecture integrating
  - Dynamics Module
  - 3D scene rendering pipeline
  - Controller Module
- Small code size (< 10KLoc)
- SIL and HIL simulation using simulated sensor data.

Although these requirements look generic, the central theme is multiple components over single components: Multiple dynamics modules over single dynamics modules, multiple 3D scenes rendering pipelines rather than a single rendering pipeline, and multiple controller modules over a single controller.

Distributed and Modular requirements are born out of necessity. For multi-agent simulation involving a few agents, it is possible to simulate the dynamics, controller, and vision pipeline on a single machine. However, with multiple 3D rendering and CNN pipelines, the physical hardware — for example, CPU, GPU & RAM — becomes a bottleneck. A distributed software architecture will allow for effortless scalability. Most of the E2ES simulation framework defines a "glue logic" between several small and large simulation components. Therefore, the glue components are naturally small and only handle the communication elements. There is another reason for purposely restricting the code size to less than the 10K Lines of Code (LoC): Evolution and Maintainability of the E2ES framework. Small teams will use the E2ES framework to push the TRL levels of their algorithms higher, but they will also evolve and maintain the algorithms to better suit their future needs. The restricted code size and good documentation will significantly enable the extensibility of the E2ES framework by small teams composed of control theory and algorithmic experts rather than professional software engineers. SIL and HIL simulation capability will enable the crucial Verification and Validation (V&V) of algorithms deployed to expensive robotic hardware, which is essential and required for pushing the TRL levels.

B. The End-to-End Simulation Architecture

Figure 1 shows the architecture for the E2ES framework that is currently under development. The E2ES framework consists of four major components: The Message Delivery Server, Unreal Engine, Pose Estimator, and the N+1 Ego/Target Spacecraft (SC). The Large Data (LD) and Small Data (SD) channels synchronize and transfer between the four major components. The architecture in Figure 1 inherits, in essence, the component structure of the single spacecraft visual simulation framework from Ref. 2. However, it vastly improves and streamlines the dataflow pipeline between components to allow for seamless vision-based multi-agent control and estimation simulation.

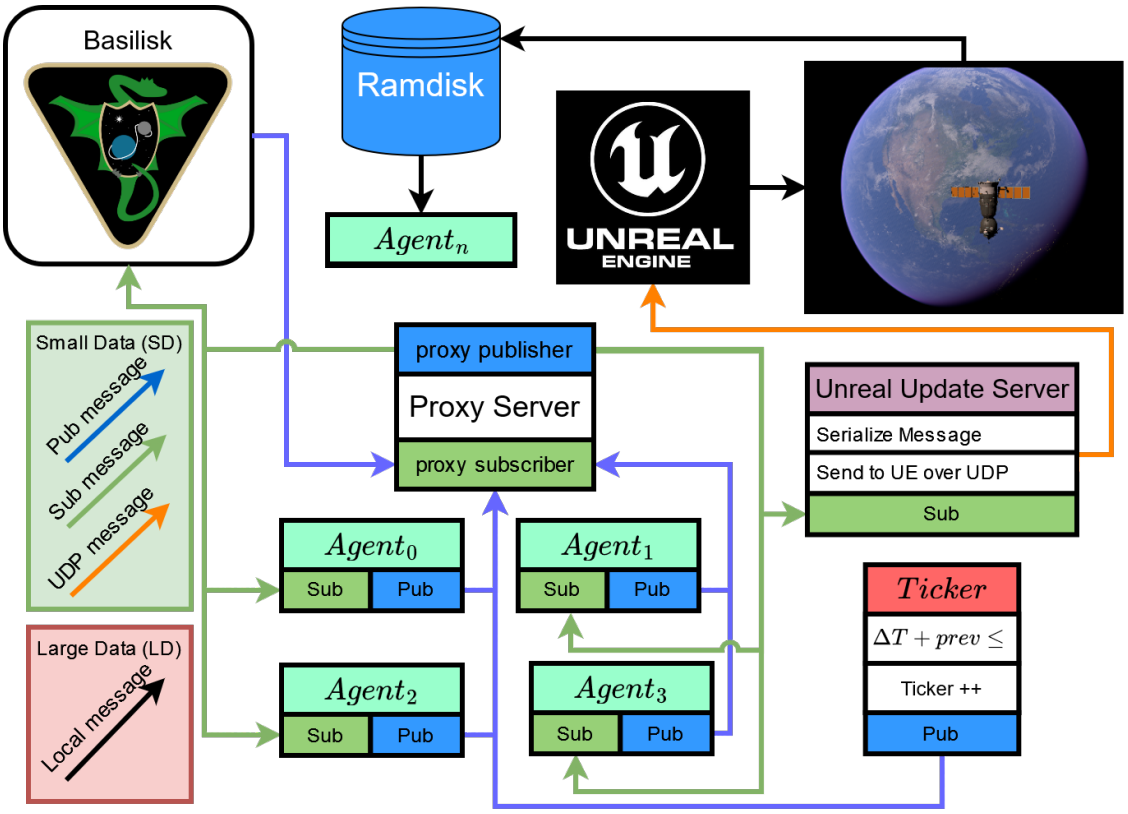


Figure 2 End-to-end Simulation Architecture, the figure shows four agent modules communicating using the MDS pub/sub architecture. These are synchronized with a ticker module to enforce correct synchronization. The Basilisk and Unreal Engine modules are also represented where the latter serializes messages from MDS to Unreal using UDP, and the former subscribes and publishes to the proxy server.

We are constructing the E2ES framework using a client-server architecture. The Message Delivery Server (MDS) is the central server to which the clients — controllers, dynamics modules, unreal engine, and pose estimator — connect using the SD channels. LD channel provides high-volume data transfer from Unreal Engine's rendered output to Pose Estimators input. The following sections provide a detailed discussion of the various components of the E2ES framework.

1. The Large and Small Data Channels

The central requirement for multi-agent simulation is the facility for communicating essential data between disparate components such as controllers, dynamics modules, 3D rendering engines, and estimators, among others. Often, we build these components in different environments; for example, we could model astrodynamics and the spacecraft dynamics in Basilisk, the controller in MATLAB, and the CNN estimator in PyTorch. The previous architecture Ref. 2 used a more pragmatic file transfer approach to share data between these components. While this technique worked well for slow, 1-second loop-rate spacecraft dynamics, it scales poorly with higher loop rates and when multi-agent spacecraft are required. Besides scalability, the file transfer approach can restrict the simulation using distributed resources.

Not all data is the same. Dynamic simulation variables such as a spacecraft's state, control, and measurement vectors are a collection or an array of a few floating-point variables. On the other hand, the 3D rendering pipeline's output, an image, is orders of magnitude larger than dynamic simulation vectors. Although creating a single communication channel that transports the two kinds of data is possible, the associated software is complex, making extensibility and maintainability challenging. Hence, we decided on two different communication topologies: the Large Data (LD) channel for carrying the visual (image) data and Small Data (SD) channel for transporting dynamic vector data. The LD channel is restricted for local data transfer — that is, data transfer between components on the same machine. This

restriction is designed since instances of the 3D rendering pipeline (Unreal Engine), and the Pose CNN are closely coupled and will reside on the same machine. However, the SD channel is distributed, allowing for simulation vector data transfers across machines and enabling client-server communication between MDS and the various simulation components.

Sockets, including TCP/UDP, are available in every operating system, including embedded platforms, and accessible from nearly every language, including Python and MATLAB; therefore, we use them to implement the SD channel. Shared memory is the fastest way to transfer large quantities of data between components. In the E2ES framework, we use the shared memory indirectly using a RAM disk to implement the LD channel; therefore, from the perspective of the Unreal Engine and the Pose estimator, transferring and accessing data is a simple file read-write operation.

2. The Message Delivery Server

The Message Delivery Server (MDS) is a multi-threaded server application that provides communication of simulation vector data between various components. It also provides the master clock for all the simulation events. The MDS server uses the SD channel to interconnect the various simulation components, including agents, the 3D rendering pipeline, and the pose CNN. Furthermore, it is also the core element of the E2ES framework, and it is responsible for starting, initializing, running, and terminating the simulation. A user-generated setup script provides MDS with the information needed to drive the simulation, including the connection topology, which the MDS uses to generate a routing table for the messages.

The MDS is the server in the client-server architecture of the E2ES framework. Internally, the MDS follows a subscriber/publisher architecture with a proxy server providing a routing table. A client could, for example, an instance of the Basilisk module, periodically initiate a request to obtain the latest estimated pose information from the agent. Another client, for example, the agent, initiates requests to update its pose information in the Unreal module. The MDS server maintains a thread pool, and each request to the MDS server runs in a separate thread. For synchronization between threads, we apply mutex-locks. A typical E2ES user will only need to define a simple setup script containing information about agent dynamics, a graph defining inter-agent communication topology, controller, simulation time information, and other standard simulation elements. Most simulation circumstances will not require a user to delve into the low-level communication infrastructure of the MDS.

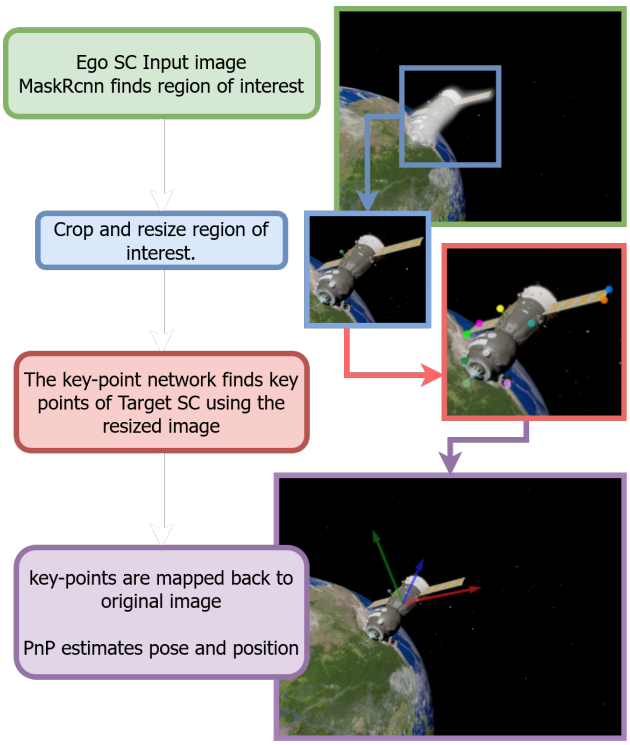
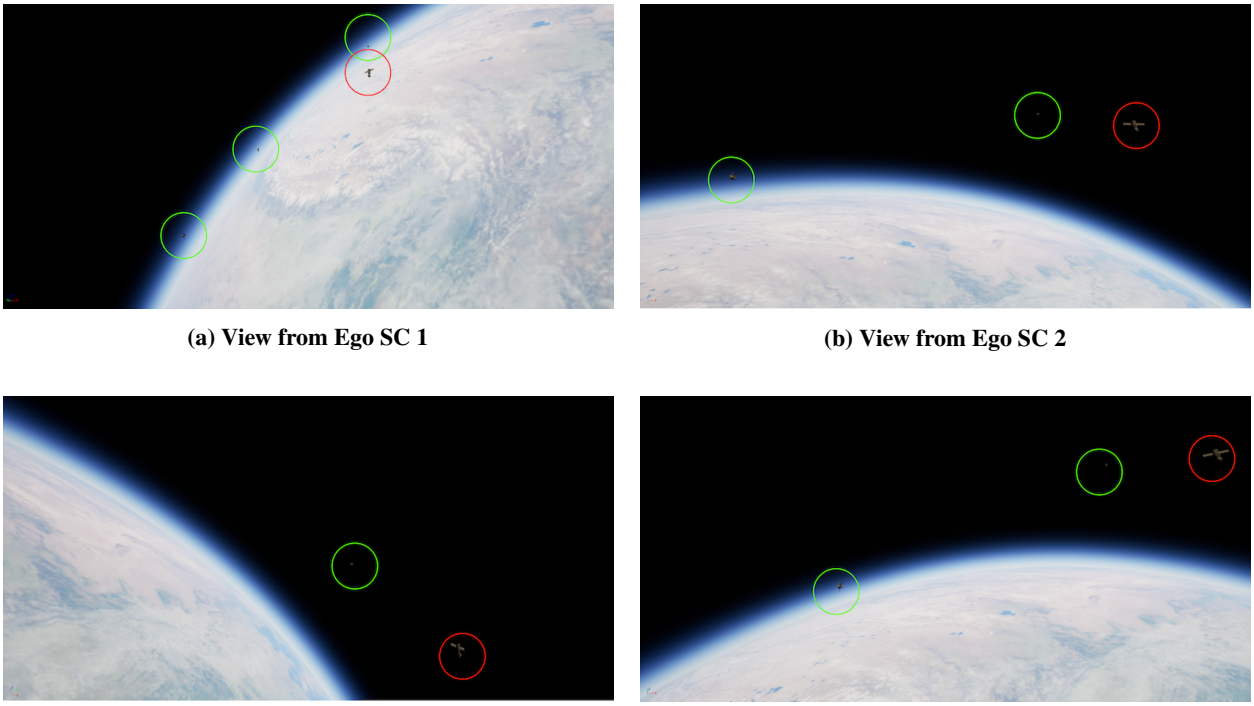


Figure 3 Pose Estimation Pipline (Ref. 2)



(a) View from Ego SC 1

(b) View from Ego SC 2

(c) View from Ego SC 3

(d) View from Ego SC 4

Figure 4 View from four Ego spacecraft

3. The Vision Pipeline: Unreal Engine and Pose estimation

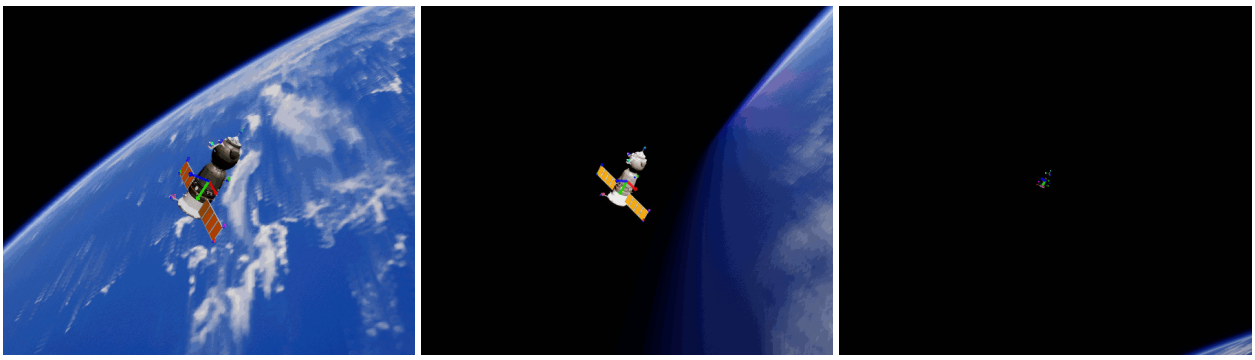
Unreal Engine and Pose estimation is the most data-heavy, computationally expensive parts of the E2ES simulation framework; Unreal Engine is the producer, and Pose CNN is the consumer. There are two significant improvements that the E2ES framework implements in the visual pipeline, which are different from the simulation discussed in Ref. 2: multi-agent capability and better performance.

The gaming industry has pioneered the work of realistic high-fidelity rendering of 3D worlds. The E2ES framework leverages the rendering capabilities of Unreal Engine to create multiple instances of the visual rendering pipeline. There are N instances corresponding to N Ego spacecraft in the E2ES simulation; each instance models the camera view associated with the Ego spacecraft. Fig. 4 shows four rendering instances running simultaneously. Using the SD channel, a custom actor object connects each UE instance to the MDS server. Introducing a Tick module decouples the synchronization from UE, and for every tick sent to MDS, each agent instance requests the latest pose vector from the MDS server and updates the position and attitude of the camera and spacecraft in Unreal. Each rendering instance writes the rendered camera view to the LD channel connected to the pose estimation module. A UE render – actor module pair exists for each spacecraft in the E2ES framework, and it resides locally. However, we can distribute N pairs across M machines and connect their outputs to the MDS server using SD channels.

The pose estimation pipeline, see fig 3, determines the position and attitude estimate of the target spacecraft using the rendered image from UE. The Pose estimation module includes an image transformation subsystem that resizes and segments the image to the region surrounding the target before supplying it to the trained key point CNN. Given the camera information and the target objects 3d keypoints, the detected 2d-keypoints are then used to create a 2d-3d correspondence. This allows for the use of PnP to get the pose. The models are ported to TorchScript for inference in C++ to decouple python execution from the pose estimation module.

4. Spacecraft System

N spacecraft subsystems form a multi-agent system in a typical simulation run. The spacecraft system component consists of three tightly coupled modules that model either the Ego or the Target spacecraft dynamics: spacecraft



(a) 25m **(b) 50m** **(c) 200m**

Figure 5 Pose estimation of target SC at 25m, 50m and 200m distances.

dynamics, controller, and MDS message client. This component of the E2ES framework requires maximum flexibility. Dynamic models can change to accommodate the required fidelity and rapid prototyping and testing of different controllers. Moreover, we commonly implement dynamics and controllers in Python or MATLAB. The E2ES framework provides a Call-Return-based interface to the MDS message client in Python and MATLAB for maximum flexibility. Basilisk is a high-performance astrodynamics package with Python bindings and is supported by the MDS message client.

The MDS message client offers two-way communication of dynamics, pose, controller, and other vector data between the spacecraft subsystem and the MDS server using the SD channel. That way, the spacecraft system can easily communicate with Unreal Engine the Pose CNN.

III. Distributed Estimation in Euclidean space

Distributed pose estimation algorithms of Target spacecraft can be decomposed into consensus algorithms for position and velocity estimation in $\mathbb{R}^n$ and attitude estimation in $SO(3)$.

A. Notations

Let $\mathbb{N}$ and $\mathbb{R}$ represent the sets of positive integers and real numbers respectively, and $SO(3)$ is the group of all rotations about the origin in $\mathbb{R}^3$. Let $\exp(\cdot)$ and $|\cdot|$ denote the natural exponential function and cardinality of a set respectively.

Let there by $N \in \mathbb{N}$ Ego spacecraft. Superscript denotes the agent index and subscript represents the time index. Therefore, $\boldsymbol{x}_k^i \in \mathbb{R}^n$ represent the estimated Target spacecraft's states by the i^{th} agent at the k^{th} time step.

The time-varying communication network topology among the agents is represented by the undirected graph $\mathcal{G}_k = \{\mathcal{V}, \mathcal{E}_k\}$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of agents and $\mathcal{E}_k$ is the set of active communication links at the k^{th} time step. Let $\mathcal{N}_k^i$ denote the set of neighbors of the i^{th} agent at the k^{th} time step:

$$\mathcal{N}_k^i = \{\ell \in \mathcal{V} | \{i, \ell\} \in \mathcal{E}_k, \ell \neq i\} \ . \tag{1}$$

B. Consensus of States in $\mathbb{R}^n$

Let there be $M \in \mathbb{N}$ consensus loops per time step. Let the state $\boldsymbol{x}_{k,j}^i$ represent the i^{th} agent's consensual state during the j^{th} consensus loop at the k^{th} time step.

Assumption 1. *We assume that the time-varying undirected graph $\mathcal{G}_k$ is strong-connected and constant (doesn't change) within each time step.*

Then the consensus algorithm for states in $\mathbb{R}^n$ is given by Algorithm 1.[3–5]

Algorithm 1 Consensus of states in $\mathbb{R}^n$

(i^{th} Ego spacecraft's steps at k^{th} time step)
Initialize $\boldsymbol{x}^i_{k,0} = \boldsymbol{x}^i_k$
for $j \in \{1, \ldots, M\}$ **do**

$$\boldsymbol{x}^i_{k,j} = \boldsymbol{x}^i_{k,j-1} + \sum_{\ell \in \mathcal{N}^i_k} a_k(i,\ell)\left(\boldsymbol{x}^\ell_{k,j-1} - \boldsymbol{x}^i_{k,j-1}\right) \tag{2}$$

$$\text{where} \qquad a_k(i,\ell) = \frac{1}{1 + \max\{|\mathcal{N}^i_k|, |\mathcal{N}^\ell_k|\}} \tag{3}$$

end for
return $\boldsymbol{x}^i_{k,M}$

Theorem 1. *Under Assumption 1, in Algorithm 1 each agent's consensual state exponentially converges to the mean of the initial states:[3–5]*

$$\lim_{j\to\infty} \boldsymbol{x}^i_{k,j} = \frac{1}{N}\sum_{\ell\in\mathcal{V}} \boldsymbol{x}^\ell_{k,0}\,, \qquad \forall i \in \mathcal{V}\,. \tag{4}$$

Proof. Let $\boldsymbol{y}_{k,j}$ represent the combined consensual states of all the agents:

$$\boldsymbol{y}_{k,j} = \begin{bmatrix} \boldsymbol{x}^1_{k,j} & \boldsymbol{x}^2_{k,j} & \ldots & \boldsymbol{x}^i_{k,j} & \ldots & \boldsymbol{x}^N_{k,j} \end{bmatrix}^T. \tag{5}$$

The discrete time collective dynamics of the network under Eq. (2) is given by:

$$\boldsymbol{y}_{k,j} = \mathbf{A}_k \boldsymbol{y}_{k,j-1}\,, \tag{6}$$

where

$$\mathbf{A}_k(i,\ell) = \mathbf{A}_k(\ell,i) = \tfrac{1}{1+\max\{|\mathcal{N}^i_k|,|\mathcal{N}^\ell_k|\}}\,, \qquad \forall \ell \in \mathcal{N}^i_k \tag{7}$$

$$\mathbf{A}_k(i,\ell) = 0\,, \qquad \forall \ell \notin \mathcal{N}_i \cup \{i\} \tag{8}$$

$$\mathbf{A}_k(i,i) = 1 - \sum_{\ell\in\mathcal{V}\setminus\{i\}} \mathbf{A}_k(i,\ell) \tag{9}$$

The state $\boldsymbol{y}_{k,j}$ in Eq. (6) can also be represented as:

$$\boldsymbol{y}_{k,j} = \mathbf{A}^j_k \boldsymbol{y}_{k,0}\,, \qquad \forall j \in \mathbb{N} \tag{10}$$

The matrix $\mathbf{A}_k$ is row and column stochastic, non-negative with positive diagonal elements, and irreducible. Then it follows from [6, Lemma 8.5.2, 8.5.4] that the matrix $\mathbf{A}_k$ is primitive. Then, it follows from Perron's theorem [6, Theorem 8.5.1, pp 516] that:

$$\lim_{j\to\infty} \mathbf{A}^j_k = \frac{1}{N}\mathbf{1}\mathbf{1}^T \tag{11}$$

$$\text{where } \mathbf{1} = \begin{bmatrix} 1 & \ldots & 1 & \ldots & 1 \end{bmatrix}^T$$

Moreover, the rate of convergence is bounded by:

$$\|\mathbf{A}^j_k - \frac{1}{N}\mathbf{1}\mathbf{1}^T\|_\infty \le C|\lambda_{N-1}(\mathbf{A}_k)|^j \tag{12}$$

where $\lambda_{N-1}(\mathbf{A}_k)$ is the second largest eigenvalue of the matrix $\mathbf{A}_k$ and C is a constant. The final consensual state $\boldsymbol{y}_{k,j}$

in Eq. (10) is given by:

$$\lim_{j\to\infty} \boldsymbol{y}_{k,j} = \mathbf{1}\left(\frac{1}{N}\mathbf{1}^T \boldsymbol{y}_{k,0}\right) = \mathbf{1}\left(\frac{1}{N}\sum_{\ell\in\mathcal{V}} \boldsymbol{x}^{\ell}_{k,0}\right) \tag{13}$$

$$\lim_{j\to\infty} \boldsymbol{x}^{i}_{k,j} = \frac{1}{N}\sum_{\ell\in\mathcal{V}} \boldsymbol{x}^{\ell}_{k,0}\,, \qquad \forall i \in \mathcal{V} \tag{14}$$

Hence each agent reaches consensus to the mean of the initial values. □

IV. Distributed Estimation in $SO(3)$ Manifold

Next, we focus on the consensus algorithms for estimating the Target spacecraft's attitude in $SO(3)$. Note the standard consensus algorithm in $\mathbb{R}^n$ does not directly apply to this case due to the nonlinear geometry of $SO(3)$ as a Riemannian manifold and the difficulty with interpolating rotation matrices.

First, we wish to include a brief review of concepts from Riemannian geometry. Let $\mathcal{M}$ be a complete Riemannian manifold and $x, y \in M$. Define Γ to be the set of all smooth constant-speed curves γ defined on a compact interval $[t_0, t_1]$ that begin at x and end at y. Let $\Lambda(\gamma) := \int_0^1 \|\dot{\gamma}(t)\|_{\gamma(t)} dt$ denote the length of the curve γ. We call $\gamma \in \Gamma$ a *geodesic* if it locally minimizes Λ over Γ, and a *minimizing* geodesic if it globally minimizes Λ. Since M is complete, a minimizing geodesic always exists connecting any pair or points.

Let $C \subset \mathcal{M}$. Suppose for any $x, y \in C$, there exists a *unique* minimizing geodesic connecting the two points whose image is contained in C. Then we call C *geodesically convex*.

A. Notations

Let $\boldsymbol{R} \in \mathrm{SO}(3)$ represent a rotational matrix, where the manifold $\mathrm{SO}(3)$ is defined as:[7]

$$\mathrm{SO}(3) = \left\{\boldsymbol{R} \in \mathbb{R}^{3\times 3} | \boldsymbol{R}\boldsymbol{R}^T = \mathbf{I}, \det(\boldsymbol{R}) = 1\right\}. \tag{15}$$

The rotation matrix has three independent dimensions, hence the Euler representation $\{\boldsymbol{e} \in \mathbb{R}^3, \|\boldsymbol{e}\|_2 = 1, \theta \in [-\pi, \pi]\}$ of the rotation matrix is given by:[7]

$$\boldsymbol{R} = \cos\theta \mathbf{I} + (1 - \cos\theta)\boldsymbol{e}\boldsymbol{e}^T + \sin\theta \boldsymbol{e}^{\wedge}, \tag{16}$$

$$\text{where } \boldsymbol{e}^{\wedge} = \begin{bmatrix} 0 & -\boldsymbol{e}(3) & \boldsymbol{e}(2) \\ \boldsymbol{e}(3) & 0 & -\boldsymbol{e}(1) \\ -\boldsymbol{e}(2) & \boldsymbol{e}(1) & 0 \end{bmatrix}. \tag{17}$$

We call $(.)^{\wedge} : \mathbb{R}^3 \to \mathfrak{so}(3)$ the hat map, and its inverse $(.)^{\vee} : \mathfrak{so}(3) \to \mathbb{R}^3$ the vee map.

The interpolation of rotational matrices $\boldsymbol{R}_1, \boldsymbol{R}_2 \in \mathrm{SO}(3)$ is given by:[7]

$$\boldsymbol{R} = \left(\boldsymbol{R}_2\boldsymbol{R}_1^T\right)^{\alpha}\boldsymbol{R}_1 := \exp\left(\alpha \log(\boldsymbol{R}_2\boldsymbol{R}_1^T)\right)\boldsymbol{R}_1, \tag{18}$$

for any weight $\alpha \in \mathbb{R}$.

The Lie algebra associated with $\mathrm{SO}(3)$ is the space of skew-symmetric matrices, and is given by:[7]

$$\text{vectorspace} : \mathfrak{so}(3) = \left\{\boldsymbol{\Phi} = \boldsymbol{\phi}^{\wedge} | \boldsymbol{\phi} \in \mathbb{R}^3\right\}, \tag{19}$$

$$\text{field} : \mathbb{R}, \tag{20}$$

$$\text{Lie bracket} : [\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2] = \boldsymbol{\Phi}_1\boldsymbol{\Phi}_2 - \boldsymbol{\Phi}_2\boldsymbol{\Phi}_1, \tag{21}$$

$$\text{where } \boldsymbol{\phi}^{\wedge} = \begin{bmatrix} 0 & -\boldsymbol{\phi}_3 & \boldsymbol{\phi}_2 \\ \boldsymbol{\phi}_3 & 0 & -\boldsymbol{\phi}_1 \\ -\boldsymbol{\phi}_2 & \boldsymbol{\phi}_1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{22}$$

Let us define $(\cdot)^{\vee}$ as the inverse of the $(\cdot)^{\wedge}$ notation:

$$\boldsymbol{\Phi} = \boldsymbol{\phi}^{\wedge} \Longrightarrow \boldsymbol{\phi} = \boldsymbol{\Phi}^{\vee}. \tag{23}$$

We can relate the elements of SO (3) to the elements of $\mathfrak{so}(3)$ through the exponential map:[7]

$$\boldsymbol{C} = \exp\left(\boldsymbol{\phi}^{\wedge}\right). \tag{24}$$

We can also go in the other direction (not-uniquely) using:[7]

$$\boldsymbol{\phi} = \log\left(\boldsymbol{C}\right)^{\vee}. \tag{25}$$

B. Geodesic Distance

A Riemannian metric often used with SO (3) is $\langle u, v\rangle_{\boldsymbol{R}} = \frac{1}{2}\operatorname{tr}(u^T v)$ for $\boldsymbol{R} \in \mathrm{SO}(3)$ and $u, v \in T_{\boldsymbol{R}}SO(3)$. Under this metric, the geodesic distance has a closed-form expression for nearly every pair of points $\boldsymbol{R}_1, \boldsymbol{R}_2 \in SO(3)$. Suppose $\log(\boldsymbol{R}_1^T \boldsymbol{R}_2)$ exists. Then their geodesic distance is:

$$d_g^2(\boldsymbol{R}_1, \boldsymbol{R}_2) = -\frac{1}{2}\operatorname{tr}\left(\log(\boldsymbol{R}_1^T \boldsymbol{R}_2)^2\right) \tag{26}$$

Theorem 2. *The maximum geodesic distance in the $SO(3)$ manifold is bounded by π.*

Proof. Let $C_1, C_2 \in SO(3)$. Then, by bi-invariance property of this Riemannian metric, we obtain that $d(C_1, C_2) = d(C_2^T C_1, I)$. Thus, it suffices to find an upper bound of $d(C, I)$ for an arbitrary $C \in SO(3)$. But, we know that, for any such C, there exist a vector $v \in \mathfrak{so}(3)$ with unit norm and $t_0 \in [0, \pi]$ such that $C = \exp(t_0 v)$ [7, 7.25]. Note that $\|v\| := \sqrt{\langle v, v\rangle} = \sqrt{\frac{1}{2}\operatorname{tr}(v^T v)} = 1$, and thus $\gamma(t) := \exp(tv)$ where t ranges over $[0, t_0]$ is a geodesic starting at I and ending at C with unit velocity. Therefore, the geodesic length of γ is equal to $t_0 \leq \pi$ which proves the claim because $d(C, I)$ coincides with a minimization over all such geodesics. □

C. Consensus of States in $SO(3)$

Let the rotation matrix $\boldsymbol{R}_k^i \in SO(3)$ represent the i^{th} spacecraft's estimated mean of the Target spacecraft's attitude. The corresponding Euler representation is denoted by $\{\boldsymbol{e}_k^i, \theta_k^i\}$.

Let the rotation matrix $\boldsymbol{C}_{k,j}^i$ represent the i^{th} agent's consensual state during the j^{th} consensus loop at the k^{th} time step. Then the consensus algorithm for rotational matrices in $SO(3)$ is given by Algorithm 2. The term $\texttt{ascending}(\mathcal{N}_k^i)$ in line 6 denotes that the for-loop is executed in ascending order of spacecraft indices, since the order of matrix multiplication is important.

Assumption 2. *All the initial rotational matrices $\boldsymbol{R}_k^i, \forall i \in \mathcal{V}$ are in the same convex ball of the $SO(3)$ manifold, i.e. the maximum geodesic distance between the initial rotational matrices $\boldsymbol{R}_k^i$ is bounded by $\frac{\pi}{2}$:*

$$d_g(\boldsymbol{R}_k^i, \boldsymbol{R}_k^\ell) < \frac{\pi}{2}, \qquad \forall i, \ell \in \mathcal{V}. \tag{33}$$

Theorem 3. *Under Assumptions 1 and 2, if the initial rotation matrices are commutative (i.e. $\boldsymbol{R}_k^i = \{\boldsymbol{e}_k^\star, \theta_k^i\}$ for all Ego spacecraft where $\boldsymbol{e}_k^\star$ is a common constant and $|\theta_k^i - \theta_k^\ell| \leq \pi$), then, in Algorithm 2, each agent's consensual rotation matrix exponentially converges to the mean of the initial rotation matrices:*

$$\lim_{j\to\infty} \boldsymbol{C}_{k,j}^i = \prod_{\ell\in\mathcal{V}} (\boldsymbol{R}_k^\ell)^{\frac{1}{N}}, \qquad \forall i \in \mathcal{V}. \tag{34}$$

Proof. Let us first prove the special case for commutative matrices, i.e. $\boldsymbol{R}_k^i = \{\boldsymbol{e}_k^\star, \theta_k^i\}$ for all Ego spacecraft. Using a series of algebraic manipulations, we can restate Eqs. (31)–(32) as:

$$\boldsymbol{C}_{k,j}^i = \left(\prod_{\ell\in\mathcal{N}_k^i} \left(\boldsymbol{C}_{k,j-1}^\ell (\boldsymbol{C}_{k,j-1}^i)^{-1}\right)^{a_k(i,\ell)}\right) \boldsymbol{C}_{k,j-1}^i, \tag{35}$$

$$\boldsymbol{C}_{k,j}^i = \left(\prod_{\ell\in\mathcal{N}_k^i} \left(\boldsymbol{C}_{k,j-1}^\ell\right)^{a_k(i,\ell)}\right) (\boldsymbol{C}_{k,j-1}^i)^{a_k(i,i)}, \tag{36}$$

Algorithm 2 Consensus of states in $SO(3)$

(i^{th} Ego spacecraft's steps at k^{th} time step)
Initialize $\boldsymbol{C}^i_{k,0} = \boldsymbol{R}^i_k$
Pre-compute weights $a_k(i,\ell)$

$$a_k(i,\ell) = \frac{1}{1+\max\{|\mathcal{N}^i_k|, |\mathcal{N}^\ell_k|\}}, \qquad \forall \ell \in \mathcal{N}^i_k \tag{27}$$

$$a_k(i,\ell) = 0, \qquad \forall \ell \notin \mathcal{N}^i_k \cup \{i\} \tag{28}$$

$$a_k(i,i) = 1 - \sum_{\ell \in \mathcal{V}\setminus\{i\}} a_k(i,\ell) \tag{29}$$

for $j \in \{1,\dots,M\}$ **do**

$$\boldsymbol{C}^i_{k,j} = \boldsymbol{C}^i_{k,j-1} \tag{30}$$

  for $\ell \in \texttt{ascending}(\mathcal{N}^i_k)$ **do**

$$\boldsymbol{C}^i_{k,j} = \left(\left(\boldsymbol{C}^\ell_{k,j-1}(\boldsymbol{C}^i_{k,j})^T\right)^{\alpha_k(i,\ell)}\right)\boldsymbol{C}^i_{k,j} \tag{31}$$

$$\text{where } \alpha_k(i,\ell) = \frac{a_k(i,\ell)}{1-\sum_{\nu=\ell+1,\nu\neq i}^{N} a_k(i,\nu)} \tag{32}$$

  end for
end for
return $\boldsymbol{C}^i_{k,M}$

Restating this using elements of $\mathfrak{so}(3)$:

$$\exp\left((\boldsymbol{\phi}^i_{k,j})^\wedge\right) = \exp\left(a_k(i,i)(\boldsymbol{\phi}^i_{k,j-1})^\wedge + \sum_{\ell\in\mathcal{N}^i_k} a_k(i,\ell)(\boldsymbol{\phi}^\ell_{k,j-1})^\wedge\right) \tag{37}$$

Within the exponential, this equation is analogous to Eq. (2), hence it follows from Theorem 3 that:

$$\lim_{j\to\infty} \exp\left((\boldsymbol{\phi}^i_{k,j})^\wedge\right) = \exp\left(\frac{1}{N}\sum_{\ell\in\mathcal{V}}(\boldsymbol{\phi}^\ell_{k,0})^\wedge\right) \tag{38}$$

$$\lim_{j\to\infty} \boldsymbol{C}^i_{k,j} = \prod_{\ell\in\mathcal{V}} (\boldsymbol{C}^\ell_{k,0})^{\frac{1}{N}}, \quad \forall i \in \mathcal{V}. \tag{39}$$

Remark 1. *Under Assumptions 1 and 2, we have observed—through extensive Monte Carlo analysis—that agent's rotation matrices in Algorithm 2 always reach consensus (even when they are not commutative), i.e.,*

$$\lim_{j\to\infty} \boldsymbol{C}^i_{k,j} - \boldsymbol{C}^\ell_{k,j} = 0, \qquad \forall i \in \mathcal{V}. \tag{40}$$

However, a theoretical guarantee for this observation depends on much deeper geometric properties of the "positively curved" $SO(3)$ Lie group. In particular, one can easily argue that the updates in (31) does not leave the "geodesic convex hull" of the initial rotation matrices, but in general, arguing the convergence requires further analysis of the boundary of this convex body. This is deferred to our follow-up work.

□

V. Numerical Simulations

We will also present numerical results to demonstrate the effectiveness of these algorithms.

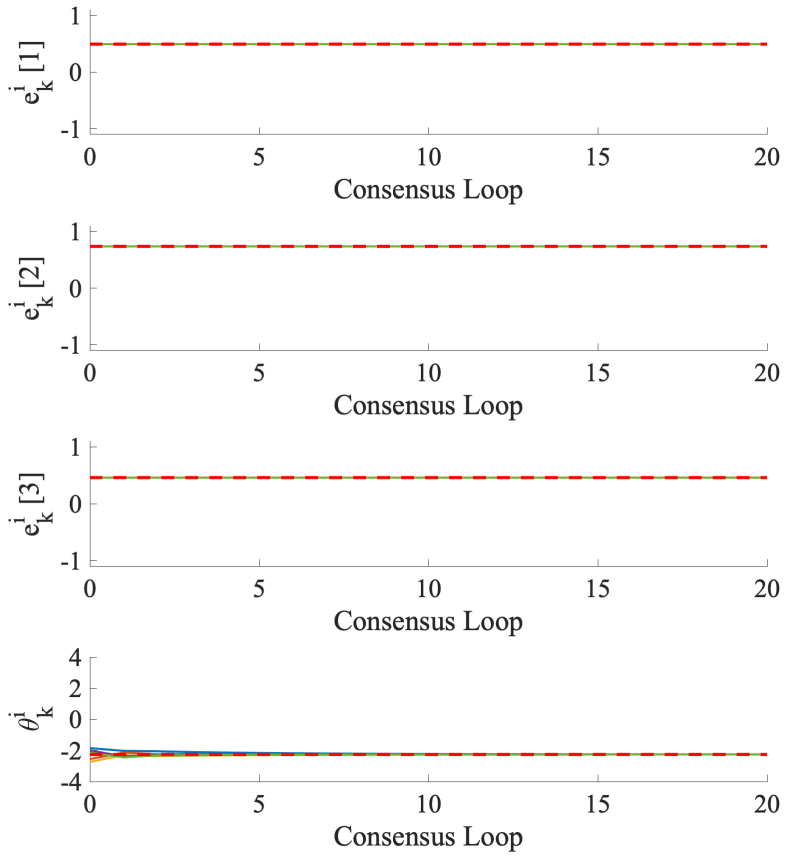


Figure 6 Consensus of 5 agents in the special case

A. Special Case

In the special case in Theorem 3, the initial rotation matrices are commutative, i.e. $\boldsymbol{R}_k^i = \{\boldsymbol{e}_k^\star, \theta_k^i\}$ for all Ego spacecraft and $\boldsymbol{e}_k^\star$ is a common constant. Here we show consensus of five agents, where they start from the following initial conditions:

$$\boldsymbol{e}_k^\star = [0.4959 \quad 0.7367 \quad 0.4598]\,, \tag{41}$$

$$\theta_k^i \in \{-1.8499, \quad -2.5568, \quad -2.7431, \quad -1.9942, \quad -2.1396\} \quad \text{radians} \tag{42}$$

The communication network topology is given by:

$$\mathcal{G}_k = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{43}$$

Using Algorithm 2, we see that all the agents converge to the mean of the initial rotation matrices (see Eq. 34) given by:

$$\prod_{\ell \in \mathcal{V}} (\boldsymbol{R}_k^\ell)^{\frac{1}{N}} = \{\boldsymbol{e}_k^\star, \quad -2.2567 \,\text{radians}\}\,. \tag{44}$$

B. General Case

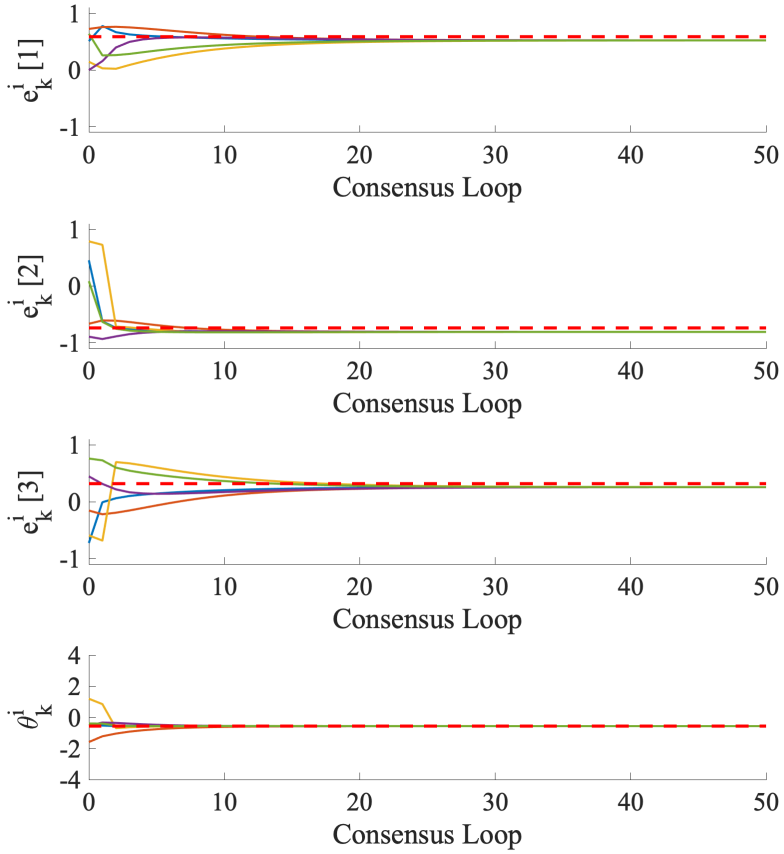


Figure 7 Consensus of 5 agents in the general case

In the general case, where the initial rotation matrices are not commutative, it follows from Theorem 3 that the agents will converge using the consensus algorithm, but they need not converge to the mean of the initial rotation matrices. Let us start with the following initial rotational matrices:

$$\boldsymbol{e}_k^1 = [0.5144 \quad 0.4556 \quad -0.7265], \qquad \theta_k^1 = -0.3986 \quad \text{radians} \tag{45}$$

$$\boldsymbol{e}_k^2 = [0.7295 \quad -0.6661 \quad -0.1556], \qquad \theta_k^2 = -1.5872 \quad \text{radians} \tag{46}$$

$$\boldsymbol{e}_k^3 = [0.1487 \quad 0.7909 \quad -0.5936], \qquad \theta_k^3 = 1.1962 \quad \text{radians} \tag{47}$$

$$\boldsymbol{e}_k^4 = [0.0022 \quad -0.8951 \quad 0.4458], \qquad \theta_k^4 = -0.5324 \quad \text{radians} \tag{48}$$

$$\boldsymbol{e}_k^5 = [0.6439 \quad 0.0862 \quad 0.7602], \qquad \theta_k^5 = -0.3939 \quad \text{radians} \tag{49}$$

The communication network topology is given by:

$$\mathcal{G}_k = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \tag{50}$$

Using Algorithm 2, we see that all the agents converge to the following rotation matrix:

$$\lim_{j\to\infty} \boldsymbol{C}^i_{k,j} = \{[0.5268 \quad -0.8104 \quad 0.2564], \quad -0.5639\,\text{radians}\} \quad \forall i \in \mathcal{V}, \tag{51}$$

which is not equal to the mean of the initial rotation matrices (see Eq. 34) given by:

$$\prod_{\ell\in\mathcal{V}} (\boldsymbol{R}^\ell_k)^{\frac{1}{N}} = \{[0.5922 \quad -0.7403 \quad 0.3183], \quad -2.2567\,\text{radians}\}\,. \tag{52}$$

VI. Conclusions

In this paper, we presented novel algorithms for consensus in $SO(3)$ manifold and numerical results to demonstrate their convergence properties. Future work will focus on the consensus of the combined pose estimate (position and attitude) in the Lie group $SE(3)$. Next, we will extend these algorithms to probability distributions in $SO(3)$ and $SE(3)$ manifolds, similar to consensus of probability distributions in Euclidean space.[8]

Acknowledgements

Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] Rahimi, N., Talebi, S., Deole, A., Mesbahi, M., Bandyopadhyay, S., and Rahmani, A., "Robust Controller Synthesis for Vision-based Spacecraft Guidance and Control," *AIAA SciTech Guidance, Navigation, and Control (GNC)*, 2022.

[2] Becktor, J., Seto, W., Deole, A., Bandyopadhyay, S., Rahimi, N., Talebi, S., Mesbahi, M., and Rahmani, A., "Robust Vision-based Multi-spacecraft Guidance Navigation Control using CNN-based Pose Estimation," *IEEE Aerospace Conference*, 2022.

[3] Olfati-Saber, R., and Murray, R., "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. Autom. Control*, Vol. 49, No. 9, 2004, pp. 1520 – 1533.

[4] Olshevsky, A., and Tsitsiklis, J. N., "Convergence Speed in Distributed Consensus and Averaging," *SIAM Journal on Control and Optimization*, Vol. 48, No. 1, 2009, pp. 33–55.

[5] Ren, W., and Beard, R. W., "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Trans. Autom. Control*, Vol. 50, 2005, pp. 655 – 661.

[6] Horn, R. A., and Johnson, C. R., *Matrix Analysis*, Cambridge University Press, Cambridge, England, 1985.

[7] Barfoot, T. D., *State estimation for robotics*, Cambridge University Press, 2017.

[8] Bandyopadhyay, S., and Chung, S.-J., "Distributed Bayesian Filtering using Logarithmic Opinion Pool for Dynamic Sensor Networks," *Automatica*, 2018. Accepted.