Distributed MPC for co-ordinated RPO between multiple spacecrafts

*Note: Sub-titles are not captured in Xplore and should not be used

Adarsh Rajguru
Astronautical Engineering
University of Southern California
Los Angeles, United States
adarsh.rajguru@outlook.com

Helena Teixeira-Dasilva
Mechanical Engineering
University of Southern California
Los Angeles, United States
hcteixei@usc.edu

Derek Chibuzor
Mechanical Engineering
University of Southern California
Los Angeles, United States
chibuzor@usc.edu

***Abstract*—Rendezvous and Proximity Operations (RPO) are cornerstone capabilities that enable critical on-orbit activities such as refueling, servicing, inspection, repair, and In-Space Assembly and Manufacturing (ISAM). As commercial participation in these domains expands, thousands of future missions will depend on safe, autonomous, and scalable RPO operations. However, the prevailing paradigm remains asymmetric: a single servicer spacecraft assumes full Guidance, Navigation, and Control (GNC) responsibility while the client or target remains passive. This architecture, historically sufficient for government-led missions, increasingly constrains safety and efficiency as rendezvous attempts multiply across heterogeneous commercial spacecraft and operators. The lack of a standardized, cooperative framework further amplifies operational risk, with non-successful rendezvous posing growing conjunction hazards in Earth orbit. In this paper we evaluate the application of Model Predictive Control (MPC) in two different forms to see the benefits of doing robust optimal control at the proximity phases of RPO when multiple spacecraft are trying to approach a single client. This research lays the foundation of a fundamental new approach in cooperative Rendezvous and Proximity Operations (CoRPO), in which two or more participating spacecraft actively, share state information and actuation responsibilities throughout the RPO sequence. The hypothesis is that bidirectional coordination and shared optimal control can reduce fuel usage while increasing safety and mission resilience.**

***Index Terms*—Rendezvous Proximity Operations, Model Predictive Control, Optimal Control, Multi-agent Control**

I. Introduction

A. Background and context

Rendezvous and Proximity Operations (RPO) constitute the core enabling capability for On-Orbit Servicing (OOS) or In-Space Assembly, and Manufacturing (OOS / ISAM). Whether supporting refueling, orbital relocation, or the modular construction of large space infrastructures, effective RPO is the operational backbone of this rapidly expanding commercial domain. However, the prevailing paradigm remains inherently asymmetric: a single *servicer* spacecraft assumes full responsibility for guidance, navigation, and control (GNC), while the *client* or target vehicle remains largely passive. Coordination and conflict avoidance are typically managed through pre-planned, ground-based procedures [1]. Although this approach has been proven and reliable in traditional government-led missions, it increasingly limits safety, responsiveness, and scalability as RPO activities proliferate across multiple heterogeneous commercial spacecraft and operators [2].

II. Modern day challenges for RPO missions

Traditional servicer-client RPO places the burden of sensing and control on one spacecraft, forcing conservative safety buffers and suboptimal control approaches in the presence of sensing gaps, keep-out constraints, plume impingement, and uncertainty in non-cooperative geometries. Moreover, many current systems separate the docking mechanism from relative navigation sensors, which complicates fault responses and increases the integration burden [2], [3]. The current state-of-the-art approaches in conducting RPO are outdated. Below are some additional problems that make the current approach less resilient, reliable, and efficient.

A. Perception brittleness in the real space scene

1) Vision / LiDAR struggles with harsh lighting, specular glint, deep shadow, stray light, and sensor occlusions from plumes or appendages as shown in Figure 1.
2) Pose estimation on non-cooperative geometry is still data-hungry and brittle to domain shift.
3) Small perception errors can escalate into safety margins and fuel, necessitating conservative operations.
4) Exchange of state information between the servicer and client during all phases of RPO can provide closure on localization in the presence of unreliable sensor data. This reduces significant errors and aids the controller in minimizing the impact on fuel and energy consumption.

B. Planning under tight, moving constraints is hard to do

1) True 6-DoF planning must respect keep-out zones, plume-impingement limits, sensor FoV, approach corridors, and time-varying lighting.

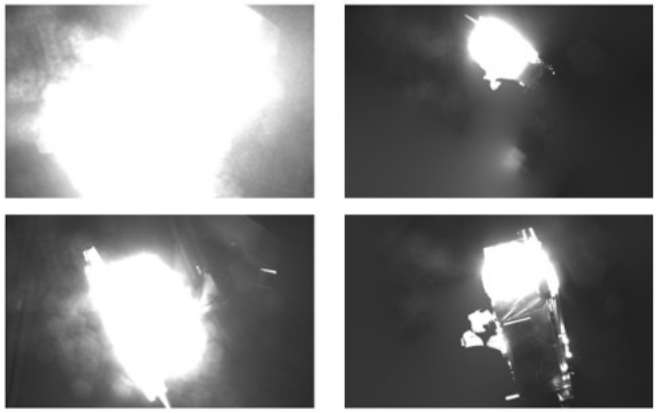

Fig. 1. Samples rejected from the sunlamp domain due to extreme surface glow and camera overexposure (left) and failed post-processing (right) [4].

2) Most flight-proven stacks default to scripted corridors; general, real-time optimal planning with guarantees is rare. Modern planners still struggle with plume / FoV / Keep-Out Zone (KOZ) constraints, as well as non-convexity. Even International Space Station's visiting vehicles' tooling enforces fixed corridors.
3) *So what*? We over-constrain operations, pay in ΔV / time, and still can't prove safety in edge cases. Plume impingement and KOZ constraints remain first-class constraints and drive suboptimal, conservative paths.

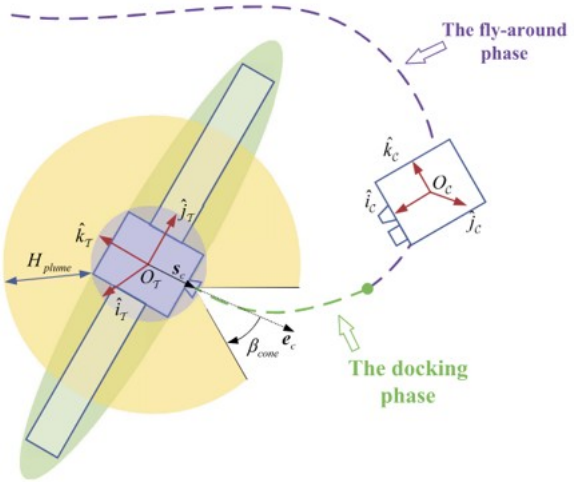

Fig. 2. Representation of proximity process and path constraints [5].

4) An optimal control strategy, that can leverage exchange of state information, can handle multiple dynamic constraints and uncertainties among multiple agents, thus enabling the real-time resolution of the optimal path.

C. We lack flight-grade, formal safety guarantees

1) Tools like reachability, Control Barrier Functions (CBFs), and Run Time Assurance (RTA) are maturing, but rarely fly for RPO yet, and almost never with two maneuvering vehicles.
2) *So what*? Mission assurance default to "test more / fly simpler," limiting autonomy and resilience. Recent research shows promising CBF RTA formulations, but these are largely at low TRL for spaceflight.

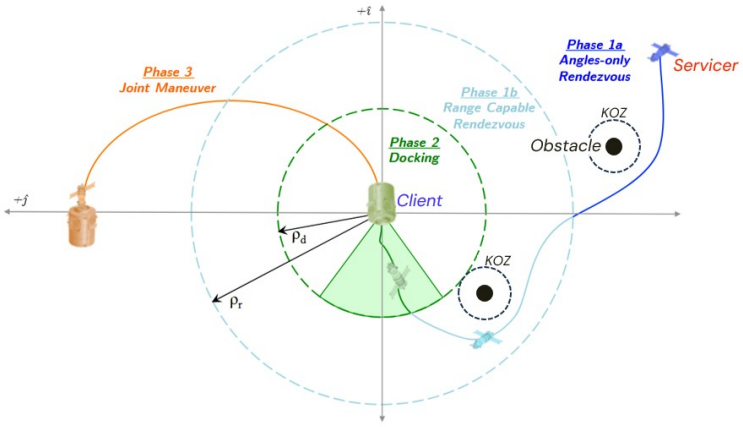

Fig. 3. Illustration of an obstacle that is deemed as a KOZ and CBFs embedded in navigation control strategy can avoid such KOZ [6].

3) CBFs are a mathematical tool from control theory. The new RPO approach employs an optimal control strategy that can incorporate CBFs to ensure the system state remains within a safe set (e.g., "spacecraft must never enter a collision zone", see figure 3). Works by modifying the control input: you give the controller a nominal command (say, from an MPC or learned policy), and the CBF ensures the final command won't violate the safety inequality. *Analogy*: like a "mathematical guardrail" that enforces collision-avoidance no matter what.
4) RTA is an architecture pattern for certifiable autonomy. The New RPO approach can run the advanced controller (MPC) in parallel with a simple, guaranteed-safe backup controller. At each timestep, the RTA monitor checks: "Will the advanced controller keep us safe?" *If yes*: advanced controller's action is executed. *If no*: switch immediately to the safe backup. This guarantees safety while still allowing high-performance autonomy.

III. Significance of Study - Motivation

There is a growing demand (thousands of proposed RPO missions) for low-cost and resilient in-orbit assembly capabilities for government and commercial spacecraft of all sizes. Some ambitious astrophysics mission proposals include assembling a giant telescope in orbit using CubeSats as shown in 4. Hence, the new way of performing RPO will:

- Unlock these new commercial opportunities in LEO
- Significantly reduce the fuel consumption and improve the payload capacity of ISAM for assembling very large monolithic spacecraft in orbit.
- Increase resiliency to normalize satellite repairs and maintenance in the future with the help of autonomy.
- Reduce dependency on ground resources.

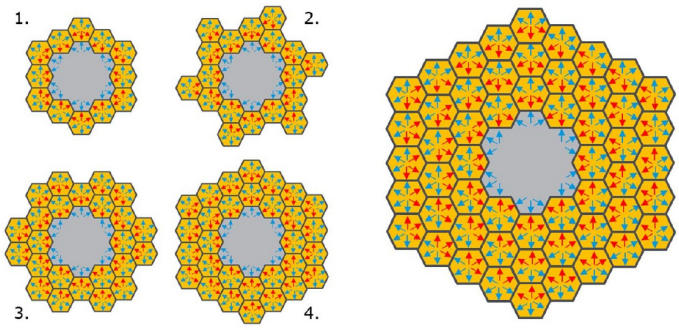

Fig. 4. Assembly sequence of mirrors using segments of small hexagonal satellites.

Hence it is relevant to investigates a new paradigm - **Cooperative RPO (CoRPO)** that leverages robust optimal control strategy that can be computationally efficient to implement. In CoRPO all spacecraft actively communicate their state vectors, when capable, share mobility responsibilities. The central premise is that real-time state exchange (position, attitude, uncertainties, and mode / status) and cooperative guidance can reduce propellant consumption, time-to-dock, and collision risk while increasing autonomy and resilience.

IV. Problem formulation

The servicer approaches the target client through different phases of RPO. The terminology used for the phases of Rendezvous and Proximity Operations (RPO) follows the nomenclature and definitions established by the Consortium for Execution of Rendezvous and Servicing Operations (CONFERS) [8]. Figure 21 illustrates the complete sequence of RPO phases and presents a generalized schema derived from the methodologies described by Fehse [10] and Xie et al. [11], which are based on the Automated Transfer Vehicle (ATV) and Shenzhou missions, respectively [12]. For the purpose of this research work the phases of RPO are defined in Appendix B.

A. Problem statement and hypothesis

In Phases 1(b), and 2 of RPO the complexity compounds due to multiple reasons stated in section 2 earlier and that impacts the fuel consumption needed for maneuvering in those phases, lowering performance and impacting spacecraft payload mass and functionality. To address the above problem, we hypothesize that *A cooperative optimal control strategy based on a Model Predictive Control (MPC) architecture can achieve more than 30% fuel savings in Phases 1(a), 1(b), and 2 of RPO compared to a conventional single-servicer baseline*.

B. Use cases for MPC

1) RPO has complex, multi-dimensional constraints:
   - Spacecraft must respect FoV limits, KoZ boundaries, plume impingement cones, docking approach corridors, and relative velocity limits.
   - These constraints are often non-convex.
   - Classical controllers (PD / LQR) cannot enforce constraints explicitly.
   - MPC incorporates constraints directly into the optimization, guaranteeing feasible and safe trajectories.
2) Dynamics change across RPO phases
   - Far-field motion follow HCW / TH linear models
   - Close proximity motion is 6-DoF, thrust dominated, and highly non-linear
   - No single controller can handle both regimes.
   - MPC naturally supports multi-phase or hybrid switching, allowing different models in Phases 1(a) and 1(b) of RPO.
3) Co-operative RPO requires coordinated decision making
   - Multiple spacecraft must share state information and jointly decide how to minimize fuel and risk.
   - Independent controllers typically fight each other or behave conservatively.
   - MPC enables distributed or cooperative optimization, where each spacecraft reasons about the future trajectory of the other.
4) Fuel efficiency
   - Traditional bang-bang or rule-based controllers expend expensive ΔV to guarantee safety.
   - Smooth, predictive, fuel-optimal maneuvers.
   - MPC optimizes fuel use directly, minimizing cumulative thrust effort over a future horizon.
5) Uncertainty must be handled explicitly
   - Sensors errors (camera / LiDAR), actuation noise, and communication delays affect safety margins.
   - Classical controllers struggle with uncertainty without becoming overly conservative
   - Robust and stochastic MPC can tighten constraints automatically based on uncertainty, enabling safe but less conservative maneuvering.
6) MPC "looks ahead" – classical controllers don't
   - PD / LQR reacts only to instantaneous errors.
   - RPO requires anticipating future conflict (e.g., a predicted KoZ intrusion 10 seconds ahead).
   - MPC predicts future trajectories, avoiding constraint violations before they happen.

C. Architectures of MPC

There are two architectures of MPC that we implemented to evaluate the boost in performance of performing RPO in the proximity and gapple phases. These architectures are: (1) Distributed MPC and (2) Decentralized MPC. Distributed MPC uses a single centralized solver that solves the global optimization problem. It may yield better performance at the cost of computational complexity and latency. In decentralized MPC agents solve local control problems and share solutions. It is less computationally intensive but may yield suboptimal performance. Figure 5 shows the two difference MPC architectures that we used.

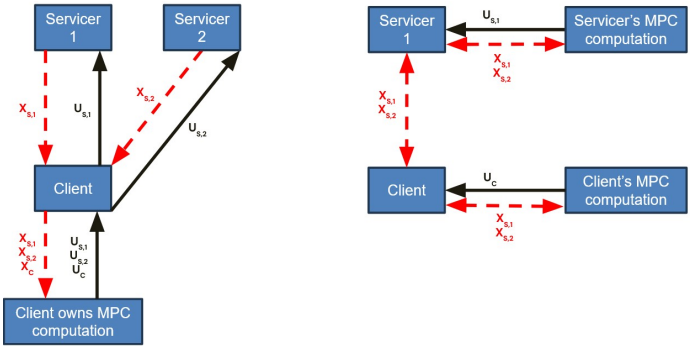

Fig. 5. Layout of MPC architectures

V. Method and Experiment

A. Dynamics: 3 DoF Double Integrator

Often spacecraft dynamics can be modeled with 6 degrees of freedom as shown in Figure 6. Where the spacecraft motion can be translational in X, Y and Z, as well as rotational in roll, pitch, and yaw. The control inputs can be modeled as force and torque in all directions of motion.

For our implementation, we chose to focus solely on the position of the spacecraft and not model the orientation and rotational dynamics. This allowed us to run more experiments efficiently, and still estimate rendezvous with close spatial proximity. We modeled the spacecraft as a 3 degree of freedom (3 DoF) point mass. This includes a state vector with X, Y, Z positions. And the control inputs will be forces in X, Y and Z.

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}, \quad v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

$$F = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

In the dynamics, our state vector, $\mathbf{x}$, and control vector $\mathbf{u}$ include:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix} \in \mathbb{R}^6, \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{F} \end{bmatrix} \in \mathbb{R}^3$$

Continuous-Time Model: The dynamics model we implemented is a 3 DoF double-integrator. The double-integrator simplification relies on the assumption that no external forces beyond the control input affect the dynamics. Additionally, we assume a constant mass to leverage a linear time-invariant model. It is extrapolated into the discrete time model to create the simulations, in which a zero-order hold is used to estimate discrete dynamics. We also assume actuators are ideal with no delay or noise. Our dynamics are only defined in translational directions and rotational dynamics is ignored. The continuous time model for the double integrator is shown below.

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \quad \dot{\boldsymbol{v}} = \frac{1}{m}\boldsymbol{F}$$

$$\dot{x}(t) = A\,x(t) + B\,u(t),$$

with

$$A = \begin{bmatrix} 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix}, \qquad B = \begin{bmatrix} 0_{3\times3} \\ \frac{1}{m} I_3 \end{bmatrix},$$

where m is the spacecraft mass and I_3 denotes the 3×3 identity matrix.

Discrete-Time Model: The discrete–time dynamics are written as:

$$x_{k+1} = A_d\,x_k + B_d\,u_k,$$

where

$$A_d = \begin{bmatrix} I_{3\times3} & dt\,I_3 \\ 0 & I_3 \end{bmatrix}, \qquad B_d = \begin{bmatrix} \frac{1}{2}dt^2\,\frac{1}{m} I_3 \\ dt\,\frac{1}{m} I_3 \end{bmatrix}.$$

Equivalently, in component form:

$$p_{k+1} = p_k + dt\,v_k + \tfrac{1}{2}dt^2\,\frac{1}{m}F_k, \tag{1}$$

$$v_{k+1} = v_k + dt\,\frac{1}{m}F_k. \tag{2}$$

Where dt represents the time step, and p_k, v_k and F_k represent the 3 dimensional position, velocity, and force respectively at time k.

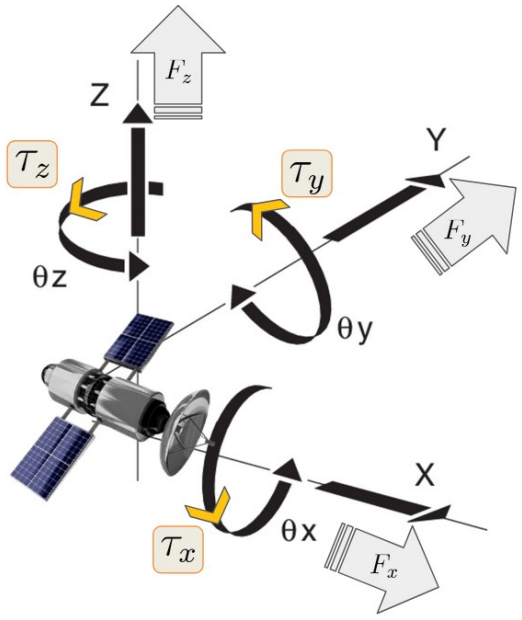

Fig. 6. Dynamics of the spacecraft model used for the MPC implementation.

B. 3 cases of RPO and initial conditions for simulations

We developed three cases of RPO to study the benefits of MPC implementation. They are described as shown below:

- **Case 1**: Agent 0 (Client) is arriving to a specific end destination for operational functionality. This is Agent 0's goal. Agent 1, 2, 3,... (servicers) are tasked to approach to Agent 0. This is the servicer's goal. They both have to adhere to constraints and keep out zones.

- **Case 2**: Agent 0 (Client) and Agent 1, 2, 3,...(servicers) are arriving to a specific end destination for operational functionality. They both have to adhere to constraints, keep out zones and avoid collision between each other.
- **Case 3**: Agent 0 (client) and Agent 1, 2, 3,...(servicers) are both trying to get to each other for soft capture and then continue maintaining desired proximity.

For each of the three aforementioned cases, the following parameters were used in simulation. It should be noted that these parameters were chosen for simplicity, not necessarily for realism with respect to the RPO use case:

- **Discretization**
  - Time step, $\Delta t = \{0.5, 0.1, 0.5\}$ s.
  - Prediction horizon, $N = \{25, 50, 25\}$ steps.
  - Total simulation time, $T = \{50, 125, 50\}$ steps.
- **Inertial parameters**
  - System mass, $m = 50$ kg.
- **Initial and final conditions**

TABLE I
INITIAL AND FINAL CONDITIONS FOR EACH CASE AND AGENT

Case	Agent	Initial State x_0,	Final State x_f
1	0	$[0, 0, 0, 0, 0, 0]^T$	$[50, 50, 50, 0, 0, 0]^T$
	1	$[100, 100, 100, 0, 0, 0]^T$	Go to Agent 0
2	0	$[0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0]^T$
	1	$[100, 100, 100, 0, 0, 0]^T$	Go to Agent 0
	2	$[50, 50, 100, 0, 0, 0]^T$	Go to Agent 0
3	0	$[0, 0, 0, 0, 0, 0]^T$	Go to Agent 1
	1	$[100, 100, 100, 0, 0, 0]^T$	Go to Agent 0

- **Collision constraints**
  - Minimum separation distance, $d_{\min} = 0.5\,\text{m}$.
  - Number of obstacles (except case 2), $n_{\text{obs}} = 3$.
  - Obstacle states (position and radius):

$$p_{\text{obs}} = \begin{bmatrix} 20 & 20 & 20 \\ 80 & 80 & 80 \\ 60 & 60 & 20 \end{bmatrix} \in \mathbb{R}^{n_{\text{obs}} \times 3},$$

$$r_{\text{obs}} = \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} \in \mathbb{R}^{n_{\text{obs}}}.$$

- **Decentralized objective, per agent**

$$J = \sum_{k=0}^{N-1} \left(x_k^\top Q x_k + u_k^\top R u_k \right) + x_N^\top H x_N,$$

$$Q = 10 \times \mathbf{I}_{n_x}, \quad R = 0.01 \times \mathbf{I}_{n_u}, \quad H = 100 \times \mathbf{I}_{n_x},$$

C. Decentralized computation architecture

There are many architectural formulations of decentralized MPC. But, the two main formulations we used for this study are Jacobi method and Gauss-Seidel method.

- **Jacobi Method**: All agents update their controls in parallel using previous step's neighbor predictions. In the **Jacobi** method, all agents begin from an initial state and compute their control actions in parallel using the information available at that time. After executing these controls, each agent transitions to its next state. Once all agents have completed the step, they exchange updated state information with one another. This cycle repeats over the full computation horizon. Figure 7 shows a visual layout of Jacobi method.

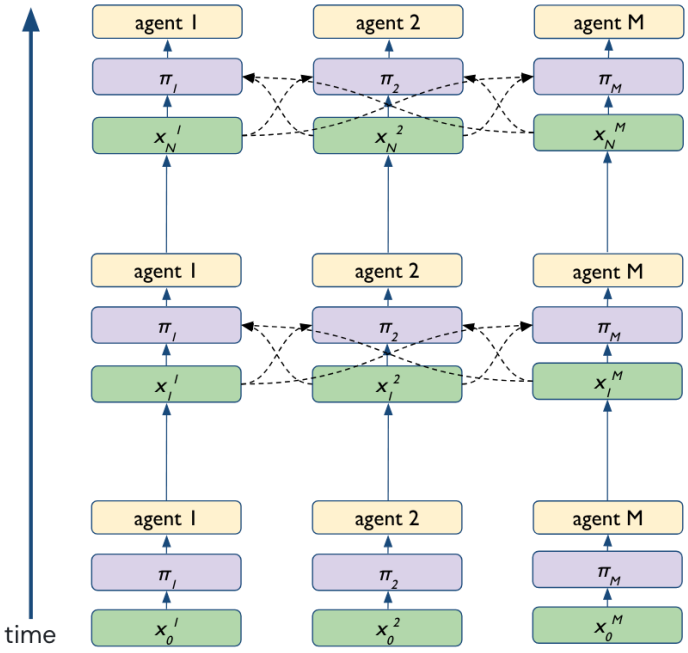

Fig. 7. Jacobi method for decentralized MPC

Algorithm 1 shows how we setup Jacobi's method for decentralized MPC architecture. A key limitation of Jacobi is that agents only receive others' updated states *after* completing their own moves, which can be problematic for collision avoidance between each other at very close proximity. But this issue is handled better by the Gauss–Seidel method discussed in the next subsection.

Algorithm 1 Jacobi's method of decentralized MPC

Require: Number of agents M, horizon N, initial states $x_0^{(m)}$, predictions $X^{(m)}$

for $k = 0$ to $T - 1$ **do**
  for all agents $m = 1, \ldots, M$ **do**
    $X_{\text{others}}^{(m)} \leftarrow \{X^{(j)}\}_{j \neq m}$
    Solve local OCP:
    $(X_{\text{opt}}^{(m)}, U_{\text{opt}}^{(m)}) = \arg\min J^{(m)}(X^{(m)}, U^{(m)}; X_{\text{others}}^{(m)})$
    $\tilde{X}^{(m)} \leftarrow \text{shift}(X_{\text{opt}}^{(m)})$
  end for
  for $m = 1$ to M **do**
    $X^{(m)} \leftarrow \tilde{X}^{(m)}$
    $x_{k+1}^{(m)} \leftarrow f(x_k^{(m)}, U_{\text{opt}}^{(m)}(:, 0))$
  end for
end for=0

Gauss-Seidel Method: Agents update sequentially, with each agent using the most recently updated information from agents that have already executed within the same iteration. After Agent 1 advances to its next state, its updated state is communicated to the remaining agents. Agent 2 then leverages Agent 1's updated state to compute and execute its own step, after which its state is shared in turn. This process continues until all agents have advanced. Algorithm 2 summarizes the Gauss–Seidel procedure for decentralized MPC, and Fig. 8 provides a visual schematic of the update sequence. Compared to the Jacobi method, the Gauss–Seidel update order makes fresher state information available during the iteration, which can improve collision-avoidance performance.

Algorithm 2 Gauss-Seidel's method in decentralized MPC

Require: Number of agents M, horizon N, initial states $x_0^{(m)}$, predictions $X^{(m)}$

for $k = 0$ to $T-1$ **do**
  for $m = 1$ to M **do**
    $X_{\text{others}}^{(m)} \leftarrow \{\tilde{X}^{(j)}\}_{j<m} \cup \{X^{(j)}\}_{j>m}$
    Solve local OCP:
    $$(X_{\text{opt}}^{(m)}, U_{\text{opt}}^{(m)}) = \arg\min J^{(m)}(X^{(m)}, U^{(m)}; X_{\text{others}}^{(m)})$$
    $\tilde{X}^{(m)} \leftarrow \text{shift}(X_{\text{opt}}^{(m)})$
    $X^{(m)} \leftarrow \tilde{X}^{(m)}$
    $x_{k+1}^{(m)} \leftarrow f(x_k^{(m)}, U_{\text{opt}}^{(m)}(:,0))$
  end for
end for
=0

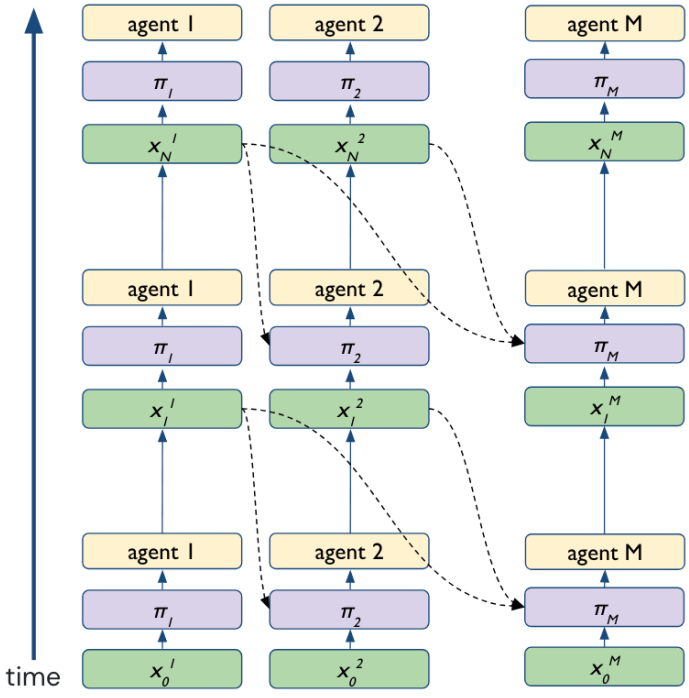

Fig. 8. Gauss-Seidel method for decentralized MPC

Decentralized MPC (a proposed use case): Many current systems employ a straight-line path within a docking cone, with a prescribed range versus closing-velocity profile (e.g., 10 cm/s at 100 m, tapering to a few cm/s at contact). R-bar/V-bar glideslope algorithms from the proximity phase are often reused for the final tens of meters, with additional docking-cone (see Fig. 9) constraints [12]. In a Gauss-Seidel based decentralized MPC scheme (discussed next), each spacecraft sequentially updates its control using the most recent prediction of the other. This is well suited to provide step-by-step cooperative guidance. Such an approach can maintain the target docking port within a required line-of-sight cone while generating relative position commands to remain inside this cone in a fuel-efficient and constraint-aware manner.

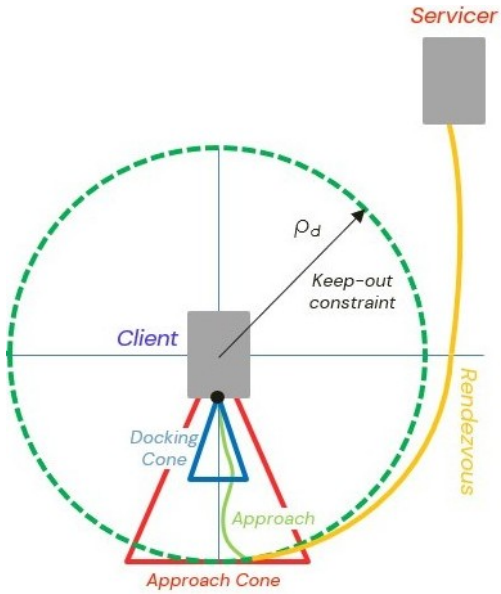

Fig. 9. Docking cone in the final phase of RPO

D. Distributed computation architecture

In this architecture, each agent transmits its state to a centralized controller. The controller may be hosted onboard the client vehicle or on a separate supervisory agent that is not directly involved with the group. The individual agent states and control inputs over a set horizon are then stacked into a single, augmented state vector. The MPC optimization is performed by this single controller, which simultaneously accounts for the dynamics of all agents, enforces inter-agent keep-out zones and obstacle constraints, and satisfies mission objectives such as specified terminal conditions or formation-keeping requirements. The optimized polices are then sent back to each agent, which includes the control input for the next step. This process gets repeated as the agents move through the timeline. Figure 10 shows the layout of this architecture, which is also detailed in algorithm 3.

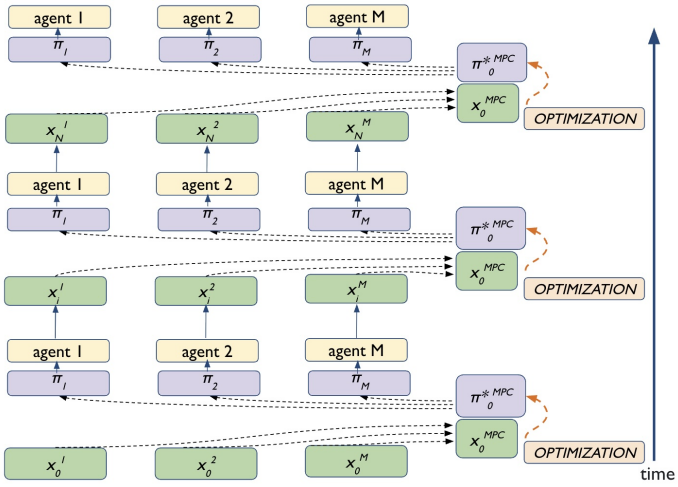

Fig. 10. Distributed computation architecture

Algorithm 3 Distributed MPC Solver for M Agents

Require: Cost function J,
Agent state $\{x^{(m)} = [x_0..x_N]^{(m)}\}_{m=1}^{M}$,
Agent input $\{u^{(m)} = [u_0..u_N]^{(m)}\}_{m=1}^{M}$,
Agent dynamics $\{A_m, B_m\}_{m=1}^{M}$

Stacked state and input:
$x \leftarrow [\, x^{(1)};\, x^{(2)};\, \ldots;\, x^{(M)}\,]$
$u \leftarrow [\, u^{(1)};\, u^{(2)};\, \ldots;\, u^{(M)}\,]$
Solve OCP:

$$(X_{\text{opt}}, U_{\text{opt}}) = \arg\min_{[x,u]^T} J$$

for $m = 1$ to M **do**
  $x_{opt}^{(m)} = X_{opt}[m]$
  $u_{opt}^{(m)} = U_{opt}[m]$
  $x_{k+1}^{(im)} \leftarrow A_m x_k^{(m)} + B_m u_{opt}^{(m)}(:,0))$
end for

Ensure: =0

The constraint and cost function construction are shown in algorithm 4. The cost function in the optimization problem encompasses pairwise distances, constraints encompassing dynamics, input bounds, and obstacle avoidance for each agent.

E. Simulation setup

In subsection V-B, we introduced the three RPO scenarios used in our simulation study. For each case, we held the initial conditions and constraint sets fixed and evaluated three MPC implementations: (1) decentralized MPC using the Jacobi method, (2) decentralized MPC using the Gauss–Seidel method, and (3) distributed MPC. Across all architectures, the controller's objective is to minimize control input, measured as the cumulative thrust commanded over the horizon. We adopt total thrust as the primary performance metric because propellant consumption is directly proportional to it (for a fixed thruster specific impulse). All simulation parameters, including spacecraft mass and discretization time step, are specified in subsection V-B.

Algorithm 4 Distributed MPC Construction for M Agents

Require: horizon N, Agent state $\{x^{(m)} = [x_0..x_N]^{(m)}\}_{m=1}^{M}$, Agent input $\{u^{(m)} = [u_0..u_N]^{(m)}\}_{m=1}^{M}$, Agent dynamics $\{A_m, B_m\}_{m=1}^{M}$, cost matrices Q, R, H, obstacle set $\mathcal{O}$

Stacked state and input:
$x \leftarrow [\, x^{(1)};\, x^{(2)};\, \ldots;\, x^{(M)}\,]$
$u \leftarrow [\, u^{(1)};\, u^{(2)};\, \ldots;\, u^{(M)}\,]$
Stacked dynamics:
$A_{\text{joint}} \leftarrow \text{blkdiag}(A_1, A_2, \ldots, A_M)$
$B_{\text{joint}} \leftarrow \text{blkdiag}(B_1, B_2, \ldots, B_M)$
for $k = 0$ to $N-1$ **do**
  $x_{k+1} \leftarrow A_{\text{joint}} x_k + B_{\text{joint}} u_k$
  Define pairwise relative distance:
  for $i = 1$ to $M-1$ **do**
    for $j = i+1$ to M **do**
      $r_{ij,k} \leftarrow x_k^{(j)} - x_k^{(i)}$
    end for
  end for
  Add input bounds:
  $u_{\min} \le u_k \le u_{\max}$
  Add obstacle-avoidance constraints:
  for $i = 1$ to M **do**
    for all obstacles $(c, r) \in \mathcal{O}$ **do**
      $\|p_{i,k} - c\|^2 \ge (r + \delta_{safe})^2$
    end for
  end for
end for
Construct finite-horizon cost:

$$J = \sum_{k=0}^{N-1} \left(\sum_{i<j} r_{ij,k}^{\top} Q r_{ij,k} + u_k^{\top} R u_k \right) + \sum_{i<j} r_{ij,N}^{\top} H r_{ij,N}$$

Ensure: =0

VI. Experiments - Simulation Results

A. RPO case 1 simulation results

1) Decentralized MPC results for case 1: Figures 11 shows the trajectory for the Case 1 results of decentralized MPC using both the Gauss-Seidel and Jacobi methods. For both methods the result was identical. In this scenario, the client (Agent 0) is commanded to translate to a specified point in space, while the servicer (Agent 1) is tasked with rendezvousing with Agent 0. Agent 0 is shown in blue and agent 1 in orange. The total force needed was 14.8 kN.

2) Distributed MPC results for case 1: Figure 12 presents the Case 1 results for distributed MPC architecture. Agent 0's trajectory is shown in orange has a goal to arrive at a specific location in space designated by the star. Agent 1's goal, trajectory shown in blue, is to approach agent 0. The red circles represent the keep out zones. Figure 13 shows the plotted relative position between the two agents for case 1. The

total control effort used in Case 3 with distributed MPC was 8.2 kN. A complete summary and breakdown can be found in the appendix.

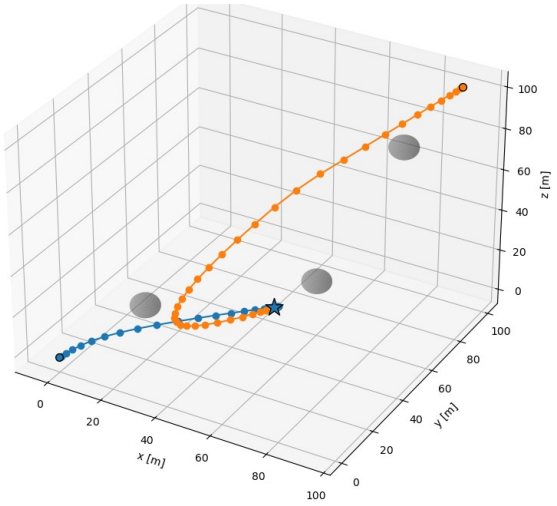

Fig. 11. Decentralized MPC results using Gauss-Seidel method for Case 1

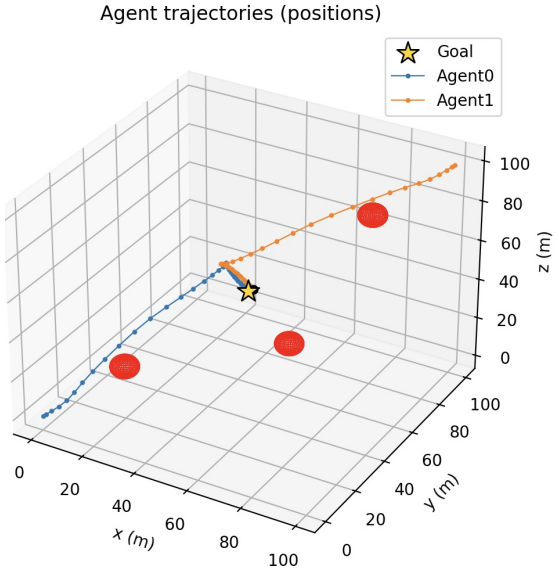

Fig. 12. Distributed MPC results for Case 1

B. RPO case 2 simulation results

1) Decentralized MPC results for case 2: Figure 14 shows the trajectory for the Case 2 results of decentralized MPC using both the Gauss-Seidel and Jacobi methods. For both methods the result was identical. In this scenario, the client (Agent 0) remains at a fixed location in the agents' relative (inertial) reference frame. Here, "fixed" refers to the local frame used to describe inter-agent motion: because the vehicles are in close proximity and share nearly the same orbit, they appear to co-orbit together in the geocentric frame. Agents 1 and 2 are tasked with rendezvousing with Agent 0. Agent 1 is shown in orange and Agent 2 is shown in green. No constraints were applied in this simulation, and the total thrust expenditure was for the Gauss-Seidel and Jacobi methods was 16.4 kN.

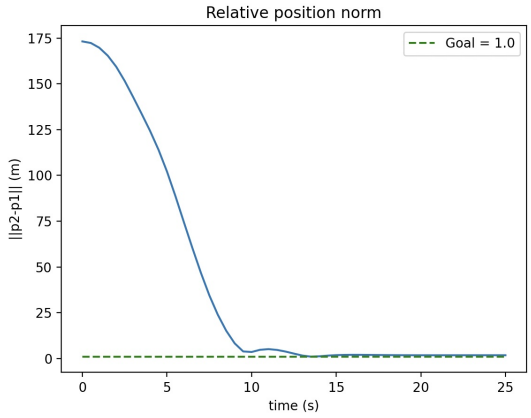

Fig. 13. Relative position between two agents using distributed MPC architecture for case 1.

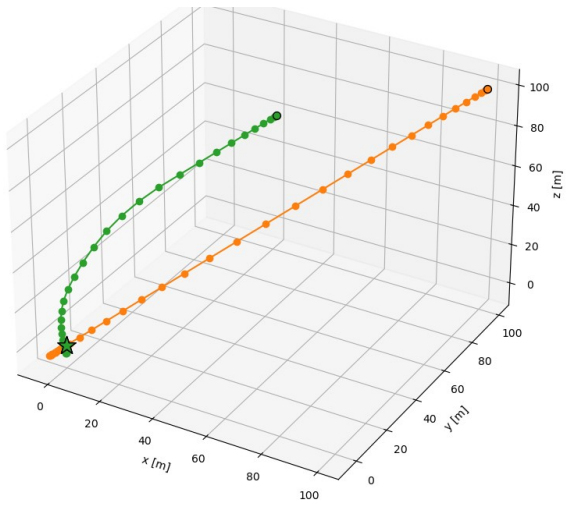

Fig. 14. Identical Decentralized MPC results using Gauss-Seidel and Jacobi methods for Case 2.

2) Distributed MPC results for case 2: Figure 15 presents the Case 2 results for the distributed MPC architecture. In this scenario, two servicer agents (Agents 1 and 2) are tasked with approaching the client (Agent 0) and achieving a standoff distance of 1 meter. Fig. 16 shows the relative distances of Agents 1 and 2 with respect to Agent 0, (leader), and each other. The green dotted line denotes a 1 meter separation threshold. The total control effort used in Case 3 with distributed MPC was 9.0 kN.

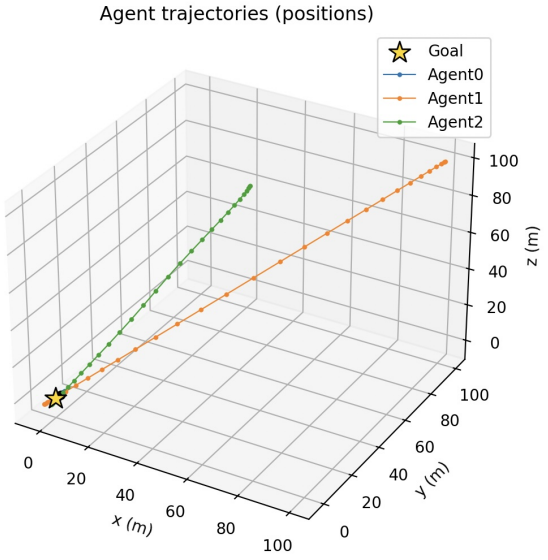

Fig. 15. Distributed MPC results for Case 2.

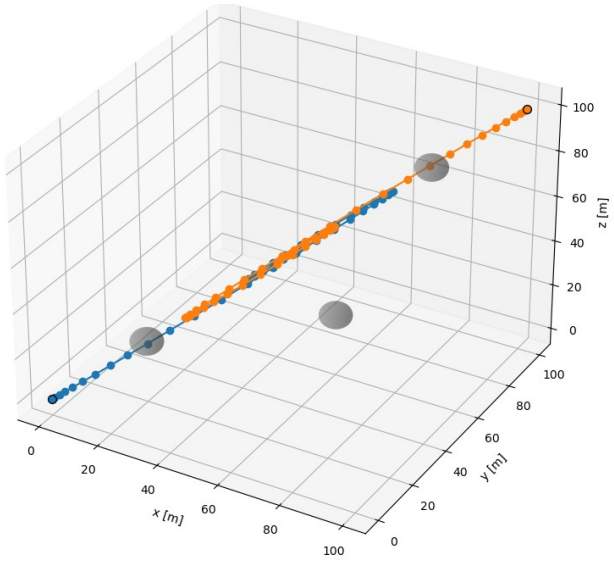

Fig. 17. Identical decentralized MPC results using Gauss-Seidel and Jacobi methods for Case 3.

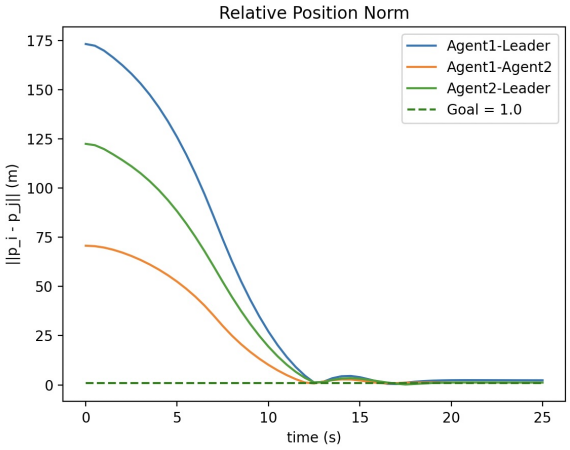

Fig. 16. Relative position of two servicer agents approaching the client using distributed MPC architecture (Case 2).

C. RPO case 3 simulation results

1) Decentralized MPC results for case 3: Figure 17 shows the trajectory for the Case 3 results of decentralized MPC using both the Gauss-Seidel and Jacobi methods. For both methods the result was identical. Agent 0's trajectory is shown in blue and Agent 1's in oprange. In this scenario, both agents attempt to rendezvous with one another. The total thrust expenditure is identical for the two methods; however, significant overshoot is observed before the agents achieve mutual rendezvous. The vehicles repeatedly pass one another due to these overshoots, and only converge after multiple passes. Overall, the closed-loop response exhibits oscillatory convergence, which is a known drawback of the decentralized MPC formulation in this setting. Constraints were enforced in this case. Notably, this behavior is mitigated under the distributed MPC architecture.

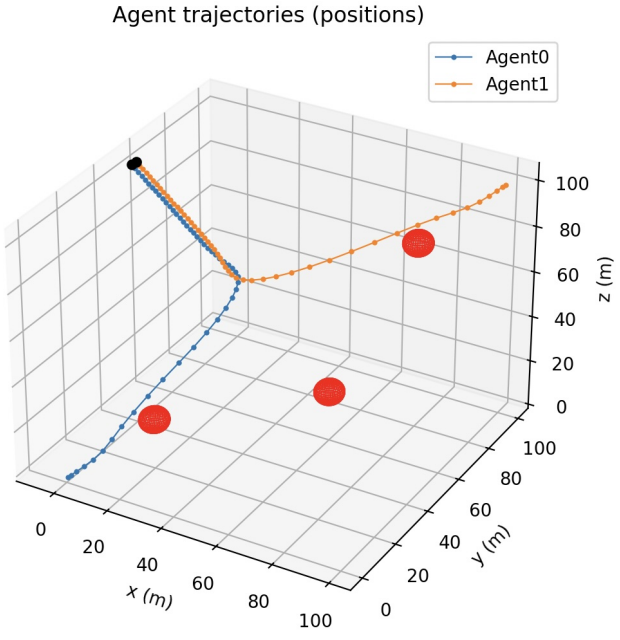

Fig. 18. Distributed MPC results for Case 3.

2) Distributed MPC results for case 3: Figure 18 presents the Case 3 position trajectory results for the distributed MPC architecture. In this scenario, both agents are tasked with approaching one another and remaining within a prescribed proximity threshold. The results show the two trajectories converging and effectively merging into a common path, after which the agents maintain close separation. This coupled motion arises from the double-integrator dynamics: once relative position and velocity are regulated, the agents tend to co-move along a shared trajectory. Figure 19 shows the plotted relative

position between the two agents for Case 3. The total control effort used in Case 3 with distributed MPC was 7.5 kN.

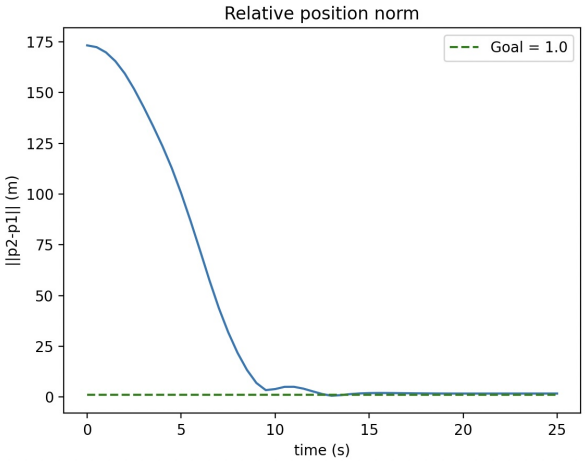

Fig. 19. Relative position between two agents using distributed MPC architecture for case 3.

VII. Failures, Limitations and Lessons Learned

A. Decentralized MPC

Though decentralized MPC can be less computationally expensive than distributed MPC, this advantage comes at the cost of comparatively suboptimal performance. This performance gap arises from two primary factors: (a) stochasticity and uncertainty and (b) information sharing latency and synchronization. With respect to the former, dynamical uncertainty in the form of disturbances,

$$W \in \mathbb{R}^{T \times n_x},$$

Regarding latency and synchronization, as previously discussed, the Jacobi method restricts information sharing to occur only after all agents have moved, which does not guarantee collision avoidance for agents that move later. The Gauss–Seidel method attempts to mitigate this by sharing information after each individual agent moves. However, neither method accounts for variability in agents' OCP solve times. When solve times differ, information exchange can become desynchronized or stall, leading to suboptimal performance. This limitation can also be avoided through a distributed architecture.

B. Distributed MPC

There are some limitations in the implementation of distributed MPC. In this architecture all the agents are considered together in the optimization. For a single agent, in our case, the state vector is 6×1 and the control input vector is 3×1; ∴, as the number of servicer agents increases, the stacked state and control vectors grow proportionally and can become quite large, despite simplifying assumptions in our dynamics. As a result distributed MPC is more difficult to scale in complexity while still providing a feasible solution efficiently.

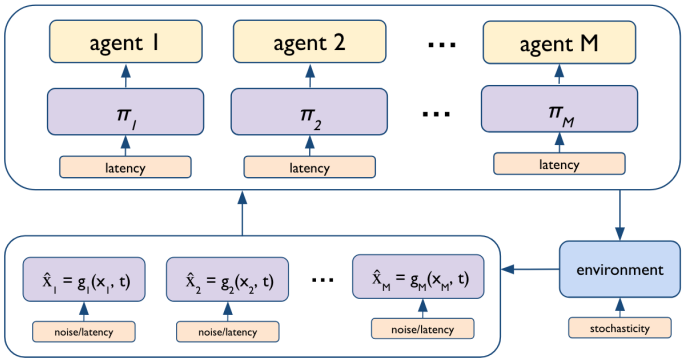

Fig. 20. Sources of latency, stochasticity, and uncertainty that hinder the performance of decentralized approaches.

One difficulty found in our experimentation was in enforcing individual agent goals, since each agent's states and controls are solved for an optimal solution. When Agent 0 was given an individual goal of staying stationary (Case 2), it tended to move towards the other agents to increase optimality. The weight (Q) on costs associated with individual agent's goal had to be scaled significantly to mitigate this behavior. Additionally, with large numbers of agents, obstacle avoidance constraints were often violated, since the optimization problem was so complex.

VIII. Conclusion

We explored the use of MPC as an enabling control framework for *Cooperative RPO (CoRPO)* in which multiple spacecraft share state information and actively coordinate their proximity operations. We evaluated 3 RPO scenarios (Cases 1–3) using a simplified 3-DoF double-integrator model and compared two decentralized MPC update schemes (Jacobi and Gauss–Seidel) against a centralized distributed MPC formulation. Across all approaches, performance was assessed using total commanded thrust as a proxy for fuel consumption.

The simulation results demonstrate that distributed MPC consistently reduced total thrust relative to decentralized MPC across all three cases. As summarized in table II, distributed MPC achieved lower team-level control effort in each scenario, with the largest improvement occurring in Case 3, where strong coupling between agents is required to satisfy mutual rendezvous and proximity objectives under constraints. Beyond reduced thrust, distributed MPC exhibited qualitatively improved closed-loop behavior in tightly coupled interactions: whereas decentralized MPC in Case 3 showed oscillatory convergence and repeated overshoot before rendezvous. The distributed formulation produced smoother convergence and stable proximity maintenance once the agents synchronized.

These findings also highlight important architectural trade-offs. Decentralized MPC is attractive for its lower per-agent computational burden and reduced reliance on a single point of failure, but its performance is limited by delayed and/or

inconsistent information sharing, particularly when solve times vary across agents. This can degrade collision-avoidance robustness and lead to oscillatory behavior in coupled tasks. Conversely, distributed MPC naturally captures inter-agent coupling, keep-out zones, and obstacle constraints within a single optimization, which improves coordination and fuel efficiency; however, it scales poorly as the number of agents grows because the stacked state and input dimensions increase linearly with group size and the constraint set grows rapidly.

TABLE II
COMPARISON OF DECENTRALIZED AND DISTRIBUTED MPC FOR TOTAL THRUST

	Decentralized MPC	**Distributed MPC**
Case	*Total Thrust [kN]*	*Total Thrust [kN]*
Case 1	14.6	8.2
Case 2	16.4	9.0
Case 3	29.2	7.5

Overall, the results support the central premise of CoRPO: shared state information and joint optimization can reduce control effort and improve closed-loop behavior during proximity operations. Future work should extend this study beyond the simplified double-integrator model to full 6-DoF relative dynamics, incorporate sensing and actuation uncertainty (e.g., communication latency delays, state-estimation errors, and stochasticity). Also, integrate formal safety mechanisms such as CBFs and RTA. Finally, studying hierarchical, clustered, or hybrid distributed–decentralized MPC variants may be promising to show further efficiency enabling close to real-time performance for larger multi-agent groups.

REFERENCES

[1] David A. Barnhart et al. "Enabling Safe Efficient Rendezvous: The Value of Cooperative and Communicative RPO". In: 75th International Astronautical Congress (IAC). Milan, Italy, 2024

[2] Adarsh Rajguru et al. "Enabling Safe Rendezvous and Proximity Operations for In-Space Docking Between Two Actively Maneuvering Satellites". In: AAS/AIAA Space Flight Mechanics Meeting. AAS 25-440. 2025.

[3] Adarsh Rajguru, Dave Barnhart, and Sami Haq. Application of a single sensor using a Perspective and Point algorithm to enable safe rendezvous and proximity operations between two actively maneuvering satellites. Pre-print. 2025.

[4] Shanky Sharma et al. "SPEED+: Next-generation dataset for spacecraft pose estimation across domain gap". In: arXiv pre-print arXiv:2110.03101 (2021). Available at https://arxiv.org/abs/2110.03101. url: https://arxiv.org/abs/2110.03101.

[5] Yanquan Zhang et al. "Trajectory optimization for spacecraft autonomous rendezvous and docking with compound state-triggered constraints". In: Aerospace Science and Technology 127 (2022), p. 107733. issn: 1270-9638. doi: 10.1016/j.ast.2022.107733. url: https://www.sciencedirect.com/science/article/pii/S1270963822004072.

[6] Christopher M. Jewison. "Guidance and Control for Multi-Stage Rendezvous and Docking Operations in the Presence of Uncertainty". Available at https://dspace.mit.edu/handle/1721.1/112362. PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 2017. url: https://dspace.mit.edu/handle/1721.1/112362.

[7] C. Pirat et al. "Toward the Autonomous Assembly of Large Telescopes Using CubeSat Rendezvous and Docking". In: Journal of Spacecraft and Rockets (2021), pp. 1–13. doi: 10.2514/1.A35072.

[8] The Consortium for Execution of Rendezvous and Servicing Operations (CONFERS). CONFERS Lexicon. https://satelliteconfers.org/confers-lexicon/. Accessed July 19, 2025. Lexicon maintained by CONFERS Technical Committee; referenced in CONFERS Resources and official documents. 2024

[9] Lecture 7.3 - The Jacobi and Gauss-Seidel Iterative Methods, course lecture notes, University of Notre Dame. Available: https://www3.nd.edu/ zxu2/acms40390F12/Lec-7.3.pdf

[10] W. Fehse. Automated Rendezvous and Docking of Spacecraft. 1st. Vol. 16. Cambridge Aerospace Series. Definitive reference on spacecraft rendezvous and docking. Cambridge, UK: Cambridge University Press, 2003, p. 516.

[11] Y. Xie et al. Guidance, Navigation, and Control for Spacecraft Rendezvous and Docking: Theory and Methods. 1st. E-book (PDF/EPUB) version. 495 pp. Cham, Switzerland: Springer Nature Singapore, 2021

[12] Cristobal Garrido and David Barnhart. "Fuel-Time Trade Space Analysis for the Rendezvous Problem: How to Establish a Common Ground for the Different RPO Procedures". In: Proceedings of the AAS/AIAA Astrodynamics Specialist Conference. (Preprint) AAS 25-782. Marina del Rey, CA, USA, 2025.

[13] Ariba, Y., Arzelier, D., Urbina, L. S., & Louembet, C. (2016). V-bar and R-bar glideslope guidance algorithms for fixed-time rendezvous: A linear programming approach. *IFAC-PapersOnLine, 49*(17), 385–390. https://doi.org/10.1016/j.ifacol.2016.09.066

APPENDIX A
PHASES OF RPO

Far Rendezvous Phase 0: The process of RPO begins at the far field, where the servicer could be 100s of kms away from the target. In this phase, relative navigation doesn't happen because the onboard sensors for target acquisition cannot resolve the target. The dynamics of both spacecraft are governed by Keplerian orbits about Earth. The relative motion can be captured by linearized equations of motion about a reference orbit. This is where the Hill-Clohessy-Wiltshire (HCW) or Tschauner-Hempel (TH) models are valid. These models describe how a servicer moves with respect to a client when both are under the same central gravity field but have slightly different orbital elements. When the servicer is approaching from $\sim$ 100 km to 50 - 20 km, we can use either HCW or TH dynamics augmented with low-order perturbations (example J_2, drag) for realism.

Near Rendezvous Phase 1a: Near Rendezvous Phase is where relative navigation typically begins from a range of $\sim$ 50 km - 20 km. In Figure 21 $\rho_r < 50$ km and the servicer attempts to rendezvous with the target, but only has angular measurements available [6]. In this phase relative motion using HCW or TH models are also applicable since Keplerian orbital dynamics are dominant.

Proximity Phase 1b: The Proximity Phase begins when the servicer spacecraft is approximately 1–2 km from the client. This range corresponds to the operational limits of current state-of-the-art ranging sensors, such as LiDAR systems, which can reliably resolve the relative distance to the target. As illustrated in Figure 3, the blue circle denotes the safety boundary or safety ellipsoid within which the servicer acquires valid range measurements and initiates the maneuvers required to transition into the docking phase [6].

In this regime, the mutual relative dynamics dominate, as gravitational effects become nearly uniform across the local

reference frame. The gravitational differential accelerations predicted by the HCW model are negligible compared to control accelerations, and the relative motion equations effectively reduce to rigid-body kinematics expressed in the Local-Vertical-Local-Horizontal (LVLH) frame. ∴ the environment can be approximated as quasi-inertial, allowing the spacecrafts to be modeled as free-flyers with 3-DoF.

Docking / grapple Phase 2: The docking / grappling phase begins $\sim$ 50 m from client and the docking may begin from $\sim$ 3 - 2 meters from the docking port. In this phase constraints are activated. During docking, we only integrate the relative translational and rotational dynamics, in the LVLH frame. Fig. 9 shows the approach and docking cones near the client.

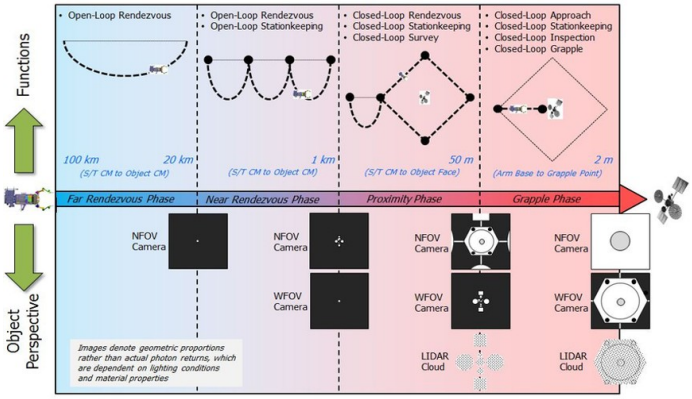

Fig. 21. All the phases of RPO

APPENDIX B
RAW DATA: CONTROL SUMMARIES FOR DISTRIBUTED MPC SCENARIOS

TABLE III
CASE 1: CONTROL EFFORT SUMMARY

Agent	$\sum_k \|F_k\|$ (N)
Agent 0	4129.925
Agent 1	4139.043
Team Total	**8268.969**

TABLE IV
CASE 2: CONTROL EFFORT SUMMARY

Agent	$\sum_k \|F_k\|$ (N)
Agent 0	5.414
Agent 1	5366.219
Agent 2	3645.768
Team Total	**9017.401**

TABLE V
CASE 3: CONTROL EFFORT SUMMARY

Agent	$\sum_k \|F_k\|$ (N)
Agent 0	3756.471
Agent 1	3756.731
Team Total	**7513.202**

APPENDIX C
ADDITIONAL CASE 3 FOR DISTRIBUTED MPC WITH 20 AGENTS

An additional Case 3 simulation was conducted with 20 agents, as shown in Fig. 22. The servicer agents are initialized on a circle surrounding the lead agent (Agent 0). All servicers converge toward Agent 0 and subsequently synchronize their motion, in nearly overlapping trajectories. After synchronization, the agents continue to satisfy the prescribed proximity constraint for the remainder of the maneuver, shown in Fig. 23, demonstrating the constraint-enforcement capability of MPC.

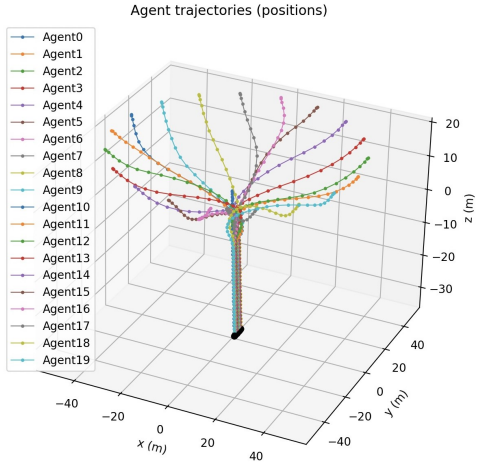

Fig. 22. Distributed MPC results for Case 3 with 20 agents.

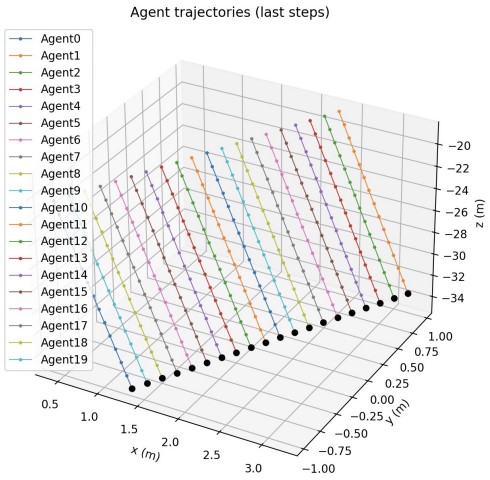

Fig. 23. Case 3 results with 20 agents after final rendezvous