

ADVANCING SIMULATION FOR IN-SPACE MISSIONS WITH A 12 DEGREES-OF-FREEDOM ROBOTIC TESTBED

Anirudh Chhabra*, Aaron Borger[†], and Donghoon Kim[‡]

Recently, the number of space missions has increased rapidly, increasing interest in in-space servicing, assembly, and manufacturing missions. As a result, Hardware-In-the-Loop (HIL) simulations of in-space missions have become a popular area of research. In previous works, the authors proposed a novel HIL spacecraft motion emulator design that offers several advantages over existing robotic simulators. Since the proposed simulator's uniqueness and advantages lie in its ability to exploit its kinematic redundancy, this paper proposes developing a software framework for the simulator that allows multiple inverse kinematics algorithms to be designed and implemented. Furthermore, several feature additions have been made to the simulator, which are discussed along with their advantages.

INTRODUCTION

Recently, the number of space missions has increased, leading to a rise in interest in In-Space Servicing, Assembly, and Manufacturing (ISAM) missions for applications, such as increasing the lifespan of existing spacecraft, debris collection and removal, and manufacturing for deep-space missions.¹ As a result, Hardware-In-the-Loop (HIL) simulations of in-space missions have become a popular area of research.² HIL simulations enable the prediction of potential issues that may arise in space due to factors, such as unmodeled dynamics, improper contact, and control algorithm limitations. Furthermore, HIL simulations allow for a deeper analysis of control algorithms, especially techniques utilizing artificial intelligence-based approaches, along various metrics, such as stability, accuracy, and robustness.³ Additionally, HIL simulators offer the ability to study contact dynamics, which is highly significant, as improper contact can lead to the failure of mission objectives.⁴

In the past, several different HIL simulators have been proposed and can be categorized based on their underlying operating mechanisms. These include orbital micro-gravity systems, sub-orbital micro-gravity systems, neutral buoyancy facilities, air-bearing systems, suspension systems, and ground robotics-based simulators². Orbital micro-gravity systems offer the most capabilities as they are operated in space but can be quite expensive to operate and maintain.^{5,6} Furthermore, such facilities are available for limited periods, reducing the number of experiments that can be conducted. Neutral buoyancy facilities are quite popular and also used for astronaut training are not preferred for space robotic simulations due to factors, such as limited power supply and the uncertain

*Graduate Student, Department of Aerospace Engineering & Engineering Mechanics, University of Cincinnati, Cincinnati, OH 45221.

[†]Graduate Student, Department of Aerospace Engineering & Engineering Mechanics, University of Cincinnati, Cincinnati, OH 45221.

[‡]Assistant Professor, Department of Aerospace Engineering & Engineering Mechanics, University of Cincinnati, Cincinnati, OH 45221.

fluid dynamics of the environment.⁷ On the other hand, air-bearing systems offer limited motion capabilities (2 Dimensional (2D) translation and yaw rotation, i.e., 3 Degrees-Of-Freedom (DOF)) and are also very expensive.⁸ Furthermore, they have low endurance as they require pressurized gas cylinders that need frequent refilling. Suspension-based simulator systems, such as NASA’s Active Response Gravity Offload System (ARGOS), offer increased complexity and therefore are not suitable for lab environments. They also provide low accuracy due to the difficulty in controlling the dynamic behavior of suspension gantries.⁹

Ground robotics-based simulators generally offer advantages over other simulator mechanisms due to their wide range of motion and flexible control mechanisms.^{8,10,11} However, most of the current systems depend on serial manipulators, which are known for low accuracy and often respond slowly to high-frequency commands. Overall, none of the existing systems simultaneously offers large motion ranges, accuracy, and cost-effectiveness. Therefore, in previous works, the authors have proposed a novel design for a 6 DOF HIL spacecraft motion emulator that offers several advantages over existing robotic simulators, such as wide motion ranges, improved accuracy, fast response, and flexibility provided by kinematic redundancy.^{11,12} It achieves this by employing a hybrid design using a combination of a 6 DOF robotic arm placed atop a 6 DOF Stewart platform. In the past, studies based on numerical simulations¹¹ and hardware experimentation were conducted to validate the system’s capabilities. However, certain issues were observed in the current design.

This research focuses on enhancing the capabilities of the spacecraft simulator’s design and functionality through hardware and software enhancements. Key upgrades include the integration of a 6-axis Force/Torque (FT) sensor to measure contact forces and torques, enabling better analysis of control algorithm performance. Spacecraft mock-ups have also been redesigned to align with the payload capacities of the robotic platforms. The software framework has been redeveloped in Python, replacing the open-source MoveIt package to support seamless integration of custom control algorithms, Inverse Kinematics (IK), and trajectory planning. The new framework employs velocity-based controllers instead of position-based ones, ensuring more robust control. Additionally, the redesigned framework facilitates the easy implementation of multiple IK algorithms leveraging the simulator’s kinematic redundancy. To summarize, this research enhances the current HIL simulator by incorporating hardware and software upgrades that support spacecraft simulation using diverse approaches. These advancements aim to provide accurate, flexible simulations for robust testing and validation of spacecraft control algorithms under realistic conditions, improving the performance of autonomous spacecraft system in complex environments.

This paper is structured as follows: First, the characteristics of the existing simulator are described, and its limitations and challenges are highlighted. Based on these challenges, the proposed hardware and software enhancements are introduced, underlining their advantages. Then, to validate the functionality of the HIL simulator, an experimental study is conducted where the hexapod emulates a servicer spacecraft and the arm represents a space manipulator aiming to insert a module into the target satellite, which is undergoing tumbling motion. Based on the experiment’s results, certain aspects of the HIL simulator are discussed along with potential future improvements.

BACKGROUND AND PRELIMINARIES

Simulator Description

The simulator is a robotic system composed of a 6 DOF *WidowX 250 Mobile* robotic arm by Trossen Robotics¹³ mounted on a 6 DOF Stewart platform, the *H.820 Hexapod* by Physik Instru-



(a) 12 DOF Motion Emulation System.



(b) 6 DOF Spacecraft with 6 DOF Manipulator.

Figure 1: Modularity of the HIL Simulator.

mente.¹⁴ These platforms were selected primarily for their compatibility with a common and stable control environment using the Robot Operating System (ROS). ROS provides flexibility for future iterations of the design, enabling the addition of components, vision sensors, force/torque sensors, and more. Furthermore, the chosen Stewart platform offers highly accurate position control, which is critical for spacecraft motion emulation.

The Vicon's Motion Capture System (MCS) is an integral part of the HIL simulator. This system measures real-time motion data, including position and velocity, of simulator components, such as satellite motion. With its high accuracy and an impressive data acquisition rate of 300 Hz, MCS delivers precise relative motion information. This data typically mimics the output from vision or ranging systems installed on deputy spacecraft during collaborative space-servicing missions.

The proposed simulator also exhibits modularity in its applications, representing various scenarios. In one scenario, the combination of the hexapod and robotic arm forms a 12 DOF redundant system, capable of emulating the 6 DOF motion of a spacecraft attached as the end-effector of the combined system. This configuration significantly enhances motion capabilities, particularly in translation and rotation, due to the system's inherent kinematic redundancy.

In another scenario, the hexapod represents a spacecraft while the robotic arm represents a space manipulator. This setup allows simultaneous simulation of both the spacecraft and the manipulator, enabling the emulation of the coupled motion of a servicer spacecraft. Such a configuration is particularly valuable for testing control methods for traditional serial-link robotic arms in space missions, where the hexapod replicates 6 DOF satellite motion.

These versatile scenarios underscored the simulator's ability to support diverse applications. Figure 1 shows the scenarios using the proposed HIL simulator.

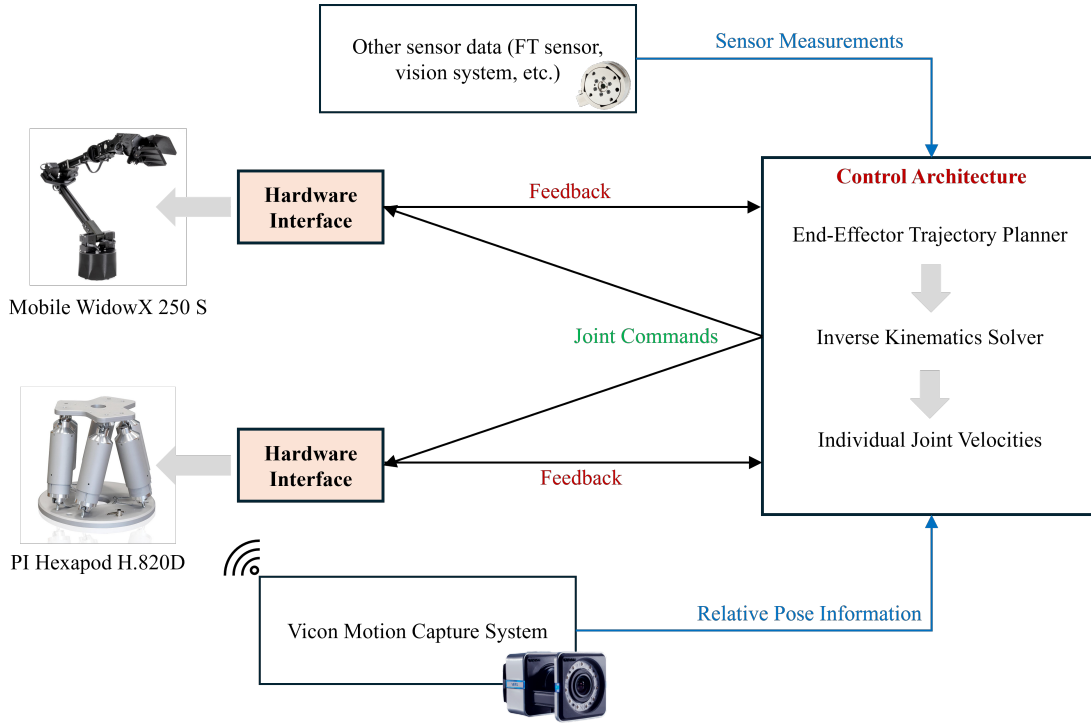


Figure 2: Control Architecture of the HIL Simulator.

Control Environment

The control environment for the simulator is developed in ROS using C++ and Python. In prior work, the independent support libraries for each robotic system were combined into a single control software for the combined robotic system using ROS MoveIt.¹⁵ The MoveIt package enables robust IK-based motion planning for the robotic system’s end-effector. While it includes basic IK solvers within its supported libraries, it also provides the capability to design custom IK solvers for the robotic system. However, building custom methodologies for individual tasks, such as IK, control algorithms, and trajectory planning, can be challenging as they must conform to the existing MoveIt framework. Furthermore, the previous approach lacked straightforward methods for implementing velocity-based IK.

Velocity-based control is crucial for most space missions, where robotic tasks require maintaining a low relative velocity between different space systems (cooperative or non-cooperative).¹⁶ This is usually not possible without a velocity-based control method. Consequently, the control framework was redeveloped from scratch using easy-to-follow Python-based programs. Finally, the control environment for the HIL simulator is illustrated in the architecture shown in Figure 2.

SYSTEM ENHANCEMENTS

Hardware Improvements

In the previous iteration of the HIL simulator, the robotic arm’s payload capacity (250 g) proved insufficient for continuous operation, often leading to the actuator overload. To address this, the payload weight was significantly reduced by utilizing a lighter and thinner 3D-printed mock-ups

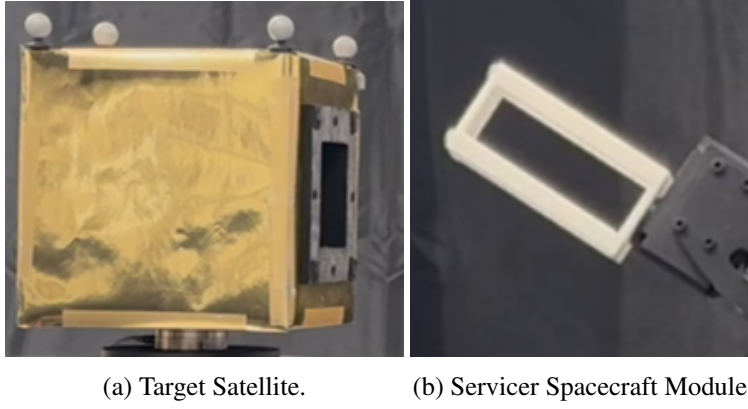


Figure 3: Experimental Mock-ups.

instead of using solid structures with high infill. This adjustment decreased the payload from approximately 100 g to 17 g, mitigating overload issues during continuous use.

The limited payload capacity also posed challenges for installing a 6 DOF FT sensor to measure contact forces and torques during mission emulation. This was resolved by placing the FT sensor at the base of the target satellite, ensuring accurate force/torque measurements. Additionally, the satellite design received an aesthetic upgrade, incorporating gold foil. Aside from enhancing the visual appeal, this modification enabled validation of the MCS's tracking performance in the presence of reflective surfaces. Figure 3 shows the two sets of mock-ups for the target satellite and the servicer spacecraft module.

In the previous iterations, the target satellite's motion information was directly available to the servicer spacecraft, representing a cooperative servicing scenario. The current setup incorporates the MCS to obtain the target satellite's motion information relative to the servicer, simulating a non-cooperative scenario.

The MCS provides highly accurate motion data using reflective markers installed on objects. Initially, four cameras were used, but the need for more cameras became evident as close-proximity missions often resulted in occluded markers, causing motion data loss at critical moments. To address this, the number of cameras was increased to 10, and the total number of markers tracked simultaneously was reduced. This minimized occlusions and marker merging, particularly in confined 3D spaces.

Markers were strategically repositioned away from contact locations, such as the module installation port, and instead placed on the base of the arm and the top of the satellite, where the chances of occlusion are lower. Figure 4 compares the two marker configurations.

Software Enhancements

The detailed control architectures for the hexapod and robotic arm are described as follows. It is worth noting that the ROS nodes for the Vicon's MCS are publicly available and can be seamlessly integrated into any ROS-based software.¹⁷ Therefore, the integration of Vicon ROS nodes is not discussed further in this paper.

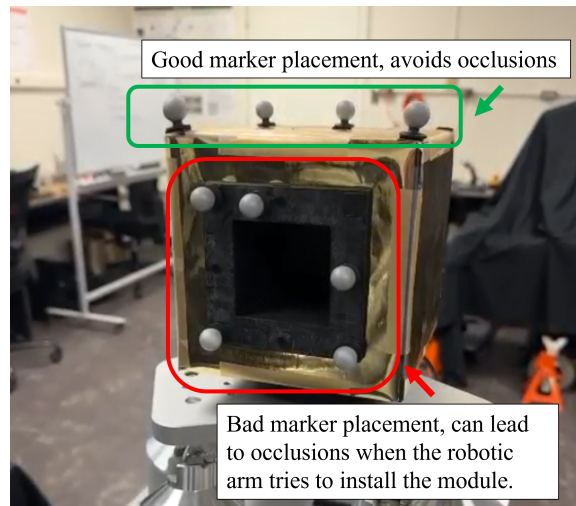


Figure 4: Reflective Marker Placement.

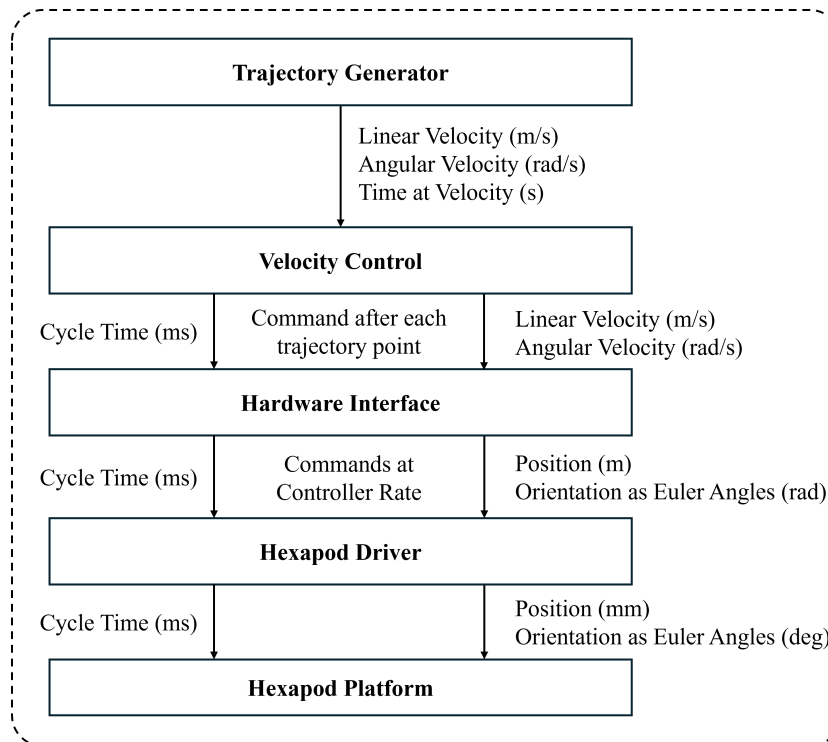


Figure 5: Hexapod Control Architecture.

Hexapod Control Architecture The Hexapod Control Architecture includes five layers, ordered from the highest to the lowest level: Trajectory Generation, Velocity Control, Hardware Interface, Hexapod Driver, and Hexapod Platform (developed by Physik Instrumente). Figure 5 describes these layers and data shared between them.

Trajectory Generation Layer consists of two components: one generating trajectories for the target satellite to emulate its motion and another for the servicer system (combined hexapod and

robotic arm). Although satellite motion is inherently non-linear, it is approximated here with constant velocity, periodically reversing direction.

Satellite motion can be defined as follows. Let $\dot{q}_{Si}$ be the velocity command for axis i about the center of the satellite, t the current time, v_i the speed for axis i , $\lfloor \cdot \rfloor$ the floor function, and T_i the period for axis i . The velocity of the satellite's center in its local frame, $\dot{q}_S \in \mathcal{R}^{6 \times 1}$, is expressed as:

$$\dot{q}_{Si}(t) = (-1)^{\lfloor t/T_i \rfloor} v_i, \quad (1)$$

where $\gamma = \lfloor t/T_i \rfloor$. The velocity of the hexapod platform, $\dot{q}_H$, is then calculated as:

$$\dot{q}_H = \begin{bmatrix} \dot{q}_{Sx} + d\omega_y \\ \dot{q}_{Sy} - d\omega_x \\ \dot{q}_{Sz} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad (2)$$

where $\dot{q}_{Sx}$, $\dot{q}_{Sy}$, and $\dot{q}_{Sz}$ are the translational velocities with respect to the center of the satellite mock-up, ω_x , ω_y , and ω_z are the angular velocities with respect to the center of the satellite mock-up, and d is the distance between the hexapod platform and the center of the satellite mock-up along the z -axis. Note that the satellite mock-up and hexapod platform are aligned on the x and y axis.

Servicer trajectories are generated to minimize the relative velocity between the satellite and servicer, assuming the satellite pose is estimated using a perception system, such as a 6 DOF pose estimation method or the MCS. The Servicer Trajectory Generator is a ROS node subscribing to the estimated satellite pose, calculates the velocity command with a proportional controller, and then publishes the velocity command to the Hardware Interface Node. The velocity command is calculated with the following:

$$\dot{q} = k_p e, \quad (3)$$

where $\dot{q} \in \mathcal{R}^{6 \times 1}$ is the velocity command, k_p is the proportional gain, and $e \in \mathcal{R}^{7 \times 1}$ is the error vector, computed as:

$$e = \begin{bmatrix} x_d - x_c \\ y_d - y_c \\ z_d - z_c \\ \beta_d \otimes \beta_c^* \end{bmatrix}, \quad (4)$$

where x , y , and z with the subscripts d and c are the desired and current translation vectors, respectively. Also, $\otimes$ is the quaternion multiplication operator, $\beta \in \mathcal{R}^{4 \times 1}$ is the quaternion, and $\beta^* \in \mathcal{R}^{4 \times 1}$ is the conjugate of β . The desired translation and orientations are set equal to the estimated translation and orientation of the satellite mock-up.

The Velocity Control Layer is a ROS node subscribing to velocity trajectory commands from the Trajectory Generation Layer. Each trajectory includes a series of 6 DOF velocities and the duration in seconds to move at each velocity. The next velocity point in the trajectory is published to the Hardware Interface after the duration for that point.

The Hardware Interface Layer subscribes to 6 DOF velocity and cycle time commands from the Velocity Control Layer, then converts the velocity commands to pose, and cycle period commands

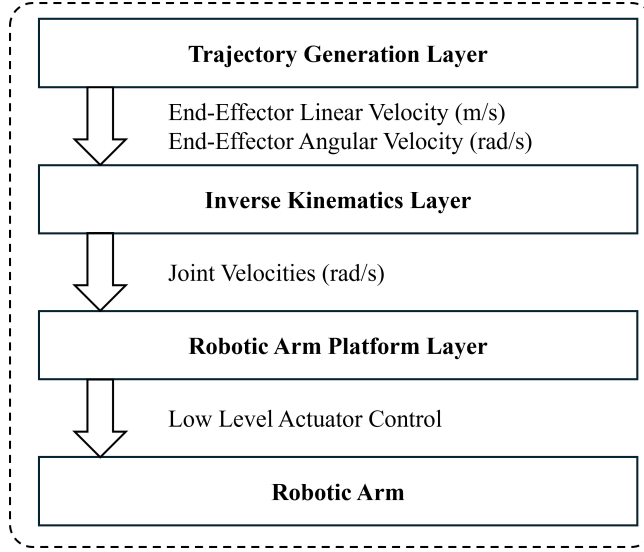


Figure 6: Robotic Arm Control Architecture.

which are transmitted to the hexapod at the controller rate with the Hexapod Driver Layer. The pose is calculated as follows:

$$\mathbf{q}_{t+dt} = \mathbf{q}_t + \dot{\mathbf{q}}_{t+dt}\tau, \quad (5)$$

where $\mathbf{q}_{t+dt}$ is the commanded pose that should be achieved at the end of the cycle, $\mathbf{q}_t$ is the current pose, and $\dot{\mathbf{q}}_t$ is the commanded velocity. The cycle period, denoted as τ , determines the extent to which the pose advances based on the commanded velocity. When τ matches the command rate (0.5 s), it can result in jittery motion due to potential communication delays in the velocity commands. To mitigate this, τ is set to 10 seconds in this work, based on experimental observations of smoother hexapod motion at this value.

It is important to note that the hexapod always updates to the latest command, meaning the previous pose ($\mathbf{q}_{t+dt}$) is overridden when a new command is received. As a result, the hexapod seamlessly handles updates without losing synchronization, provided $\tau > dt$, i.e., the command rate.

Rather than transmitting the pose command when the velocity command is received, the Hardware Interface Layer transmits the commands at the pre-determined Controller Rate (default of 25 ms) to allow the Velocity Control layer to only transmit to change to velocity and ensure smooth trajectories without stopping after achieving a pose.

The Hexapod Platform Layer¹⁸ provides a software interface to command the 6 DOF pose (position and orientation) and the amount of time to achieve the commanded pose in milliseconds. The Hexapod Platform Layer accepts the translation commands in units of millimeters, and rotational commands in units of degrees.

Robotic Arm Control Architecture The robotic arm control architecture includes three layers. The layers, in order of the highest level to the lowest level, are as follows: the Trajectory Generation Layer, the IK Layer, and the Robotic Arm Platform Layer developed by Interbotix. Figure 6 shows these layers and data shared between them.

The Trajectory Generation Layer is a ROS node written in Python that obtains the target and servicer spacecraft's pose information from the MCS, using it to calculate the relative pose between

the target and the end-effector. This relative pose information generates a velocity trajectory for the end-effector to reach the desired location in 3D space. This is one of the most important layers, as it handles the relative pose information between the current and desired positions of the end-effector by using data from various sensors, such as the MCS. The Trajectory Generation Layer then calculates the desired velocity for the end-effector as follows.

For a given position error, $[dx \ dy \ dz]^T$, between the target and current end-effector poses, the desired end-effector velocity at a given time is obtained as follows:

$$v_x = k_x dx, \quad (6)$$

$$v_y = k_y dy, \quad (7)$$

$$v_z = k_z dz, \quad (8)$$

where $k_i (i = x, y, z)$ represents the scaling factor, which is calculated as:

$$k_i = \min \left(\frac{v_{i,\max}}{\sqrt{dx^2 + dy^2 + dz^2}}, 1.0 \right), \quad \text{if } \sqrt{dx^2 + dy^2 + dz^2} > 0, \quad (9)$$

and $k_i = 0$ otherwise.

Similarly, for a given orientation error, $[d\theta_x \ d\theta_y \ d\theta_z]^T$, which represents the differences in roll, pitch, and yaw, respectively, between the target and current orientations, the desired angular velocity is calculated as follows:

$$\omega_x = k_{\theta_x} d\theta_x, \quad (10)$$

$$\omega_y = k_{\theta_y} d\theta_y, \quad (11)$$

$$\omega_z = k_{\theta_z} d\theta_z, \quad (12)$$

where the orientation errors are normalized to the range $[-\pi, \pi]$ to account for periodicity.

The scaling factor $k_{\theta_i} (i = x, y, z)$ is calculated as:

$$k_{\theta_i} = \min \left(\frac{\omega_{i,\max}}{|d\theta_i|}, 1.0 \right), \quad \text{if } d\theta_i \neq 0, \quad (13)$$

and $k_{\theta_i} = 0$ otherwise. Here, $\omega_{i,\max}$ represents the maximum allowable angular velocity for the respective axis, ensuring the desired angular velocity does not exceed system limits.

The IK Layer is a ROS node written in Python, responsible for calculating the joint rates based on the desired end-effector motion trajectory obtained from the Trajectory Generation Layer. This layer is designed to facilitate the development of custom IK algorithms as independent Python functions that can be integrated into the framework. These functions can be applied to different robotic platforms with only minor modifications.

To validate this approach, two different IK methods were implemented and integrated within the framework: the Jacobian Pseudoinverse method and the Damped Jacobian method.¹⁹ The Damped Jacobian approach was selected for this study due to its advantage of avoiding singularities as compared to the traditional Pseudoinverse method, which often leads to large joint angle changes near singularity, potentially causing collisions and/or damage in close-proximity operations.²⁰ The

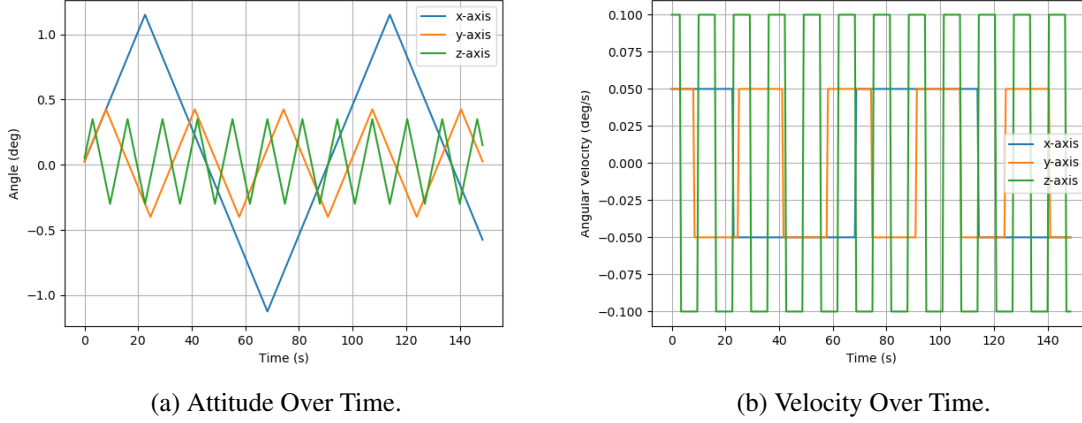


Figure 7: Satellite Behavior Metrics.

Damped Jacobian IK solver then calculates the desired joint velocities ($\dot{\mathbf{q}} \in \mathcal{R}^{n \times 1}$) using the n DOF manipulator Jacobian ($\mathbf{J} \in \mathcal{R}^{6 \times n}$) as follows:

$$\dot{\mathbf{q}} = \mathbf{J}^T (\mathbf{J} \mathbf{J}^T + \lambda \mathbf{I})^{-1} \mathbf{v}, \quad (14)$$

where, $\mathbf{I} \in \mathcal{R}^{n \times n}$ is the identity matrix, $\mathbf{v} \in \mathcal{R}^{6 \times 1}$ represents the desired 6 DOF velocity of the end-effector in the Cartesian coordinate frame, and λ is the damping factor.

The Robotic Arm Platform Layer, developed by Interbotix, is the lowest layer of this framework and provides a software interface to command the desired joint positions and velocities. It allows users to select the appropriate control mode (position or velocity) within its configuration files. Additionally, it enables the modification of PID control gains for low-level actuator control. It accepts joint angle commands in units of radians and joint velocity commands in units of radians per seconds, converting them to the corresponding actuator units of encoder counts and encoder counts per second.

EXPERIMENTAL STUDY

Experimental Setup

To demonstrate the functionality of the HIL simulator and validate the hardware and software improvements, a space servicing mission scenario is designed and chosen as the experimental case. A tumbling target satellite is emulated using the hexapod. Figure 7 shows the tumbling motion of the satellite mock-up, as defined by Eq. (1), with an angular velocity of $\mathbf{v} = [0 \ 0 \ 0 \ 0.3 \ 0.1 \ 0.2]^T$ deg/s and a period of $\mathbf{T} = [0 \ 0 \ 0 \ 5 \ 4 \ 1.5]^T$ s.

Additionally, while the hexapod representing the target satellite was controlled to follow a pre-defined trajectory, the hexapod representing the servicer spacecraft was controlled to minimize the relative angular velocity between the servicer and the target satellite using the relative pose information provided by the MCS.

On the other hand, the HIL simulator represents a 6 DOF space manipulator mounted on a servicer spacecraft, tasked with installing a module inside the tumbling target satellite. The target satellite is

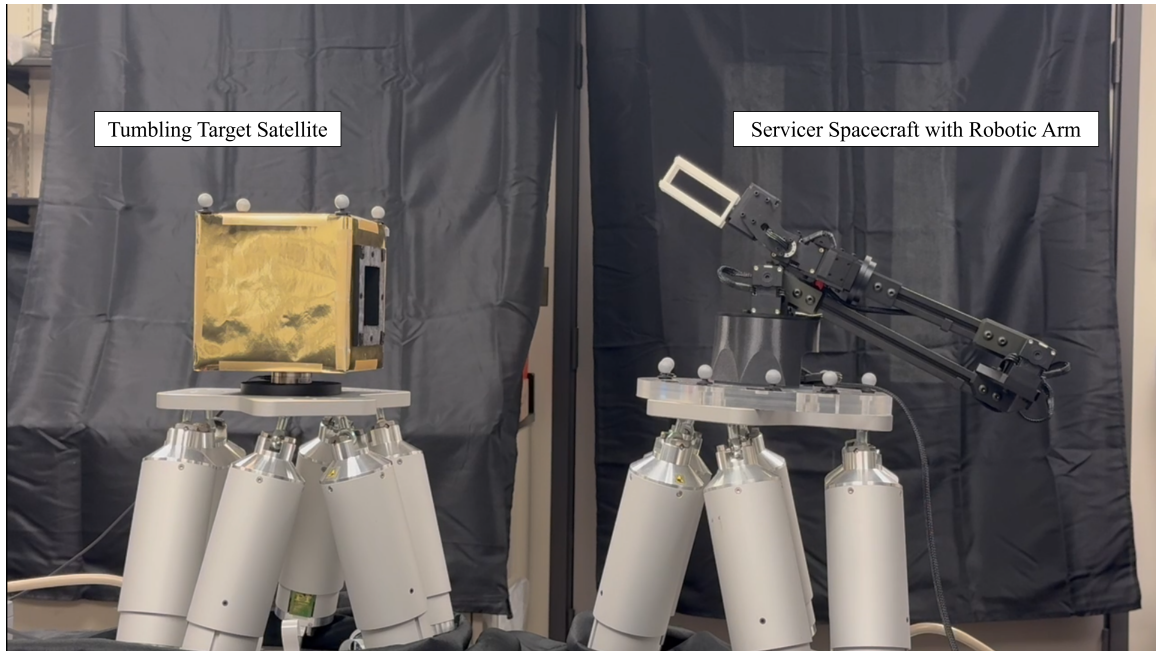


Figure 8: Experimental Setup.

a 1.5 U structure (a cube with dimensions of 150 mm) featuring a slot on its front face for module installation. The module is designed with dimensions slightly smaller than the slot to allow for a margin of error.

The two simulators are positioned 500 mm apart, with the robotic arm initially folded back into its resting state. The distance between the hexapod platform and the center of the satellite mock-up along the z -axis is 99.12 mm. The overall setup of the simulators is shown in Figure 8. This figure also shows the optimal placement of the reflective markers for the MCS, located on top of the target satellite and at the base of the robotic arm on the servicer spacecraft.

Since this mission involves installing a module into the target satellite, the trajectory planning for the robotic arm's end-effector is divided into two parts: the initial approach and the module installation. During the initial approach, the end-effector must reach in front of the docking port while maintaining the relative pose and velocity is above a specified threshold. Once the end-effector aligns the module's pose with the docking port and achieves a low pose and velocity error, the trajectory is updated to guide the module into the installation port. This approach ensures the module avoids unintended contact with target satellite. The relative criteria are defined as follows:

- Magnitude of Relative Translational Velocity Error < 0.01 m/s
- Magnitude of Relative Rotational Velocity Error < 0.05 rad/s
- Magnitude of Relative Orientation Error < 0.05 rad

Once these criteria are met, the trajectory is updated to move the module into the installation port.

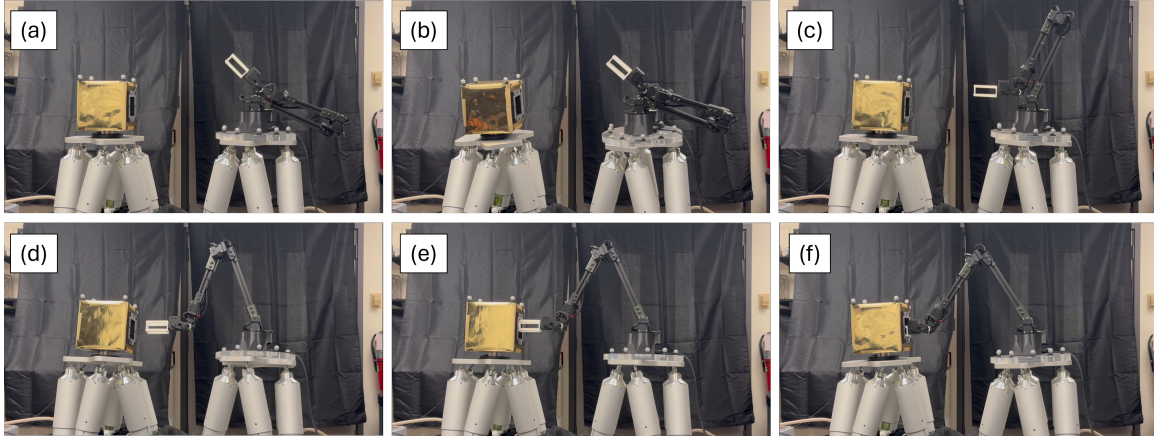


Figure 9: Sequence of the Target Satellite and the HIL Simulator.

Results and Discussion

Once the experiment is set up, the hexapod with the target satellite and the HIL simulator are connected to independent PCs, which then initiate the experiment. The overall operation of the target satellite and the HIL simulator, as described in the previous section, is shown in Figure 9, where the hexapod representing the servicer spacecraft tracks the motion of the target satellite, while the robotic arm aims to install the module into the port on the target satellite. As shown, the servicer spacecraft successfully installs the module into the port of the tumbling target satellite without collisions with the satellite body, following these steps:

- (a) The initial position, with the robotic arm in its resting state.
- (b) The servicer spacecraft begins reducing the relative motion before the robotic arm starts operating.
- (c) The robotic arm is activated once the servicer spacecraft successfully tracks the target satellite.
- (d) The robotic arm aligns the module with the port on the target satellite and waits for the relative velocity to be minimized.
- (e) The robotic arm begins moving the module into the installation port as the relative velocity decreases to acceptable levels.
- (f) The robotic arm successfully installs the module into the target satellite.

Figure 10 shows the attitude error of the servicer spacecraft while tracking the tumbling target satellite. As shown, the servicer spacecraft successfully tracks the motion of the target satellite, maintaining its Euler angles within 2 degrees of the desired values.

To further evaluate the capabilities of the platform, the motion tracking results of the robotic arm's end-effector motion are analyzed and shown in Figure 11. The figure demonstrates that the end-effector largely follows the desired Cartesian velocity obtained from the Trajectory Generation Layer, though the motion appears jittery. This behavior is attributed to the arm's payload issues, which induce occasional jumps, as well as the need to control the arm at a relatively slow pace.

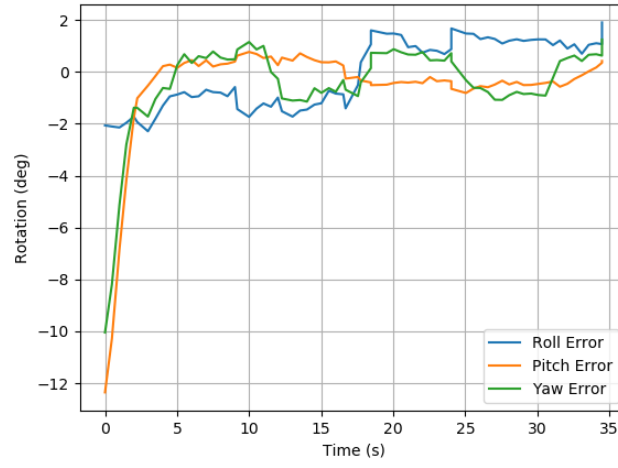


Figure 10: Satellite Tracking Error.

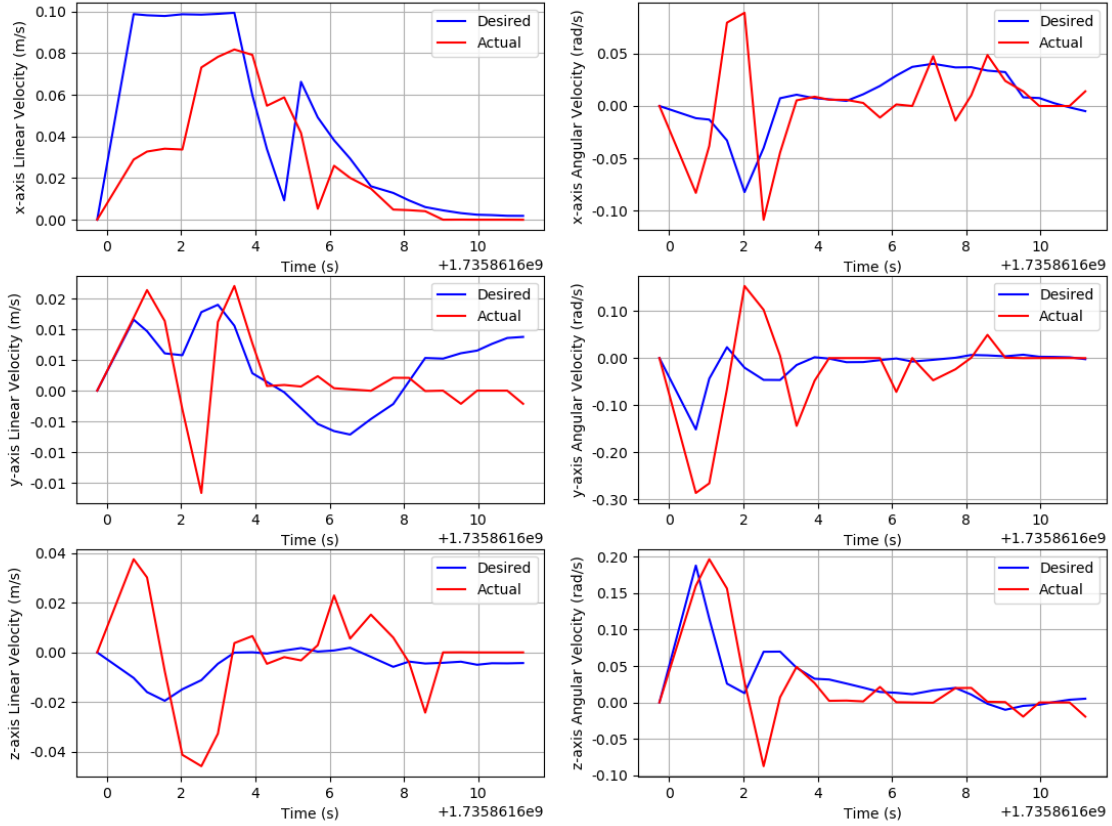


Figure 11: End-Effector Motion Tracking Results.

Around the 4-second mark, the arm stops its rotational motion and focuses on the translational motion, marking the transition to the second phase of the trajectory plan, where the module is inserted into the target port after the relative velocity threshold is met.

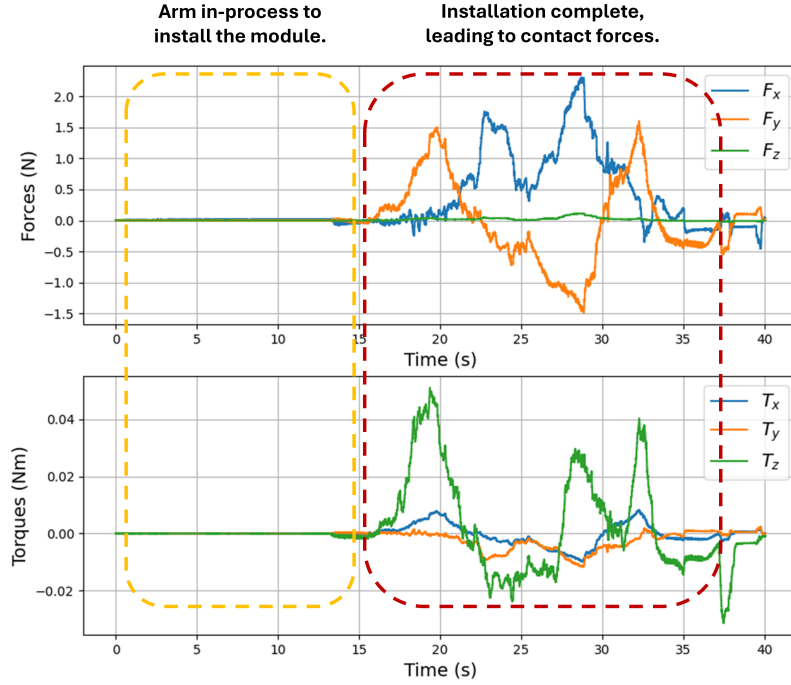


Figure 12: FT Measures during Operation.

Furthermore, the FT sensor installed at the base of the target satellite provides FT measurements over time, as shown in Figure 12. Notably, the FT sensor does not record any contact forces until the module installation is complete. This data serves as a valuable resource for implementing and validating force/impedance control algorithms in space servicing applications.

Overall, this experiment successfully demonstrates the functionality of the constructed HIL spacecraft simulator. The current study highlights the numerous advantages of using the proposed 12 DOF HIL simulator. Key benefits include its modularity, enabling it to function as a single 12 DOF system or as two coupled 6 DOF systems (satellite and manipulator). Furthermore, the revised control framework enhances the simulator's capabilities, offering easy implementation of custom trajectory planners, IK solvers, joint controllers, etc., while also providing real-time velocity-based control of the platform. This flexibility allows for the development and integration of complex control methods into the system.

In addition to validating the platform's capabilities, this study also highlights several areas for future improvement. Subsequent iterations will focus on installing and integrating a vision system using a depth camera to obtain relative pose information. This system offers several advantages over motion capture, including depth data, target identification, and localization without the need for reflective markers. Moreover, this approach will better replicate real-life conditions, where vision-based systems are typically used for pose estimation. Another significant improvement involves utilizing an enhanced IK method to enable robust control of the combined 12 DOF robotic platform. Additionally, incorporating force/torque data into the control algorithm to minimize contact forces is an important next step. Minor improvements, such as modeling more realistic relative motion between satellites and optimizing the PID response of the robot actuators, will also be considered.

CONCLUSION

This study validates the enhanced capabilities of the proposed HIL simulator in emulating complex space missions, such as installing a module on a tumbling target satellite. The experimental results demonstrate the system's ability to achieve trajectory planning, motion tracking, and collision-free module installation, underscoring the modularity and adaptability of the 12-DOF simulator. The study highlights the seamless integration of trajectory planners and Inverse Kinematics (IK) solvers, as well as the effectiveness of the velocity-based control framework in real-time operations. Additionally, the findings emphasize the simulator's potential for diverse applications, including testing advanced control strategies and developing custom components for space servicing missions. Future improvements, such as incorporating force/torque feedback into control algorithms, utilizing enhanced IK methods for robust manipulator control, and integrating vision systems for improved pose estimation, are expected to further advance the simulator's utility in addressing the demands of in-space servicing and assembly missions.

REFERENCES

- [1] D. Arney, J. Mulvaney, C. Williams, C. Stockdale, N. Gelin, and P. Le Gouellec, *In-space Servicing, Assembly, and Manufacturing (ISAM) State of Play 2023 Edition*. NASA, 2023.
- [2] M. Wilde, C. Clark, and M. Romano, "Historical survey of kinematic and dynamic spacecraft simulators for laboratory experimentation of on-orbit proximity maneuvers," *Progress in Aerospace Sciences*, Vol. 110, 2019, p. 100552.
- [3] G. Oliveira Pinho, G. Di Domenico, A. Morselli, F. Topputo, *et al.*, "An Overview of EXTREMA Project Pillar II: Validating Autonomous Guidance Algorithms via HIL Testing," *75th International Astronautical Congress (IAC 2024)*, 2024, pp. 1–11.
- [4] A. A. Ali, B. Beigomi, and Z. H. Zhu, "Development of 6DOF Hardware-in-the-Loop Ground Testbed for Autonomous Robotic Space Debris Removal," *Aerospace*, Vol. 11, No. 11, 2024, 10.3390/aerospace11110877.
- [5] D. Miller, A. Saenz-Otero, J. Wertz, A. Chen, G. Berkowski, C. Brodel, S. Carlson, D. Carpenter, S. Chen, S. Cheng, D. Feller, S. Jackson, B. Pitts, F. Perez, J. Szuminski, and S. Sell, "SPHERES: a testbed for long duration satellite formation flying in micro-gravity conditions," *Advances in the Astronautical Sciences*, Vol. 105, Jan. 2000.
- [6] T. Smith, J. Barlow, M. Bualat, T. Fong, C. Provencher, H. Sanchez, E. Smith, and T. Team, "Astrobee: A New Platform for Free-Flying Robotics Research on the International Space Station," *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, Jun. 2016.
- [7] C. R. Carignan, J. C. Lane, and P. J. Churchill, "Controlling robots on-orbit," *Proceedings 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (Cat. No.01EX515)*, 2001, pp. 314–319.
- [8] Y. Nakka, R. Foust, E. Lupu, D. Elliott, I. Crowell, S.-J. Chung, and F. Hadaegh, "Six Degree-of-Freedom Spacecraft Dynamics Simulator for Formation Control Research," *2018 AAS/AIAA Astrodynamics Specialist Conference*, 08 2018.
- [9] B. Dunbar, "JSC Engineering - Active Response Gravity Offload System,"
- [10] T. Boge, T. Wimmer, O. Ma, and T. Tzschichholz, "EPOS - Using Robotics for RvD Simulation of On-Orbit Servicing Missions," *AIAA Modeling and Simulation Technologies Conference*, 2010.
- [11] A. Chhabra and D. Kim, "Ground Robotic Platform for Simulation of On-Orbit Servicing Missions," *Journal of Aerospace Information Systems*, Vol. 19, No. 7, 2022, pp. 480–493.
- [12] A. Chhabra and D. Kim, "Development of a 12 Degrees-Of-Freedom Robotic Testbed for Experimental Analysis of In-Space Missions," *2023 AAS/AIAA Astrodynamics Specialist Conference*, August 2023, pp. 1–11. AAS 23-322.
- [13] T. Robotics, "WidowX 250 Mobile Robot Arm 6DOF," 2024.
- [14] P. Instrumente, "H-820 Datasheet," Jun 2021.
- [15] D. T. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study," 2014.
- [16] E. Papadopoulos, F. Aghili, O. Ma, and R. Lampariello, "Robotic Manipulation and Capture in Space: A Survey," *Frontiers in Robotics and AI*, Vol. 8, 2021.

- [17] M. Achtelik, “Vicon ROS Driver,” 2010.
- [18] PI-PhysikInstrumente, “Pi-physikinstrumente/pi_ros_driver: ROS driver for pi-hexapods,”
- [19] S. R. Buss, “Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods,” tech. rep., Department of Mathematics, University of California, San Diego, La Jolla, CA, USA, 2009.
- [20] Y. Nakamura and H. Hanafusa, “Inverse kinematic solutions with singularity robustness for robot manipulator control,” *Journal of Dynamic Systems Measurement and Control-transactions of The Asme*, Vol. 108, 1986, pp. 163–171.